

Autonomous Combat Robot

Senior Design Final Report

Design Team 11

Mackenzie Hawkins

Michael Hritz

Zachary Kilburn

Hong (Joey) Lau

Faculty Advisor: Dr. Robert Veillette

5/1/2018

Table of Contents

1.1.0 Need	1
1.2.0 Objective	1
1.3.1 Patent Search.....	2
1.3.2 IEEE Explore Article Search	3
1.3.3 Other Sources.....	4
1.4.0 Marketing Requirements.....	5
1.5.0 Objective Tree.....	6
2.0.0 Design Requirements Specification	7
3.0.0 Accepted Technical Design	9
3.1.0 Low Level Overview	9
3.1.1 FastTransfer Communications Protocol.....	15
3.2.0 Main Control Board	18
3.2.1 Main Control Board Hardware	18
3.2.2 Main Control Board Software.....	23
3.2.3 Main Control Board Schematic.....	28
3.2.4 Main Control Board Bill of Material	40
3.2.5 Main Control Board Implementation and Revisions	43
3.2.6 Main Control Neural Network Generation	172
3.2.7 Main Control Neural Network Generation Code	174
3.3.0 Motor Driver Board	179
3.3.1 Motor Driver Board Hardware.....	179
3.3.2 Motor Driver Board Software	184
3.3.3 Motor Driver Board Schematic	188
3.3.4 Motor Driver Board Bill of Material.....	201
3.3.5 Motor Driver Board Implementation Revisions	204
3.3.6 Motor Driver Board Code	207
3.4.0 Proximity Sensor Board	221
3.4.1 Proximity Sensor Board Hardware	221

3.4.2 Proximity Sensor Board Schematic	223
3.4.3 Proximity Sensor Board Software.....	227
3.4.4 Proximity Sensor Board Pseudo Code	231
3.4.5 Proximity Sensor Board Bill of Material	237
3.4.6 Proximity Sensor Board Implementation Revisions	238
3.4.7 Proximity Sensor Board Code.....	244
3.5.0 Weapon Motor Control Board	365
3.5.1 Weapon Motor Control Board Hardware.....	365
3.5.2 Weapon Motor Control Board Software	369
3.5.3 Weapon Motor Control Board Schematic.....	371
3.5.4 Weapon Motor Control Board Bill of Material.....	377
3.5.5 Weapon Motor Control Board Implementation Revisions	378
3.5.6 Weapon Motor Control Board Code	379
3.6.0 Mechanical Design.....	441
3.6.1 Mechanical Design Implementation and Revisions	446
3.7.0 Design Calculations	450
3.7.1 Main Control Board	450
3.7.2 Motor Driver Board	457
3.7.3 Weapon Motor Control Board	464
3.7.4 Proximity Sensor Board	474
4.0.0 Parts List	479
5.0.0 Design Schedule.....	492
6.0.0 Design Team Information	497
7.0.0 Conclusions and Recommendations	498
8.0.0 References.....	501
9.0.0 Appendices.....	503
9.0.1 FastTransfer Arduino Library	503

List of Figures

Figure 1: Objective Tree	6
Figure 2: Level 0 Functional Block Diagram	10
Figure 3: Level 0 Hardware Block Diagram	12
Figure 4: FastTransfer Packet Structure.....	16
Figure 5: Level 1 MCB Hardware Block Diagram.....	19
Figure 6: Level 2 MCB Hardware Block Diaagram.....	21
Figure 7: Level 0 MCB Software Block Diagram	23
Figure 8: Level 1 MCB Software Block Diagram Motor.....	24
Figure 9: Level 2 MCB Software Block Diagram	26
Figure 10: Main Control Board Comm. Ports Schematic	31
Figure 11: Main Control Board Communication Schematic	32
Figure 12: Main Control Board IMU Schematic	33
Figure 13: Main Control Board Indication & Settings Schematic.....	34
Figure 14: Main Control Board Inputs Schematic.....	35
Figure 15: Main Control Board Microcontroller Schematic.....	36
Figure 16: Main Control Board Micro-SD Schematic	37
Figure 17: Main Control Board Power Regulation Schematic	38
Figure 18: Main Control Board PCB Layout	39
Figure 19: MCB Competed Board	44
Figure 20: Neural Network Training Data Simulation Positioning	172
Figure 21: Neural Network Training Data Simulation Wall Distance Equation	173
Figure 22: Level 2 Motor Driver Board Hardware Block Diagram	181
Figure 23: Level 1 Motor Driver Board Software Block Diagram.....	184
Figure 24: Level 2 Motor Driver Board Software Block Diagram.....	186
Figure 25: Motor Driver Board Inputs Schematic	191
Figure 26: Motor Driver Board Monitors Schematic	192
Figure 27: Motor Driver Board Microcontroller Schematic.....	193
Figure 28: Motor Driver Board Micro SD Schematic.....	194
Figure 29: Motor Driver Board Communications Schematic.....	195
Figure 30: Motor Driver Board Encoder Decoder Schematic	196
Figure 31: Motor Driver Board PWM Generator Schematic.....	197
Figure 32: Motor Driver Board H-Bridge Driver Schematic	198
Figure 33: Motor Driver Board H-Bridge Schematic	199
Figure 34: Motor Driver Board PCB Layout.....	200
Figure 35: MDB Completed Board Front	206
Figure 36: MDB Completed Board Back.....	207
Figure 37: Level 1 Proximity Sensor Board Hardware Block Diagram	221
Figure 38: Proximity Sensor Board Inputs and Outputs Schematic	224
Figure 39: Proximity Sensor Board Communication Schematic	225
Figure 40: Proximity Sensor Board Microcontroller Schematic.....	226

Figure 41: Level 0 Proximity Sensor Board Software Block Diagram.....	227
Figure 42: Level 1 Proximity Sensor Board Software Block Diagram.....	229
Figure 43: PSB Pseudo Code Part 1.....	232
Figure 44: PSB Pseudo Code Part 2	233
Figure 45: PSB Pseudo Code Part 3	234
Figure 46: PSB Pseudo Code Part 4	235
Figure 47: Proximity Senor Board PCB Layout	236
Figure 48: PSB Completed Board	239
Figure 49: Revised Proximity Sensor Board Inputs and Outputs Schematic	240
Figure 50: Revised Proximity Sensor Board Communication Schematic	241
Figure 51: Revised Proximity Sensor Board Microcontroller Schematic	242
Figure 52: Revised Proximity Sensor Board PCB Layout	243
Figure 53: Level 1 Weapon Motor Controller Hardware Block Diagram	365
Figure 54: Level 2 Weapon Motor Control Hardware Block Diagram	366
Figure 55: Level 1 Weapon Motor Control Board Software Block Diagram	369
Figure 56: Weapon Motor Control Board Microcontroller Schematic	371
Figure 57: Weapon Motor Control Board Power Regulation Schematic.....	372
Figure 58: Weapon Motor Control Board I/O Schematic	373
Figure 59: Weapon Motor Control Board Debugging Schematic	374
Figure 60: Weapon Motor Control Board IGBT Driver Schematic	375
Figure 61: Weapon Motor Board PCB Layout.....	376
Figure 62: Completed WMCB	378
Figure 63: Robot Chassis	442
Figure 64: Weapon Suspension Bearing Block Model	443
Figure 65: Weapon Suspension Bearing Block Octagon Section Assembly Model.....	443
Figure 66: Weapon Model	444
Figure 67: Battery Enclosure Model	445
Figure 68: Robot Assembly Model.....	446
Figure 69: Power Panel Underside.....	447
Figure 70: Power Distribution Panel Implementation	448
Figure 71: MDB and WMCB Panel Underside.....	449
Figure 72: MDB & WMCB Panel Implementation.....	450
Figure 73: Robot Direct Scan.....	455
Figure 74: Graphical Interpretation of Direct scan	455
Figure 75: Graphical Interpretation of scanned information differentiated twice	455
Figure 76: Motor Driver Board Thermal Characteristics Simulation.....	458
Figure 77: MDB Single FET Power Chart	458
Figure 78: MDB Single FET Switching Power Waveform	461
Figure 79: MDB FET Power Turn On Spike Waveform	462
Figure 80: MDB FET Power Turn Off Spike Waveform.....	462
Figure 81: IGBT Gate Charging Simulation Results $V_g=10V$ $R_g=2\text{ohms}$	465
Figure 82: IGBT Gate Charging Simulation Results $V_g=10V$ $R_g=2\text{ohms}$	465

Figure 83: IGBT Gate Capacitance Charging Simulation Schematic.....	465
Figure 84: IGBT Gate Charging Simulation Results Vg=10V Rg=1.1ohms	466
Figure 85: IGBT Gate Charging Simulation Results Vg=18V Rg=2ohms	467
Figure 86: IGBT Voltage Regulator Output Capacitance Simulation Schematic.....	467
Figure 87: APXW005A0X Output Voltage vs Output Current	468
Figure 88: APXW005A0X Output Current vs. Output Voltage	468
Figure 89: IGBT Voltage Regulator Response Cs=10uF	469
Figure 90: IGBT Voltage Regulator Response Cs=20uF	470
Figure 91: IGBT Voltage Regulator Response Cs=100uF	470
Figure 92: IGBT Voltage Regulator Response Cs=2000uF	471
Figure 93: IGBT Voltage Regulator Response Cs=1000uF	471
Figure 94: Disassembled LIDAR Storage Code	478
Figure 95: Final Gantt Chart	492
Figure 96: Final Gantt Chart	493
Figure 97: Final Gantt Chart	494
Figure 98: Actual Gantt Chart	495
Figure 99: Actual Gantt Chart	496

List of Tables

Table 1: Level 1 Hardware Proximity Sensor Board Module	12
Table 2: Level 1 Hardware LIDAR Module.....	13
Table 3: Level 1 Hardware Hall Effect Sensor Module.....	13
Table 4: Level 1 Hardware Remote Receiver Module.....	13
Table 5: Level 1 Hardware Motor Driver Board Module	13
Table 6: Level 1 Hardware Main Control Board Module	14
Table 7: Level 1 Hardware Encoder Module.....	14
Table 8: Level 1 Hardware Weapon Motor Control Board Module.....	14
Table 9: Level 1 Hardware Power Switching Device Module.....	14
Table 10: Level 1 Hardware Current Sense Module	15
Table 11: Level 1 Hardware Left & Right Locomotion Motor Module.....	15
Table 12: Level 1 Hardware Weapon Motor Module	15
Table 13: Level 1 MCB Hardware Voltage Regulator Module	19
Table 14: Level 1 MCB Hardware Voltage Regulator Module	20
Table 15: Level 1 MCB Hardware Gyroscope Module	20
Table 16: Level 2 MCB Hardware 9 Volt Switching Regulator Module.....	22
Table 17: Level 2 MCB Hardware 5 Volt Switching Regulator Module.....	22
Table 18: Level 2 MCB Hardware 3.3 Volt Linear Regulator Module	22
Table 19: Level 2 MCB Hardware Transceiver Module	22
Table 20: Level 2 MCB Hardware PIC Module	23
Table 21: Level 2 MCB Hardware MPU-6050 Gyroscope Module	23
Table 22: Level 0 MCB Software Module.....	24
Table 23: Level 1 MCB Software Call/Receive Command Module	25
Table 24: Level 1 MCB Software Environment Analyzer Module	25
Table 25: Level 2 MCB Call/Request Module.....	26
Table 26: Level 2 MCB Communications Parsing Module	26
Table 27: Level 2 MCB Software IsLinearSlope? Module.....	27
Table 28: Distance Calculator Module.....	27
Table 29: Level 2 MCB Attack/Evade Module.....	27
Table 30: Level 2 MCB Navigator Module.....	27
Table 31: Main Control Board Bill of Material	42
Table 32: Level 2 Motor Driver Board Hardware Current Monitor Module	181
Table 33: Level 2 Motor Driver Board Hardware H-Bridge Module	181
Table 34: Level 2 Motor Driver Board Hardware PWM Generator Module	182
Table 35: Level 2 Motor Driver Board Hardware Bridge Driver Module	182
Table 36: Level 2 Motor Driver Board Hardware Microcontroller Module	182
Table 37: Level 2 Motor Driver Board Hardware Micro SD Storage Module	182
Table 38: Level 2 Motor Driver Board Hardware Level Shifter Module	183
Table 39: Level 2 Motor Driver Board Hardware Board Monitor Sensor Module.....	183
Table 40: Level 2 Motor Driver Board Hardware Rotary Encoder Decoder Module	183

Table 41: Level 1 Motor Driver Board Software Protection Throttle Module	184
Table 42: Level 1 Motor Driver Board Software Closed Loop Controller Module	184
Table 43: Level 2 Motor Driver Board Software Decode Speed and Direction Data Module	186
Table 44: Level 2 Motor Driver Board Software Read Temperature Data Module	186
Table 45: Level 2 Motor Driver Board Software Map Temperature Data Module.....	187
Table 46: Level 2 Motor Driver Board Software Thermal Throttle Module	187
Table 47: Level 2 Motor Driver Board Software Read Encoder Counter Module	187
Table 48: Level 2 Motor Driver Board Software Map Encoder Counter Module	187
Table 49: Level 2 Motor Driver Board Software PID Controller Module	187
Table 50: Level 2 Motor Driver Board Software Write PWM Duty Ratio Module.....	188
Table 51: Motor Driver Board Bill of Material	204
Table 52: Level 2 Proximity Sensor Board Hardware Linear 5 VDC Regulator Module	222
Table 53: Level 2 Proximity Sensor Board Hardware LIDAR Sensor Module	222
Table 54: Level 2 Proximity Sensor Board Hardware Hall Effect Sensor Module.....	222
Table 55: Level 2 Proximity Sensor Board Hardware Microcontroller Module	222
Table 56: Level 0 Proximity Sensor Board Software Weapon Position Evaluator Module.....	227
Table 57: Level 0 Proximity Sensor Board Software Scan Environment Module	228
Table 58: Level 0 Proximity Sensor Board Software Process Data Module	228
Table 59: Level 0 Proximity Sensor Board Software Data Storage Module.....	228
Table 60: Level 1 Proximity Sensor Board Software Sample Hall Effect Sensor Module.....	229
Table 61: Level 1 Proximity Sensor Board Software Weapon Position Evaluator Module.....	230
Table 62: Level 1 Proximity Sensor Board Software Enable LIDAR Sensor Module	230
Table 63: Level 1 Proximity Sensor Board Software Process Environment Data Module	230
Table 64: Level 1 Proximity Sensor Board Software Data Storage Module	230
Table 65: Level 1 Proximity Sensor Board Software Communications Parsing Module.....	231
Table 66: Level 1 Proximity Sensor Board Software Send Data to MCB Module	231
Table 67: Proximity Sensor Board Bill of Material	238
Table 68: Level 1 Weapon Motor Control Hardware Microcontroller Module	366
Table 69: Level 2 Weapon Motor Control Hardware Microcontroller Module	367
Table 70: Level 2 Weapon Motor Control Hardware RS485 Transceiver Module.....	367
Table 71: Level 2 Weapon Motor Control Hardware Opto-Isolator Module.....	367
Table 72: Level 2 Weapon Motor Control Hardware 18V Regulator Module	367
Table 73: Level 2 Weapon Motor Control Hardware 5V Regulator	368
Table 74: Level 2 Weapon Motor Control Hardware IGBT Driver Module.....	368
Table 75: Level 2 Weapon Motor Control Hardware I2C ADC Module	368
Table 76: Level 1 Weapon Motor Control Software External Interrupt Module.....	369
Table 77: Level 1 Weapon Motor Control Software Communications Parsing	370
Table 78: Level 1 Weapon Motor Control Software Closed Loop Control Calculation.....	370
Table 79: Level 1 Weapon Motor Control Software Output Compare Module	370
Table 80: Level 1 Weapon Motor Control Software I2C Module.....	370
Table 81: Weapon Motor Control Board Bill Of Materials	377
Table 82: MDB Single FET Power Table	458

Table 83: MDB Rds at Load Currents Table.....	460
Table 84: MDB Theoretical Single MOSFET Power Table	461
Table 85: MDB Simulation VS Theoretical Difference Table.....	461
Table 86: IXGN200N60B3-ND Switching Characteristics	464
Table 87: IGBT Voltage Regulator Output Capacitance Simulation Results	472
Table 88: MDB Revised Material Cost.....	487
Table 89: PSB Revised Material Cost.....	488
Table 90: WMCB Revised Material Cost	489
Table 91: Mechanical Revised Material Cost	491

1.0.0 Problem Statement

1.1.0 Need

Competitive combat robots such as the ones that compete in the RoboGames combat event are difficult to operate due to the large moment of inertia that results from spinning weapon systems and the constant change in viewing angle the operator of the robot experiences as the robot navigates around the arena. The difficulty of operation can cause operating mistakes that result in excessive damage to the robot and potentially the loss of the round. There is a need for an autonomous system in a combat robot that can navigate the robot around the arena, identify the other competing robot, and make the decision whether to attack or retreat from the competing robot. (MJH)

1.2.0 Objective

The objective is to build a combat robot that is able to compete in the Robogames combat event autonomously. An autonomous system will be designed to enable the robot to navigate around the RoboGames arena autonomously. However, RoboGames requires the robot to also operate manually. Therefore, the robot must have the ability to operate manually and autonomously. A weapon motor controller to control the spinning weapon attachment and a motor driver controller to control the motors attached to the wheels must be designed. A main control board and a sensor interface board for the autonomous feature must also be designed. (MPH)

1.3.0 Research Survey

1.3.1 Patent Search

Patent US4638445A published on January 1, 1987, describes an approach that is very applicable to the design goals of this project. The patent describes a method of using two arrays of sensors to provide distance and position data to a computer. One of the sensors captures proximity information about an object that is near the robot and that is in-line with the sensor viewing angle. The second sensor captures proximity information about an object that is far from the robot and in-line with the sensor acquisition angle. The captured data is then processed by the computer so that the robot can complete predetermined goals. The patent states “this mobile robot utilizes multiple means for collision avoidance” [4] referring to the sensor data and contact pads positioned around the robot. This arrangement of sensors can provide the robot with a simple form of robotic vision. (MJH)

Patent US 5165064 A, “publication on Nov 17, 1992, by, Paul J. Mattaboni”, describes a method of achieving mobile robot guidance and navigation by employing one or more arrays of ultrasonic transducers and infrared detectors. The transducer purpose is to monitor different sectors of the robot workspace to allow the user to constantly know what is occurring without being there physically to observe. For the infrared sensors, the purpose is to allow the robot to allow the user to determine the location of the robot. There will be an infrared beacon system that has a uniquely coded infrared signal that the sensors will detect, the robot will need a decoder for this to work properly. Both of these components would be extremely beneficial to the robot and the user due to it providing guidance of obstacles (such as arena wall location, random structures in arena) and where the robot is while in the arena. (HL)

Patent US8060251 B2, "publication on Nov 15, 2011 by Hector H. Gonzalez-Banos, Victor Ng-Thow-Hing, and Allen Y. Yang" described an interface for robot motion control. The Patent describes the robot being controlled by 4 possible embodiments. The one that will be focused on is the first embodiment which consists of a controller that is used to control the robots motion through interface. Thus, interface will communicate directly with the driver modules to directly control the robot. This is a very useful method to be applied to the robot due to the purpose of the robot in a combat situation there must be some form of control of movement used for the robot or else it will be stationary during the whole duration of the match. (HL)

1.3.2 IEEE Explore Article Search

The first article, titled "Twin Low-cost Infrared Range Finders for Detecting Obstacles Using in Mobile Platforms," discusses how two IR sensors are used to detect obstacles. The author uses one short range IR sensor and one long range IR sensor to detect the objects while the sensors rotate on the servo motor. The short range detection of the IR sensor ranges from 20 - 150 cm. and the long range detection of the other IR sensor is 100 -500 cm.

The author states, "With the limitation of data transfer from microcontroller via serial communication, the indoor environments have to be static" [1]. According to the statement just mentioned, faster data transfer is necessary if the robot is to detect other moving robots. This article is a good starting point for the motion tracking system that needs to be developed for the combat robot. It really relates to how the robot is going to use IR sensors to track other enemy robots and determine where the walls are in the arena. The concept of using short and long range IR sensors will be highly applicable to the combat robot [1]. (MPH)

The second article, titled " Hybrid Position and Orientation Tracking for a Passive

"Rehabilitation Table-Top Robot," discusses a real time hybrid 2D position and orientation tracking system made for a rehabilitation robot. The author discusses the use of a webcam and two optical mouse sensors to track the movement and orientation of the robot. The webcam aspect of this system is not applicable to the combat robot. Image tracking would not be fast enough. However, the author's concept of calculating the "fused" data is interesting and may be applicable to the combat robot with the use of long and short range IR sensors to establish the position tracking [2]. (MPH)

1.3.3 Other Sources

The paper 'Sensor-Based Collision Avoidance Solutions for Drone Fleets' by, Jon Gabay, explores "solutions for localized sensing and detecting of foreign objects in proximity" [4]. Although the paper is intended for autonomous functionality of drones the methods are applicable for a collision avoidance system in the proposed project. The paper explains the functionality of several types on proximity sensors including acoustic, GPS and radar. The paper explains that small, high-frequency RF sensors are currently being implemented in "collision-avoidance radar and lift-gate proximity detection" [4]. (MJH)

Another article titled 'Fast Grid-based Position Tracking for Mobile Robots' by Wolfram Burgard, Dieter Fox, and Daniel Hennig discusses robotic sensor integration and estimation of a robots position based on multiple sensor systems. A discussion on a grid-based solution is presented and experimental results are examined. Sensors are known to be noisy in their response, noted by the writers that uncertain information arises from "imperfect sensors, measuring errors, simplifications, open or closed doors, or even moving objects such as humans or other mobile robots." [5] and a method for handling this error is presented. (ZDK)

A third source titled ‘Autonomous Target Tracking Robot’ is a report based on a design project at Michigan State University, describing the use of ultrasonic sensors to track a target. While the team decided to use an optical camera to discover the object, obstacle avoidance and distance data was obtained using ultrasonic sensors. A sensor array and choice arrangement of the sensors allowed for multiple sensors to describe the state of the environment and make intelligent decisions about how to proceed. The approach described was to avoid objects but provides an example of how sensor feedback can drive microcontroller decision making. (ZDK)

1.4.0 Marketing Requirements

The proposed design must meet the following marketing requirements. (MPH)

1. The robot must be autonomous.
2. The robot must have a weapon system.
3. The robot must be self-powered.
4. The robot must have a position tracking system.
5. The robot must be able to operate upside down if it is flipped over.
6. The robot must have a safety switch to disable the robot if malfunctions occur.
7. The robot must have a manual operation mode.
8. The robot must be durable enough to withstand damage from enemy robots.
9. The robot must be capable of inflicting damage to enemy robots.

1.5.0 Objective Tree

The diagram shown in Figure 1 illustrates the marketing requirements given in Section 1.4. (HL)

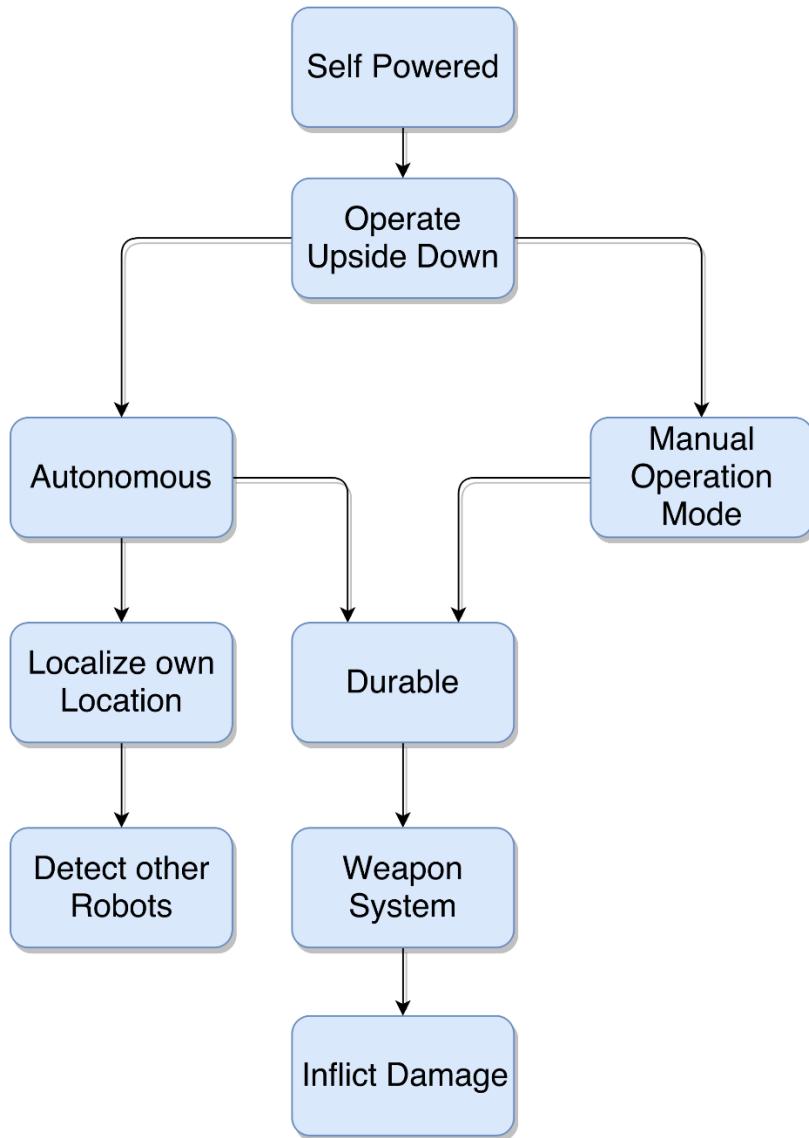


Figure 1: Objective Tree

2.0.0 Design Requirements Specification

The goal of this project is to construct a competitive robot for the combat robot event held by RoboGames. Therefore, the design requirements must meet all the rules of the competition. These rules include, but are not limited to, the following. (MJH)

- The robot must be able to be fully deactivated in under 60 seconds by a manual disconnect
- The weapon system must be able to be mechanically locked to prevent rotation when the robot is off
- The robot must weigh less than 220 pounds
- All motion of the robot must stop when the transmitting controller is turned off
- The robot must use a radio system with digitally coded, mated pairs between transmitter and receiver
- The robot must have a clearly visible light indicating all functions of the autonomous system
- The autonomous function must be able to be manually enabled and disabled
- When the transmitting controller is turned off all autonomous functions must be disabled
- All autonomous functions must disable 60 seconds after disarming
- The spinning weapon must come to a complete stop within 60 seconds of power being removed
- The robot must be able to function for the duration of the 3-minute match

In addition to the rules the robot must abide by the additional design requirements specific to the design of the robot are as follows. The arena the robot will be competing in is

a 40 foot square. With a top speed of at least 15 mph the robot will be able to navigate the length of the arena in approximately 2 seconds. Being able to identify the competing robot with a high level of certainty will result in few instances of misidentification of the competing robot. Misidentifying the competing robot could result in unnecessary damage to the system such as if the robot were to attack the arena wall. The competing robot can weigh up to 220 pounds thus movement should still be possible if the competing robot were to be on top of the system. For the autonomous system to be effective the competing robot must be identified quickly as to not provide sufficient time for their weapon system to be fully engaged. The design requirements below should be sufficient to meet the previously stated design considerations. (MJH)

- Top speed of at least 15mph
- Able to identify opponent robot with certainty above 90%
- Weapon rotation of at least 600 rpm
- Able to move at least 5mph with a 220lb load
- Able to identify a competing robot in under 3 seconds

3.0.0 Accepted Technical Design

The autonomous robot system uses LIDAR sensors to acquire proximity data about the robot's surrounding. The acquired data is processed to distinguish the walls of the area the robot will be competing in and the competing robot. The robot consists of two wheels that are used to propel the robot and a spinning drum that rotates around the exterior of the robot chassis to inflict damage to the competing robot. The rotating drum has two windows that allow the proximity sensors to see the environment. This means that the windows are also rotating and there is a limited time frame proportional to the speed of rotation of the drum that the proximity sensors must acquire their data. The position of the rotating drum is determined by hall effect sensors that trigger a home flag indication. (MJH)

3.1.0 Low Level Overview

Figure 2 illustrates the high-level primary functions of the system. The block diagram is derived from the objective tree and marketing requirements in the previous sections. The operations in the block diagram illustrates the sequence of operations that will occur when the robot is operating in autonomous mode or manual mode. The operator of the robot will send control data to identify whether to operate in autonomous mode or to maintain manual control of the robot. When operating in manual mode the control data from the remote receiver is passed to the motor controllers identifying the speed and direction the motors should rotate. When operating in autonomous mode the robot operator sends a command whether to operate in fight or flight mode. The control information to the motor controllers is calculated by the main control board. The main control board receives proximity information from the proximity sensor boards. The proximity sensor boards use a homing sensor to identify when the proximity sensor is able to capture data through the rotating window that the proximity sensor captures data through. Because the weapon system that rotates around the robot has a window in which the environment

data can be captured there is a finite time that the proximity sensor must operate. Once the data from the proximity sensors is captured by the proximity sensor board the main control board requests the captured data. From the proximity sensor data, the walls of the area and the competing robot can be identified. Once the walls of the arena and the competing robot are identified the system can send control data to the motor controllers that will attack the competing robot or flee from the competing robot. (MJH)

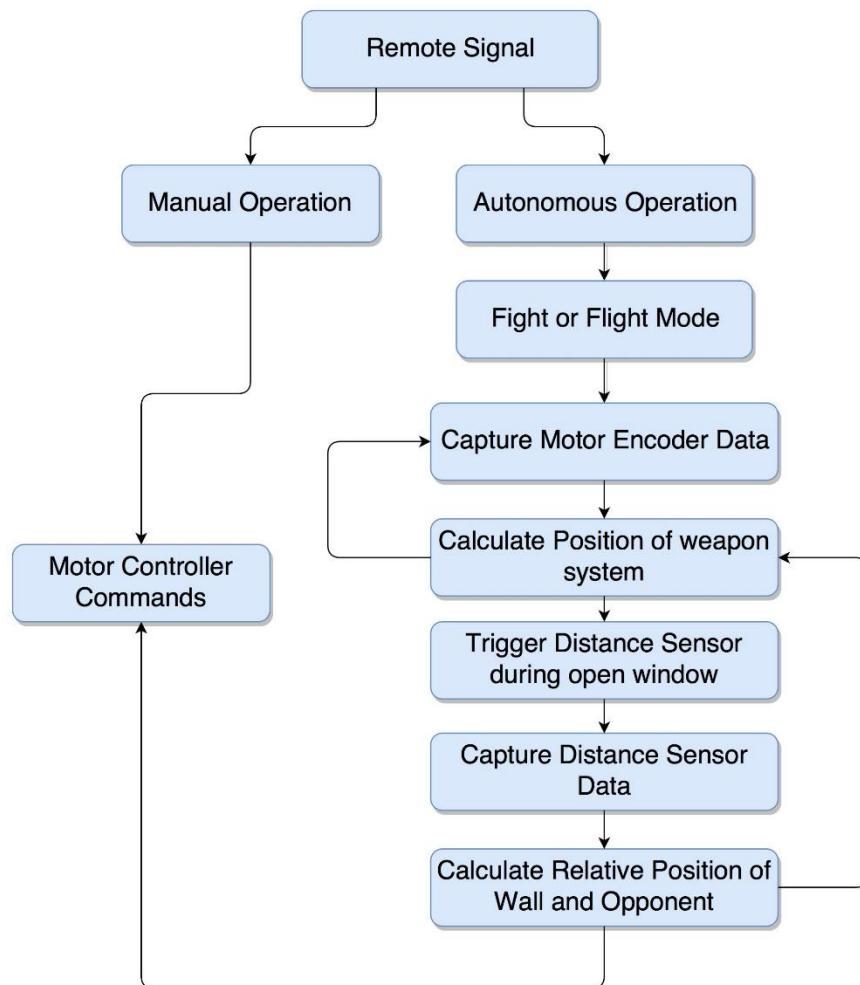


Figure 2: Level 0 Functional Block Diagram

Figure 3 shows a high-level physical representation of the major components of the intended system. Control data is received by the RC receiver that identifies the mode of operation of the robot. In manual control mode data is transmitted to both locomotion motor controllers and the weapon motor controller. In the autonomous mode of operation autonomous operating decisions are calculated by the main control board from sensor information from the proximity sensor boards. Proximity sensor boards are placed on all 8 sides of the robot each with accompanying proximity sensors. The proximity sensor board stores the captured data from the proximity sensors until the main control board requests the data. After data from the proximity sensor boards is captured the proximity data is processed by the main control board to distinguish the competing robot from the arena walls. When the competing robot and walls of the arena are identified the main control board communicates with the two motor driver boards that control the locomotion motors of the robot. Encoder feedback to the locomotion motors is used to control the speed that the locomotion motors are rotating. The main control board also communicates with the weapon motor controller to control when the weapon motor is on and how fast to rotate the weapon system. To prevent damage to the system current to the weapon motor is monitored. By monitoring the current to the weapon motor the average power delivered to the weapon motor can be limited by modulating the weapon motors on time. (MJH)

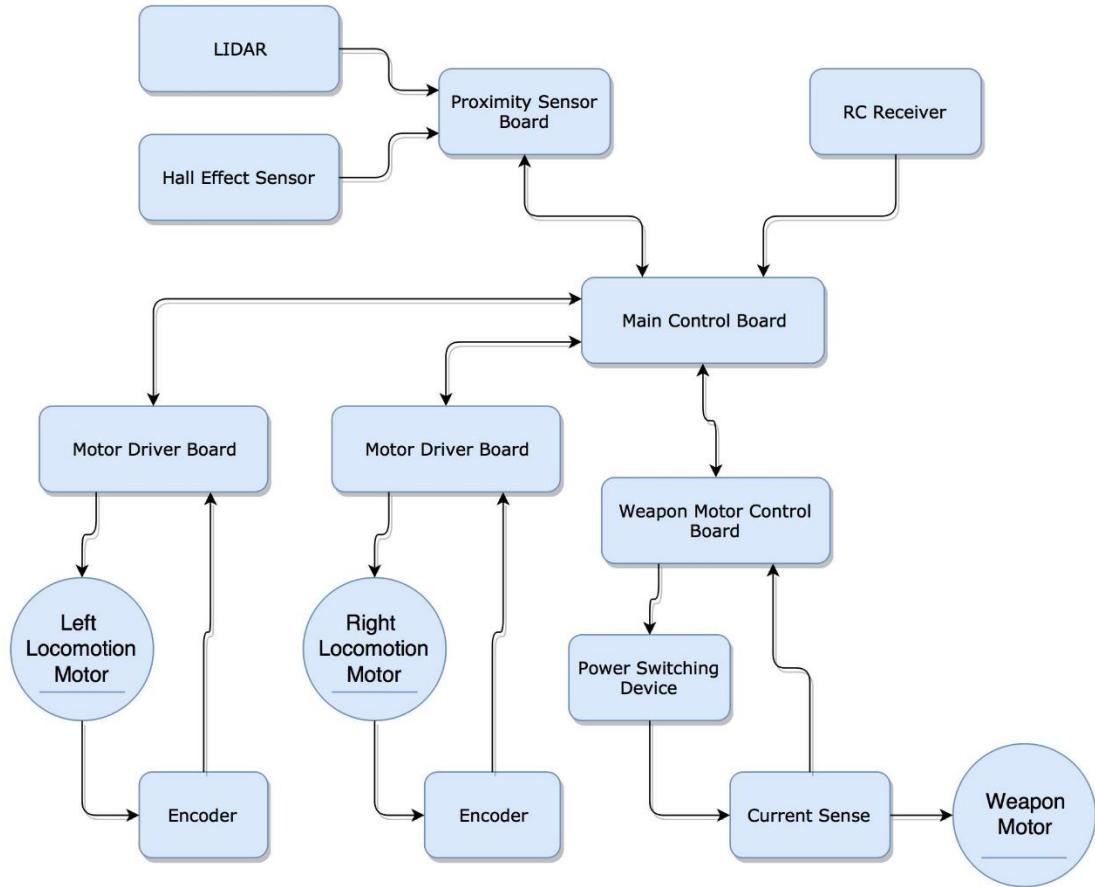


Figure 3: Level 0 Hardware Block Diagram

Module	Proximity Sensor Board
Designer	Mack J. Hawkins
Inputs	Power: 9v DC Signal: I2C Proximity Sensors Data Signal: Digital 5v Home flag indication
Outputs	Power: Regulated 5V for Proximity Sensor and Hole Flag Sensor Signal: RS485 Proximity Data
Description	This board will interface between the main control board and the proximity sensors. The sensor data will be processed and sent to the main microcontroller over a digital bus.

Table 1: Level 1 Hardware Proximity Sensor Board Module

Module	LIDAR
Designer	Off the Shelf Solution (MJH)
Inputs	Power: Regulated 5 Vdc
Outputs	Signal: I2C Proximity Sensors Data
Description	This sensor captures proximity data from the environment that will be used to distinguish the arena walls from the competing robot.

Table 2: Level 1 Hardware LIDAR Module

Module	Hall Effect Sensor
Designer	Off the Shelf Solution (MJH)
Inputs	Power: Regulated 5 Vdc
Outputs	Signal: Digital 5v Home flag indication
Description	This sensor is used to detect the position of the weapon system relative to the chassis of the robot. This data will be used to determine when to capture data from the LIDAR sensor.

Table 3: Level 1 Hardware Hall Effect Sensor Module

Module	Remote Receiver
Designer	Off the Shelf Solution (MJH)
Inputs	Power: Regulated 5 Vdc Signal: 2.4 GHz Wireless RC Signal
Outputs	Signal: 6 Channel PWM (0-5 Vdc)
Description	The remote receiver receives control information from the handheld controller and decodes the signal into 6 channels of PWM signal.

Table 4: Level 1 Hardware Remote Receiver Module

Module	Motor Driver Board
Designer	Mack J. Hawkins
Inputs	Power: 24v DC Battery Signal: RS485 Speed and Direction Signal: Quadrature Encoder Data
Outputs	Signal: High power PWM signal to drive motors
Description	This board will process the data from the main control board and drive the speed of the motors accordingly. Speed and direction will be able to be controlled.

Table 5: Level 1 Hardware Motor Driver Board Module

Module	Main Control Board
Designer	Mack J. Hawkins
Inputs	Power: 24v DC Battery Signal: PWM from RC Receiver (0-5 Vdc) Signal: RS485 Proximity Data
Outputs	Signal: RS485 Speed and Direction data to Motor Driver Boards Signal: RS485 Speed data to IGBT Driver Board Power: 9V DC to Power Proximity Sensor Boards and Weapon Control Board
Description	This board will determine whether the robot is in autonomous or manual mode of operation. If in manual mode the control signals from the RC receiver will be bypassed to the motor controllers. If in autonomous mode this board will process the data from the Proximity Sensor Board and determine where the wall and competitor are located. From this calculation speed and direction information will be sent to the motor drivers. Various temperature sensors located around the robot will also be monitored to know the status of all the subsystems.

Table 6: Level 1 Hardware Main Control Board Module

Module	Encoder
Designer	Off the Shelf Solution (MJH)
Inputs	Power: Regulated 5 Vdc
Outputs	Signal: Quadrature (0-5 Vdc)
Description	The encoders will provide rotational information to the microcontrollers about the speed and direction of rotation.

Table 7: Level 1 Hardware Encoder Module

Module	Weapon Motor Control Board
Designer	Mack J. Hawkins
Inputs	Power: 9V DC Signal: RS485 Speed data Signal: 0-5 Vdc from weapon motor current sensor
Outputs	Signal: High power PWM signal to drive IGBT gate
Description	The weapon motor control board drives the gate of the IGBT that switches power to the weapon motor. The received RS485 data sets the speed at which the weapon motor is to rotate.

Table 8: Level 1 Hardware Weapon Motor Control Board Module

Module	Power Switching Device
Designer	Mack J. Hawkins
Inputs	Power: 60v DC Battery Signal: High power PWM
Outputs	Power: High Power PWM
Description	The power switching device switches power to the weapon motor. By adjusting the duty ratio the switching device is on the speed of rotation of the weapon motor can be adjusted.

Table 9: Level 1 Hardware Power Switching Device Module

Module	Current Sense
Designer	Mack J. Hawkins
Inputs	Power: Regulated 5 Vdc
Outputs	Signal: 0-5 Vdc Proportional to the current to the weapon motor
Description	The current sense is used to monitor the current to the weapon motor. This allows protection to the battery pack by monitoring for over current draw scenarios.

Table 10: Level 1 Hardware Current Sense Module

Module	Left & Right Locomotion Motor
Designer	Mack J. Hawkins
Inputs	Signal: High power PWM
Outputs	Power: Rotation of the Motor
Description	The locomotion motors are used to propel the robot around the arena.

Table 11: Level 1 Hardware Left & Right Locomotion Motor Module

Module	Weapon Motor
Designer	Mack J. Hawkins
Inputs	Signal: High power PWM
Outputs	Power: Rotation of the Motor
Description	The weapon motor is used to rotate the weapon system around the chassis of the robot.

Table 12: Level 1 Hardware Weapon Motor Module

3.1.1 FastTransfer Communications Protocol

A communications protocol called FastTransfer is used to provide a call and response structured bus topology allowing for a multi-drop bus providing communications handling between nodes on the network. Each node will be assigned a unique address, responding to packets when a message on the bus is addressed to their address. There is one master node, in the case of the robot this is the Main Control Board (MCB). The MCB will send messages either requesting data or writing data to the slave devices on the network. Writing or reading packets are identical in form, consisting of a 5 byte header, $2*N+1$ bytes of data and a single byte CRC providing insurance for data integrity. The data sent is allocated as an integer, or two bytes of data, that are concatenated into a single 16-bit data value. Each integer of data sent has an

address byte associated with it, which allows for storing the data address in the correct location in the receive array. (ZDK)

Figure 4 shows the packet structure of a FastTransfer packet consisting of the header, a single integer of data and a CRC. (ZDK)

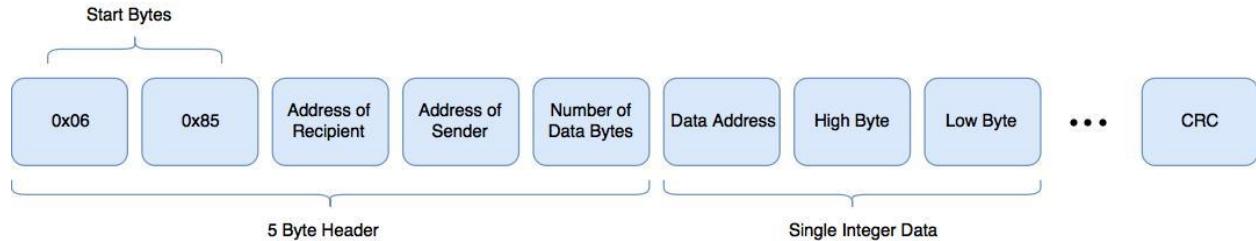


Figure 4: FastTransfer Packet Structure

When sending only a single piece of data, the total message length is 9 bytes long. Each additional piece of data sent within the packet will extend the packet length by 3 additional bytes, as the data requires the data address, and two bytes of data to make a 16-bit piece of data. (ZDK)

The device that is utilizing the FastTransfer protocol listens to the bus, awaiting the arrival of the start bytes to align the parser with the beginning of the packet. Once the start bytes are recognized, the device checks the next byte that is sent and compares this byte to its own address. If the address is mismatched, the device simply ignores the rest of the packet and waits for the next start bytes. (ZDK)

Upon acquiring a matching address to be received by the node, the packet data contents are then sorted by data address and stored into a receive array in the position indicated by the data address tied to the incoming data. FastTransfer permits a variable packet size, meaning that a variable number of data values can be sent in each message. This provides flexibility in the message contents, allowing for either multiple pieces of data to be sent in a single message, or a

single piece of data, reducing overhead for header bytes, increasing theoretical throughput versus a single message containing a single piece of data. (ZDK)

The complete FastTransfer library for Arduino IDE is included in the appendix for reference.

3.2.0 Main Control Board

The Main Control Board (MCB) allows the robot to communicate to all of the other boards and regulates the voltage from the battery to power all of the other boards excluding the Motor Driver Board. The MCB will communicate with the Proximity Sensor Boards and process the scanned raw Lidar data to identify the surrounding environment. Then gives motor movement instructions to the Motor Driver Board and weapon instructions to the Weapon Motor Control Board. (HL)

3.2.1 Main Control Board Hardware

Figure 5 describes how the Main Control Board (MCB) will interact with the other boards in the Robot. The MCB will request data from the sensor boards and then the acquired data will then be calculated and distributed to the Motor Driver Board and Weapon Motor Control Board. There will be a 24 Volt unregulated source from the battery which will flow into 9,5 and 3.3 volt regulators. The 5-volt regulator will supply power to the MCB, the 9 Volts will supply power to the Sensor Board and Weapon Motor Control Board and the 3.3 volt will be sent to the IMU. The purpose of the IMU are to indicate the angles to allow calculations for identifying its location in the arena, allow the robot to turn specific angles with feedback, and the accelerometer indicates if the robot is upside down or not. The MCB will also allow the robot to alternate between autonomous mode and manual control by a RC Control Signal. (HL)

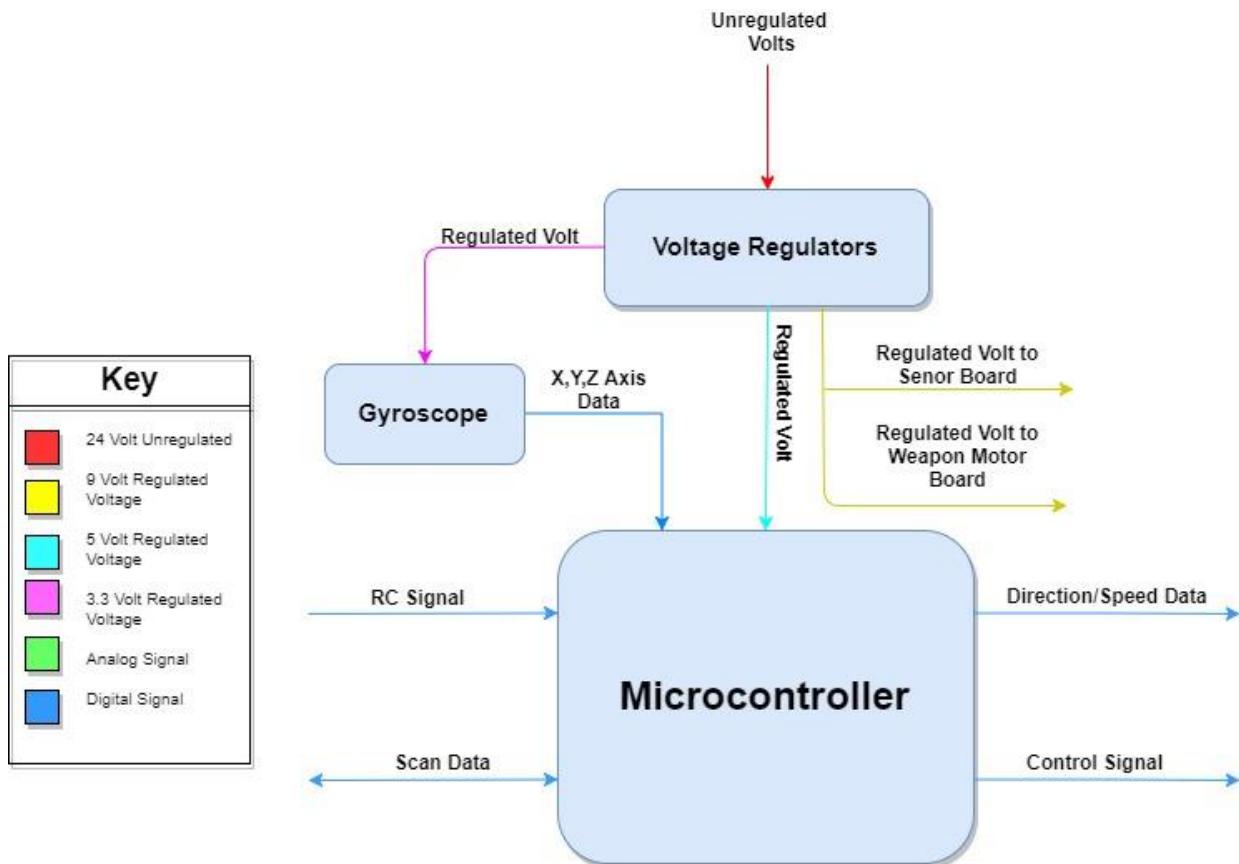


Figure 5: Level 1 MCB Hardware Block Diagram

Module	Voltage Regulators
Designer	Hong (Joey) Lau
Inputs	Power: Unregulated 24 Volt
Outputs	Power: Regulated 9 Volt to Sensor Board Power: Regulated 9 Volt to Weapon Motor Control Board Power: Regulated 5 Volt to Microcontroller Power: Regulated 3.3 Volt to Gyroscope
Description	The Lithium battery will supply a unregulated 24 Volt to the Voltage Regulators. Each voltage regulator will drop the voltage to supply the required power for each board (except Motor Driver Board) or hardware. 9 Volt will be supplied to both the Sensor Board and Weapon Motor Control Board where 5 Volt will be sent to the Microcontroller for the Main Control Board and 3.3 Volt to the Gyroscope.

Table 13: Level 1 MCB Hardware Voltage Regulator Module

Module	Microcontroller
Designer	Hong (Joey) Lau
Inputs	Signal: RC Signal Signal: Scan Data Signal: X,Y,Z Axis Data Power: Regulated 5 Volt to Microcontroller
Outputs	Signal: Direction/Speed Data Signal: Control Signal Signal: Scan Data
Description	The Microcontroller will be the main hardware for the MCB. It will interact with the other boards and gyroscope of the robot. The Scan Data I/O is bidirectional due to it requesting and receiving data directly from the Sensor Boards. The Control signal will be sent to the WMCB and the Direction/Speed Data will be sent to the MDB.

Table 14: Level 1 MCB Hardware Voltage Regulator Module

Module	Gyroscope
Designer	Hong (Joey) Lau
Inputs	Power: Regulated 3.3 Volt to Gyroscope
Outputs	Signal: X,Y,Z Axis Data
Description	The Gyroscope is powered by the regulated 3.3 Volt from the voltage regulator. The Gyroscope will send a X,Y,Z Axis Data to inform the robot which direction to move when all the information is being processed in the microcontroller.

Table 15: Level 1 MCB Hardware Gyroscope Module

In Figure 6 shows a more in depth explanation of how the MCB hardware all interact with each other. The voltage source from the battery will send an unregulated 24 volts to a 5 and 9 volt switching regulator. The switching regulators will step down the voltage so it may send power to Sensor Board, Weapon Motor Control Board and Main Control Board. The Sensor Board and Weapon Motor Control Board both will receive 9 Volt where the 5 Volt will be sent to the PIC and also to a linear 3.3 Volt regulator for the IMU-Gyroscope. The IMU-Gyroscope will tell the robot the direction it will need to go (left, right, forward and back). The PIC microcontroller will perform all the calculations and tell when the transceiver when to talk to the other boards such as retrieve data to be processed or tell how the motor should move. (HL)

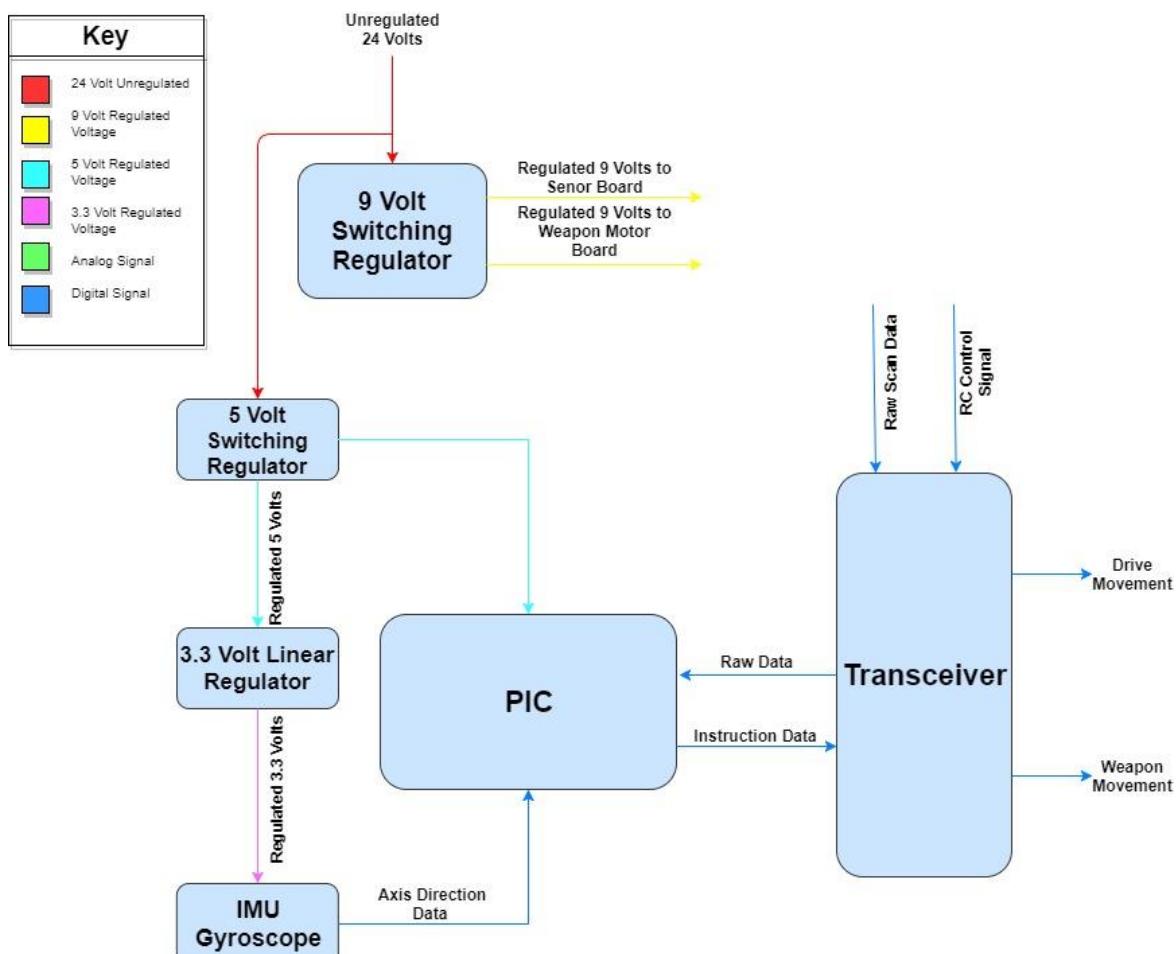


Figure 6: Level 2 MCB Hardware Block Diaagram

Module	9 Volt Switching Regulator
Designer	Hong (Joey) Lau
Inputs	Power: Unregulated 24 Volt
Outputs	Power: Regulated 9 Volt to Sensor Board Power: Regulated 9 Volt to Weapon Motor Control Board
Description	The battery source supplied an Unregulated 24 voltage. The Buck (Step down) Switch Voltage regulator steps down the 24 Volts to 9 Volts to supply power to the Proximity Sensor Board and Weapon Motor Control Board.

Table 16: Level 2 MCB Hardware 9 Volt Switching Regulator Module

Module	5 Volt Switching Regulator
Designer	Hong (Joey) Lau
Inputs	Power: Unregulated 24 Volts
Outputs	Power: Regulated 5 Volts
Description	The 5 Volt switching regulator will regulate the 24 unregulated Volt to 5 Volt for the 3.3V linear voltage regulator and to power the PIC microcontroller.

Table 17: Level 2 MCB Hardware 5 Volt Switching Regulator Module

Module	3.3 Volt Linear Regulator
Designer	Hong (Joey) Lau
Inputs	Power: Regulated 5 Volts
Outputs	Power: Regulated 3.3 Volts
Description	The 3.3 Volt regulator in the board will regulate the voltage for the Gyroscope.

Table 18: Level 2 MCB Hardware 3.3 Volt Linear Regulator Module

Module	Transceiver
Designer	Hong (Joey) Lau
Inputs	Signal: Instruction Data Signal: Raw Scan Data Signal: RC Control Signal
Outputs	Signal: Motor Drive Movement Signal: Weapon Movement Signal: Instruct when PSB to send Data
Description	The output transceiver communicates with the other boards by RS-485 to request and send data.

Table 19: Level 2 MCB Hardware Transceiver Module

Module	PIC
Designer	Hong (Joey) Lau
Inputs	Power: Regulated 5 Volt Signal: Axis Direction Data Signal: Raw Data
Outputs	Signal: Instruction Data
Description	The PIC will communicate with all the other boards and process the data received to send instructions to the other boards. It will be powered from the 5 volt regulator.

Table 20: Level 2 MCB Hardware PIC Module

Module	MPU-6050 Gyroscope
Designer	Hong (Joey) Lau
Inputs	Power: Regulated 3.3 Volts
Outputs	Signal: Axis Direction Data
Description	The Gyroscope will be powered by the 3.3 volt regulator. Once the data has been processed and identified the robot will be able to identify the directions for the robot.

Table 21: Level 2 MCB Hardware MPU-6050 Gyroscope Module

3.2.2 Main Control Board Software

Figure 7 shows a high-level representation of how the software will function. The input will be scanned data from the sensors. Since the data is just array of unorganized data the software will distinguish the data telling the robot which is the enemy and what is the wall. Thus, giving instructions for movement and when to activate the weapon. (HL)

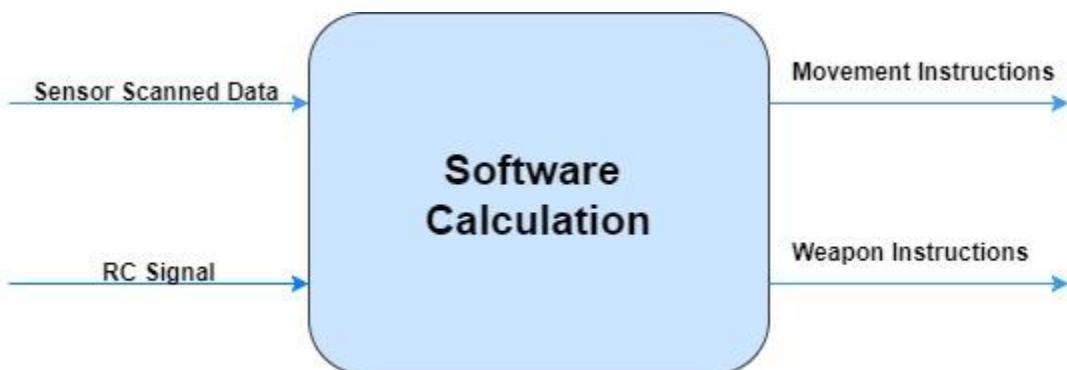


Figure 7: Level 0 MCB Software Block Diagram

Module	Software Calculation
Designer	Hong (Joey) Lau
Inputs	Signal: Sensor Scanned data Signal: RC Signal
Outputs	Signal: Movement Instructions Signal: Weapon Instructions
Description	The combat robot software will process all sensor data to give instructions of operation to the Motor Driver Board and Weapon Motor Control Board. The RC signal will allow the robot to know if it is in autonomous mode or manual control mode.

Table 22: Level 0 MCB Software Module

Figure 8 shows a level 1 block diagram of how the software will operate. There will be a Call/Receive that will request and receive signals that will later be sent to the Environment Analyzer to process the signals for instructions for the robot. The Environment Analyzer will process the raw data and send control signals to the Motor Driver Board and Weapon Motor Control Board. (HL)

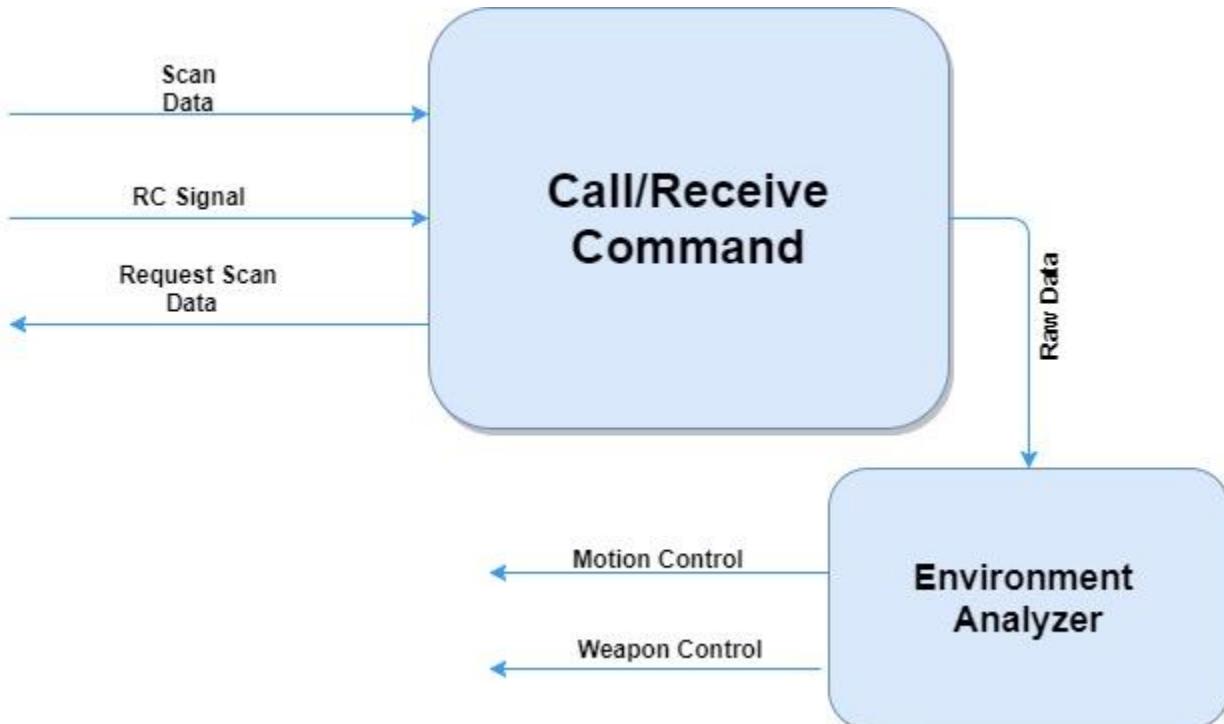


Figure 8: Level 1 MCB Software Block Diagram Motor

Module	Call/Receive Command
Designer	Hong (Joey) Lau
Inputs	Signal: Scan Data Signal: RC Signal
Outputs	Signal: Request Scan Data Signal: Raw Data
Description	The Call/Receive Command request the scanned data from Sensor Board. Once the information has been received, the Raw data will then be sent to the Environment Analyzer Algorithm to be calculated.

Table 23: Level 1 MCB Software Call/Receive Command Module

Module	Environment Analyzer
Designer	Hong (Joey) Lau
Inputs	Signal: Raw Data
Outputs	Signal: Motion Control Signal: Weapon Control
Description	The Environment Analyzer will organize the data and differentiate the wall, location of the robot and the enemy robot in the arena. Once the analyzer has processed all the information it will send instruction to tell where the robot will need to move and when to turn on the weapon.

Table 24: Level 1 MCB Software Environment Analyzer Module

Figure 9 shows the process how the robot will distinguish useful and non-useful information collected from IR Sensors. The software will request the scanned data from the sensor boards and then us an algorithm to identify if the scanned area has a linear region. If the scanned area has a linear region it will declare that area as wall and if not it will declare it as an object. If the scanned area has a linear region the software will use a distance calculator algorithm correlating the with geometry of the square arena to identify where in the area the robot is located. While data is being processed the Weapon Motor Control Board will send the weapon speed data to the microcontroller to see if the weapon is up to speed to attack or evade. The software will do an if/else function where if the weapon speed is past or at a specific value the robot will go in Fight mode, otherwise it will stay in Flee mode until the weapon speed is at the desired value. Once information has been processed and analyzed, it will be sent to the

Motor Driver Board and Weapon Motor Control Board to control the weapon and movement.

(HL)

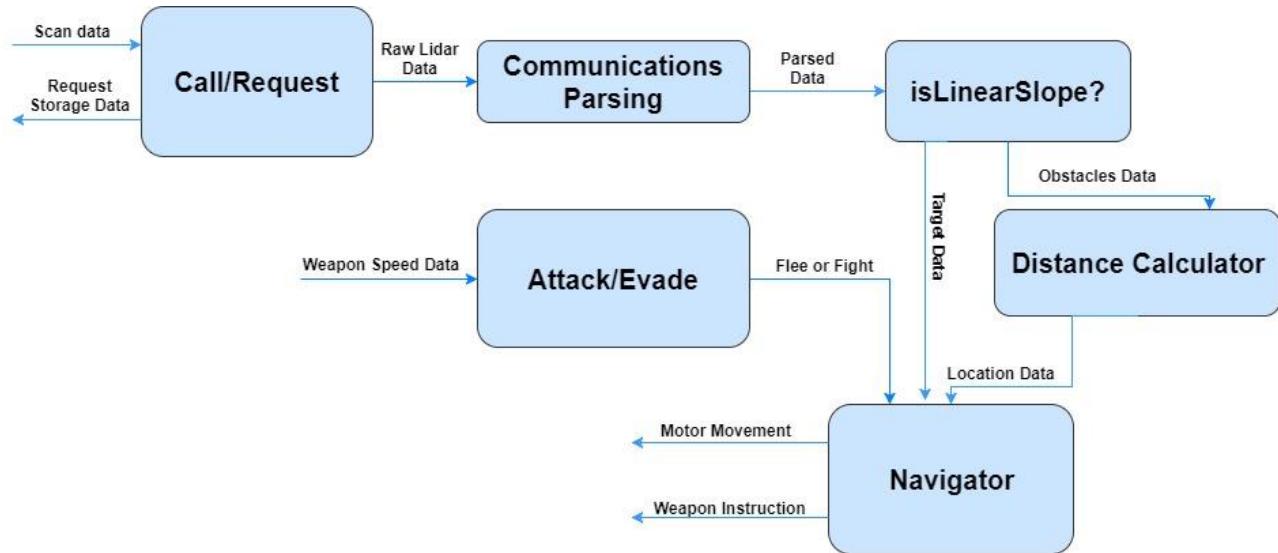


Figure 9: Level 2 MCB Software Block Diagram

Module	Call/Request
Designer	Hong Lau
Inputs	Signal: Scan Data
Outputs	Signal: Request Storage Data
Description	The Call/Request command will communicate with the Sensor Board and request and retrieve the scan data from the Lidar Sensor located in the Storage bank.

Table 25: Level 2 MCB Call/Request Module

Module	Communications Parsing
Designer	Hong Lau
Inputs	Signal: Raw Lidar Data
Outputs	Signal: Parsed Data
Description	The communication parsing process will analyze the incoming data then split it into pieces of data that can be easily filtered into useful and un-useful information.

Table 26: Level 2 MCB Communications Parsing Module

Module	IsLinearSlope?
Designer	Hong Lau
Inputs	Signal: Raw Lidar Data
Outputs	Signal: Target Data Signal: Obstacles Data
Description	The Raw Data retrieved from the RS-485 will then go through a function named "isLinearSlope?" This function will read the Raw array data from the Sensor boards and differentiate the wall from the robot. It will use an algorithm which will read if the scanned data has linear slope which relates to the square arena walls (Obstacles Data). If the Slope is not linear it will register that as an object (Target Data).

Table 27: Level 2 MCB Software IsLinearSlope? Module

Module	Distance Calculator
Designer	Hong Lau
Inputs	Signal: Obstacles Data
Outputs	Signal: Location Data
Description	The Distance Calculator will read the Obstacles Data (Scanned Wall data) from the "IsLinearSlope?" function and read how far the obstacle is from the robot. Since the arena is a uniform square the algorithm will read the distance of the scan data and apply geometry to that information to calculate exactly where the Robot is in the arena.

Table 28: Distance Calculator Module

Module	Attack/Evade
Designer	Hong Lau
Inputs	Signal: Weapon Speed Data
Outputs	Signal: Flee or Fight
Description	The Weapon Motor Control Board will send the weapon speed data where a function will decide if the robot shall flee or fight. If the speed of the weapon is not equal or higher than a certain speed the robot will go into flee mode otherwise it will be in fight mode.

Table 29: Level 2 MCB Attack/Evade Module

Module	Navigator
Designer	Hong Lau
Inputs	Signal: Target Data Signal: Location Data
Outputs	Signal: Motor Movement Signal: Weapon Instruction
Description	Once the Software has differentiated the Wall from the enemy robot. The Target Data (Enemy Robot) and Location Data (walls and position in arena) will then be processed and instructions for Movement and Weapon activation will be executed.

Table 30: Level 2 MCB Navigator Module

3.2.3 Main Control Board Schematic

In Comm. Ports Schematic sheet shown in Figure 10 shows the communication connection Molex Clik-Mate ports that allow communication between the Main Control Board and the other boards in the robot.

In Communication Schematic sheet shown in Figure 11 shows how the Main Control Board will communicate with the proximity Sensor Board, Motor Driver Board and Weapon Motor Control Board. The MCB will use a RS-485 transceiver to communicate to the other boards by protocol signals. The two Voltage Translators (U7 & U8) are programmable unidirectional and the pin connect to one of the PIC I/O controls the direction. One voltage translator will transmit one-step up the voltage form 3.3V to 5V and the receiver translator will step down the voltage from 5V to 3.3V. For partial-power-down, the circuitry disables the outputs, which will prevent damaging current backflow through the device when it is powered down.

In IMU Schematic sheet shown in Figure 12 is the Inertial Measurement Units (IMU). The IMU has 3D digital accelerometer and a 3D digital gyroscope, which allows the user to motion track the robot movement. The IMU will additionally allow the robot to identify its position also direction it is moving when it is maneuvering around the arena.

In Indication & Settings Schematic sheet shown in Figure 13 shows two I2C I/O expanders. One expander (U9) consist a DIP Switch and the other expander (U4) consists a 7-segments display. The device's outputs (latched) have high-current drive capability for directly driving LEDs but has low current consumption.

In Inputs Schematic sheet shown in Figure 14 consists the Power Input, External 9V Supply support as a backup if the 9 voltage regulators has issues, two channels for programming, a PIC USB Input for communication and a RC Receiver Input to notify the robot when it is in autonomous mode. The Power Input, External 9V Supply Support and the two channel for programming uses an 8 Position Receptacle Connector surface mount Molex connector. The RC Receiver Input uses a 20 Position Receptacle Connector surface mount Molex connector.

In Microcontroller Schematic sheet shown in Figure 15 shows the PIC32MZ2048EFG064 microcontroller that is used on the Main Control board that will do all the calculations for the robot and decision making for the robot actions. The microcontroller runs at 200MHz which will allow the robot to perform calculations quickly for decision making. The VDD Decoupling Capacitors suppresses high-frequency noise in power supply signals to prevent any interference to the ICs. The reset button is for a quick reset if anything occurs that would cause the robot to prevent it to act accordingly and I2C pull up is used to ensure a known state for a signal.

In Micro-SD Schematic sheet shown in Figure 16 the Micro-SD is powered by a 3.3V regulator. The schematic consists of the housing which the micro-SD is inserted into, a 3.3v linear regulator, and a level shifter. The micro-SD allows the users to retrieve any data the robot has processed to be analyzed later.

In Power Regulation Schematic sheet shown in Figure 17 there are three Voltage regulators shown, 9V switch regulator, 5V switch regulator and a 3.3V linear regulator. The 9V switch regulator will supply power to the Proximity Sensor boards and Weapon Motor Control Board. The 5V will supply power the Main Control board and the 3.3V will supply power to the

IMU. The power LED's will notify when voltage are going through the lines or if the robot is on or off.(HL)

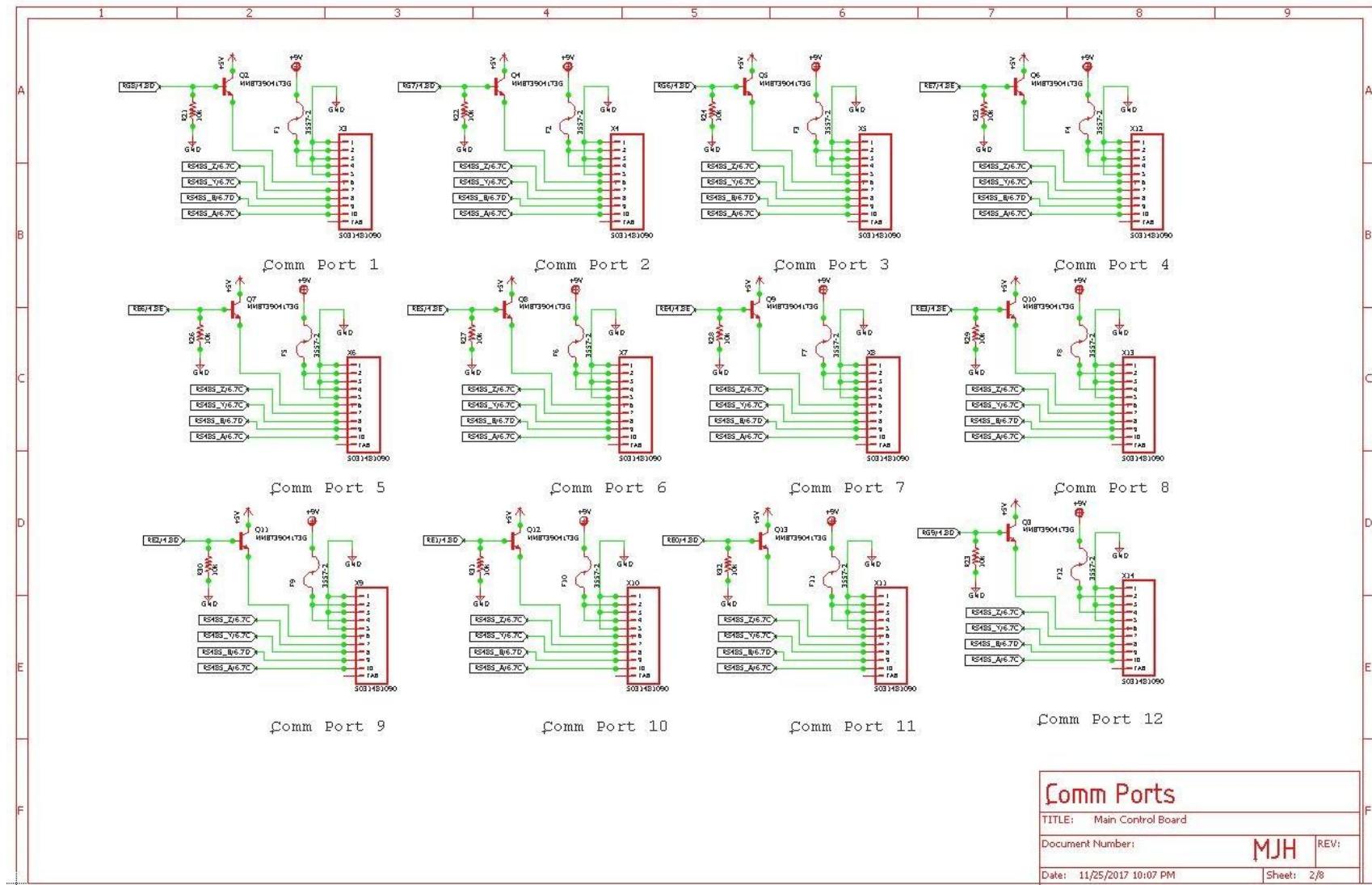


Figure 10: Main Control Board Comm. Ports Schematic

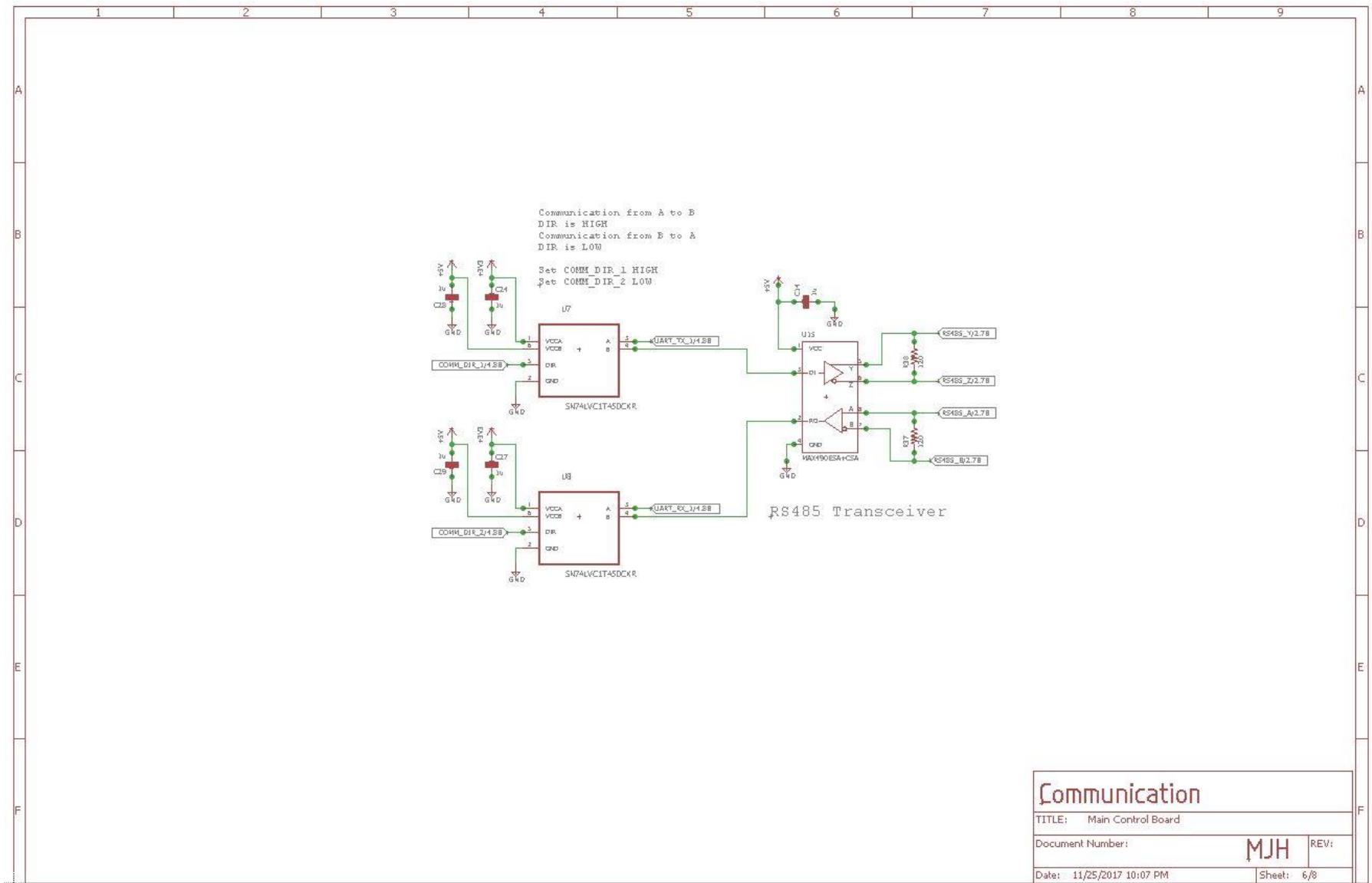


Figure 11: Main Control Board Communication Schematic

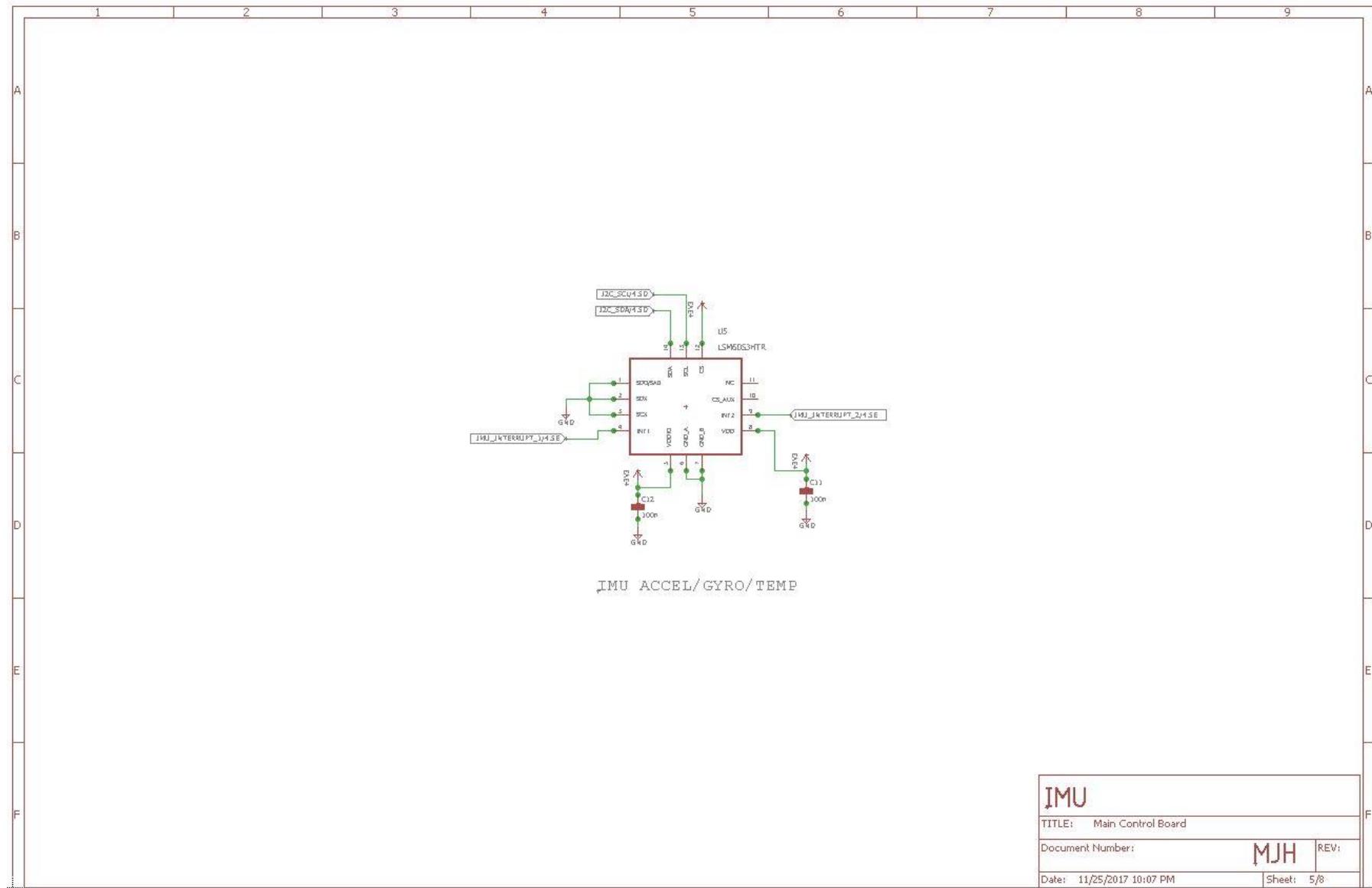


Figure 12: Main Control Board IMU Schematic

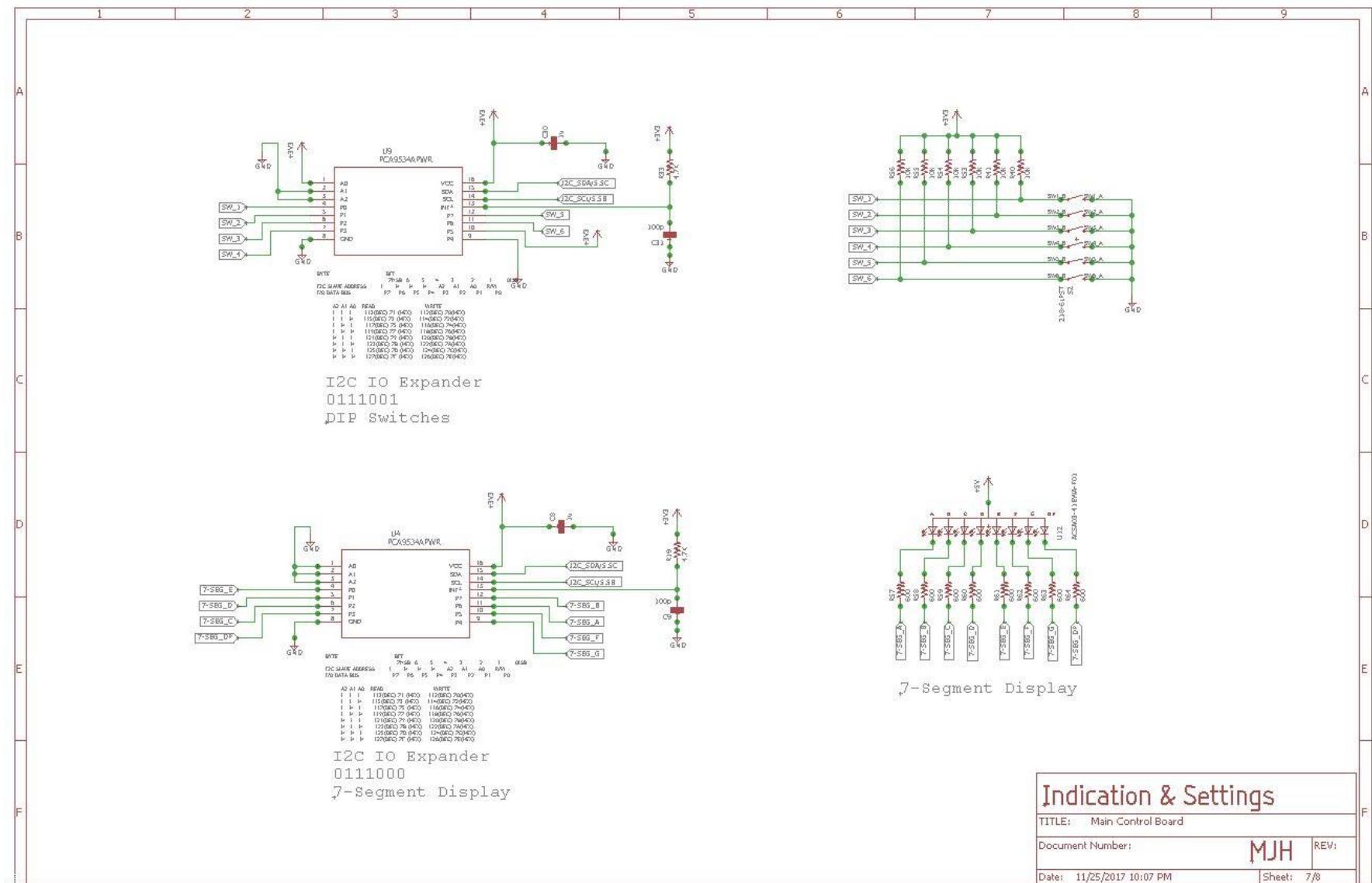


Figure 13: Main Control Board Indication & Settings Schematic

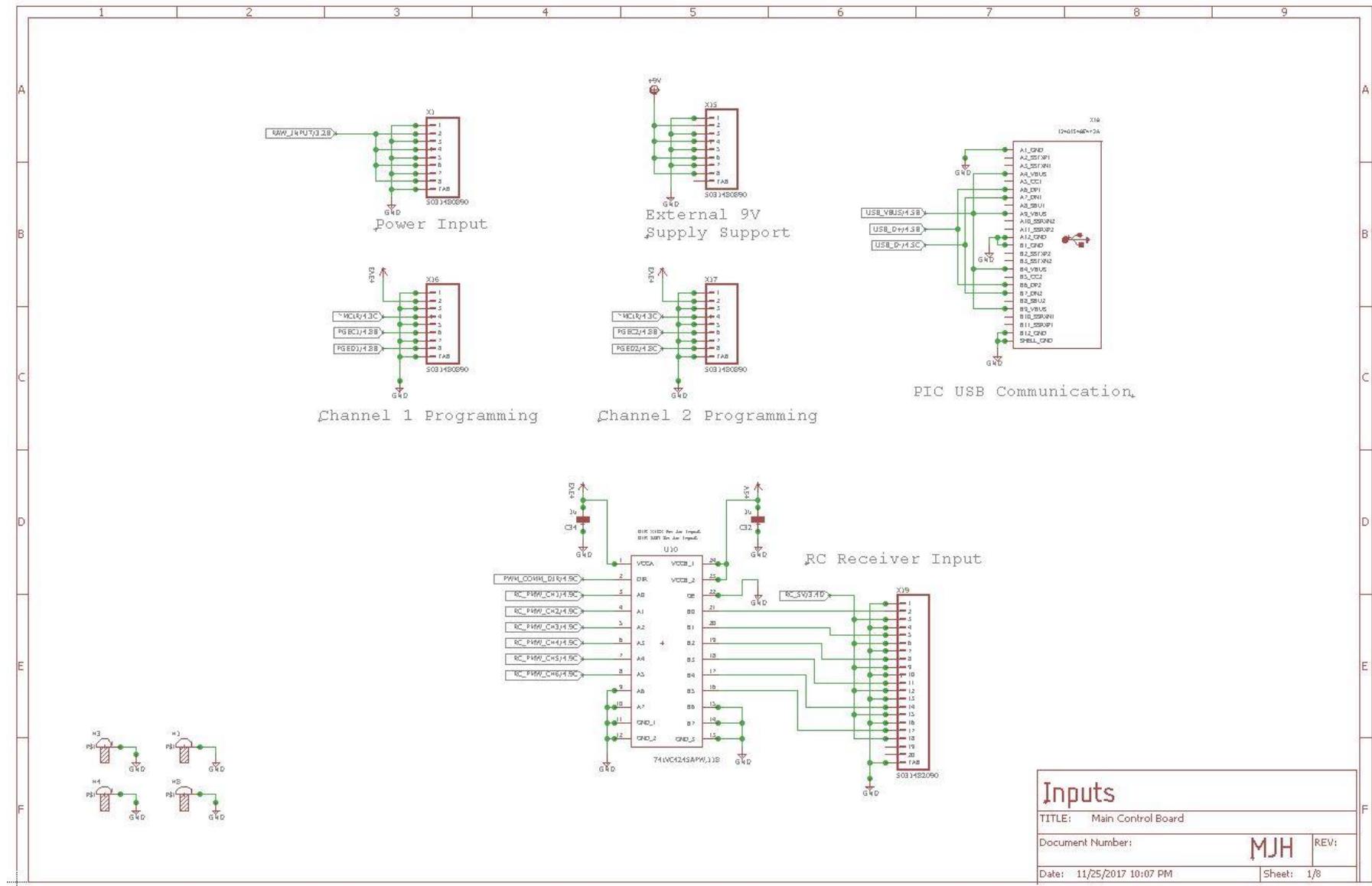


Figure 14: Main Control Board Inputs Schematic

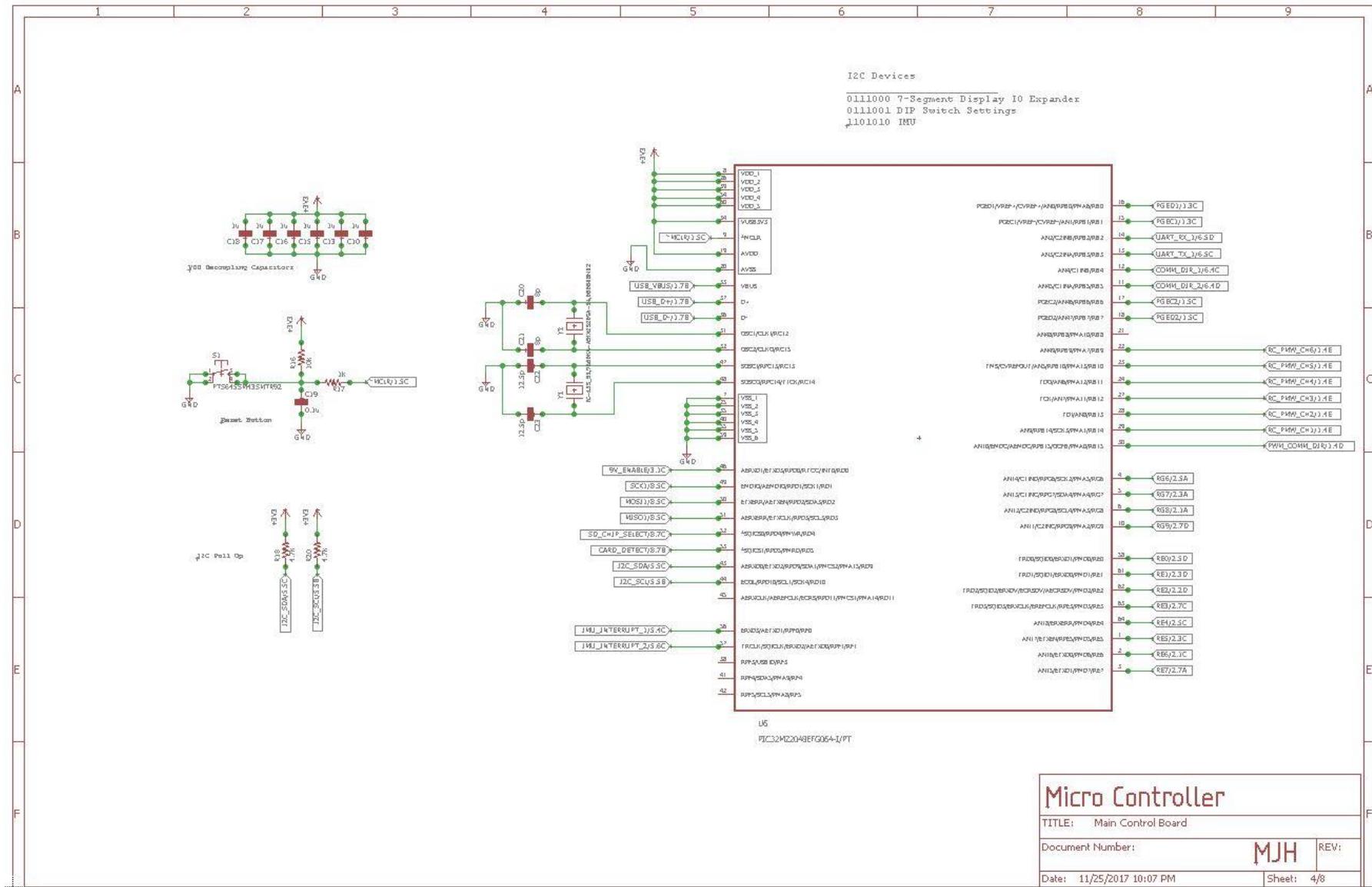


Figure 15: Main Control Board Microcontroller Schematic

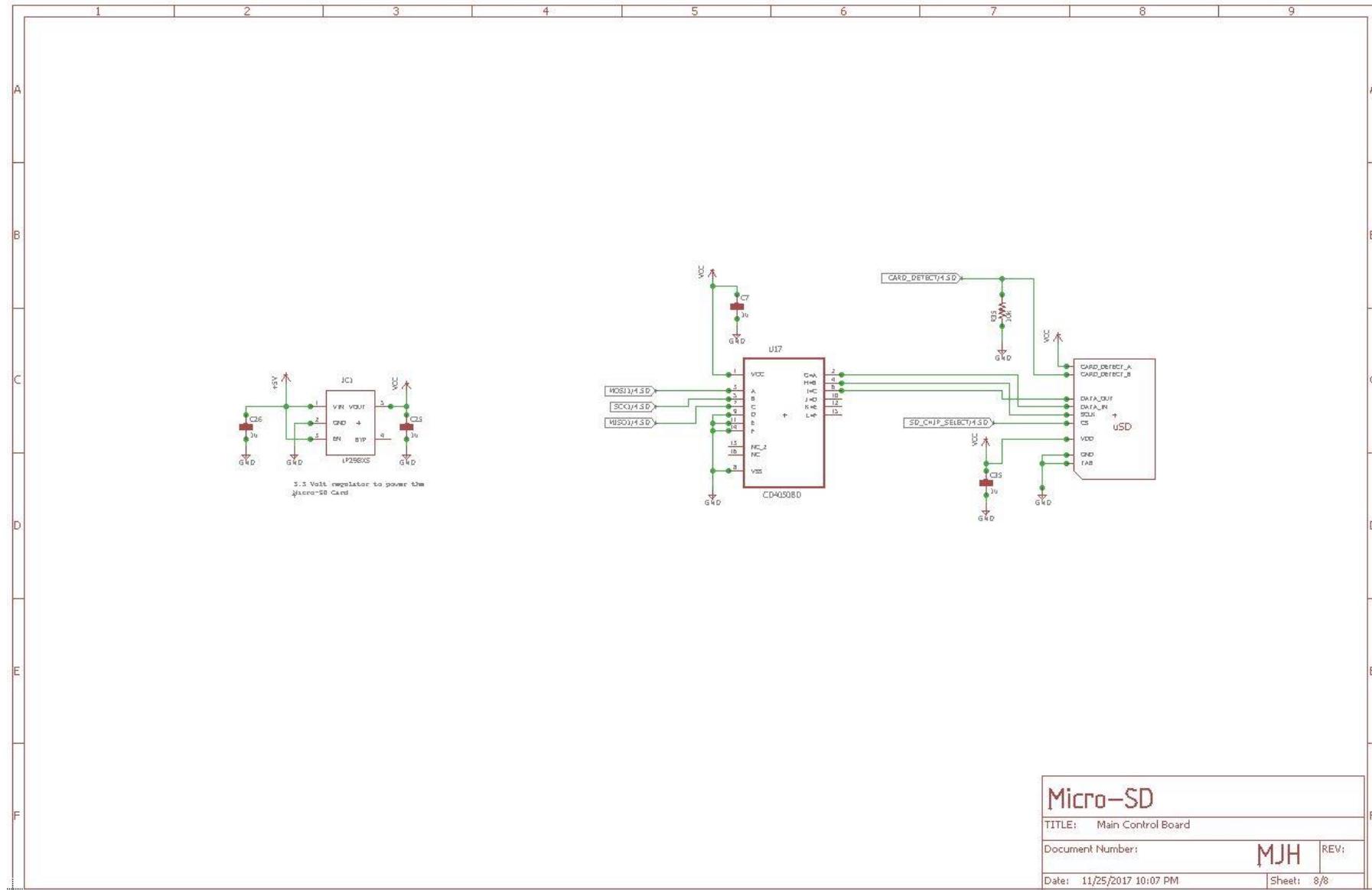


Figure 16: Main Control Board Micro-SD Schematic

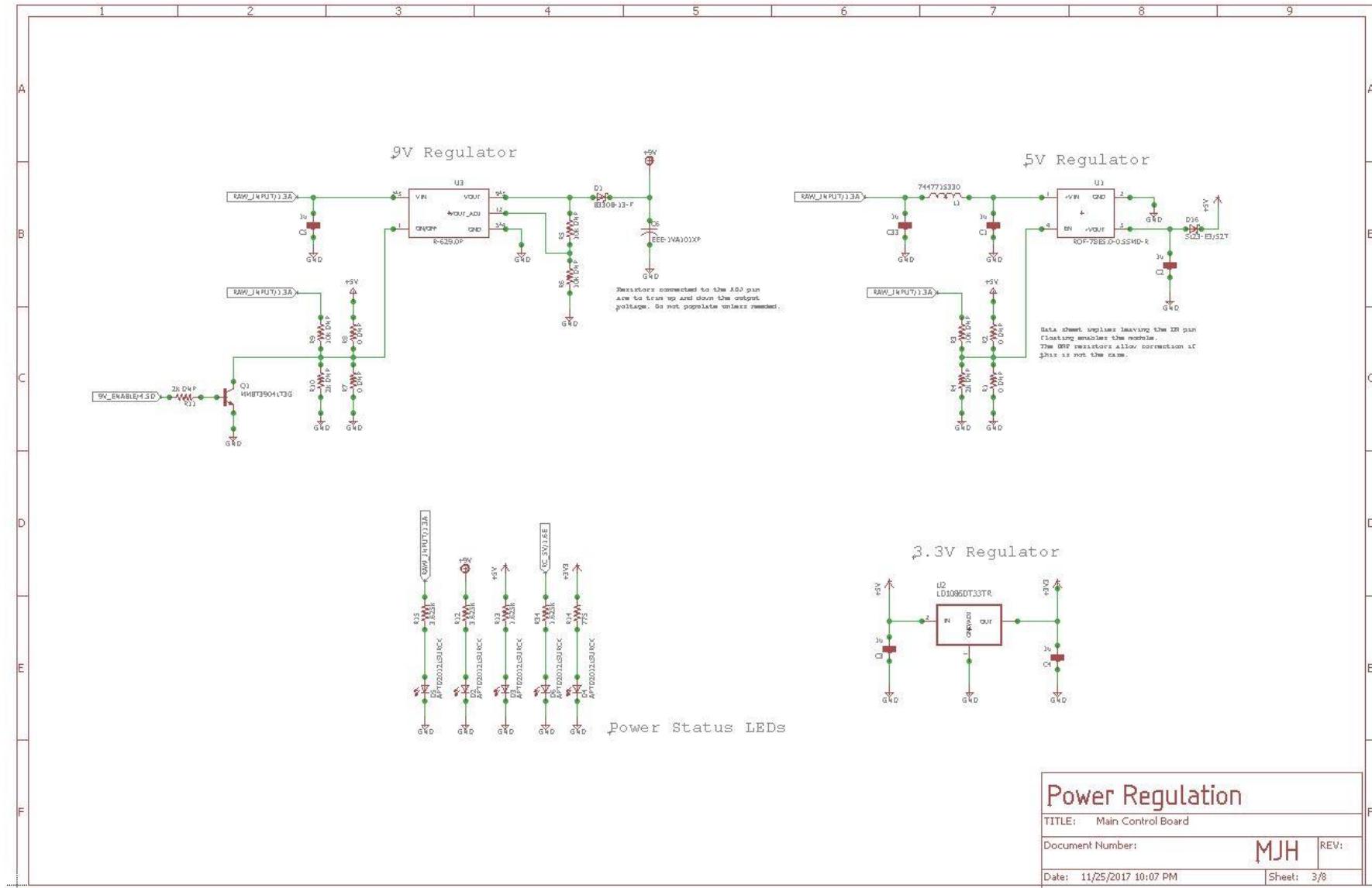


Figure 17: Main Control Board Power Regulation Schematic

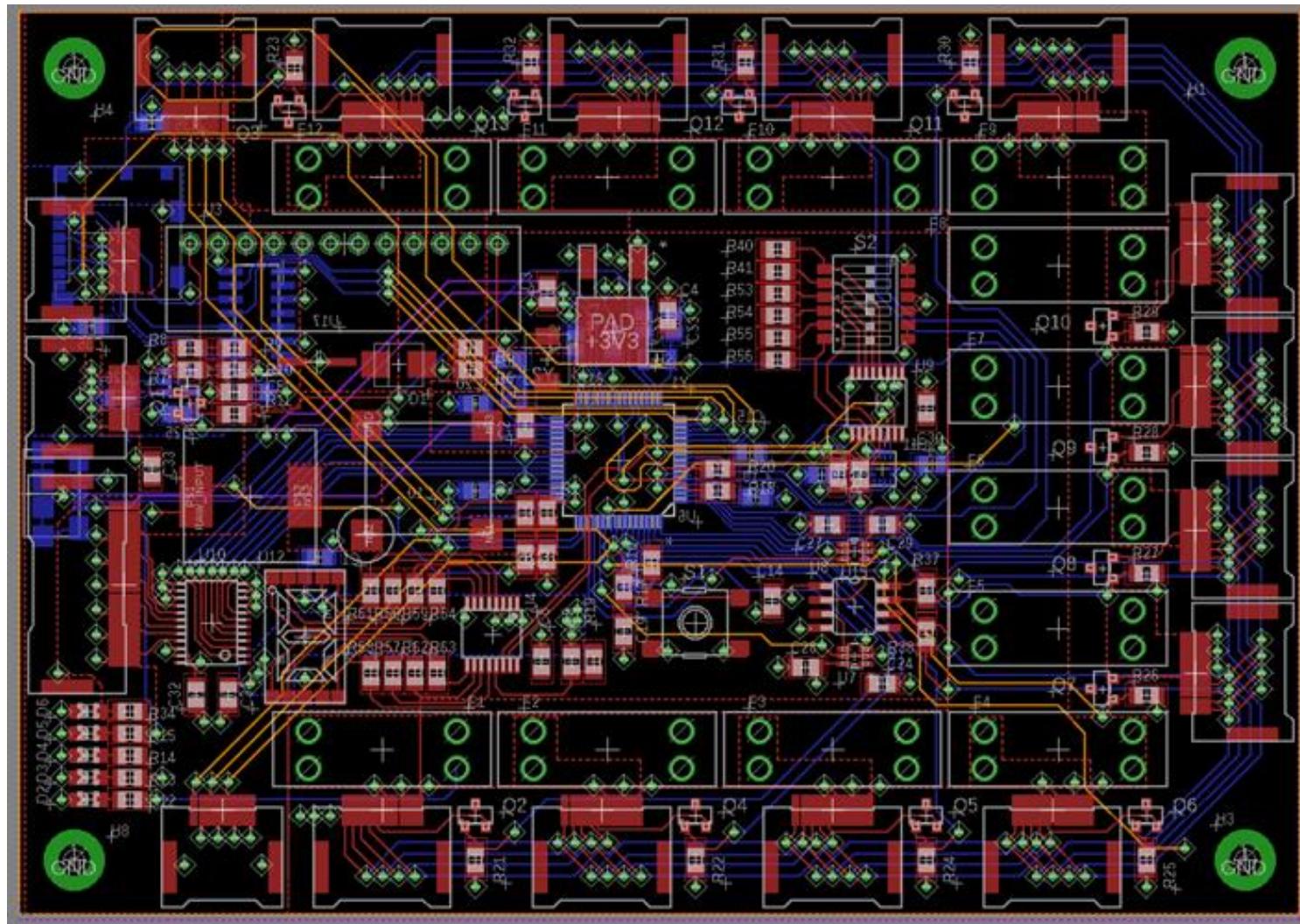


Figure 18: Main Control Board PCB Layout

3.2.4 Main Control Board Bill of Material

The bill of material in Table 31 lists the components needed to construct two of the Main Control boards. The reference designators in Table 31 correspond to the reference designators in the schematics of the Main Control board. (HL)

Qty	Refdes	Part Num.	Description
4	R1, R2, R7, R8	311-0.0ERCT-ND	0 Ohms Jumper 0.25W, 1/4W Chip Resistor 1206 (3216 Metric) Moisture Resistant Thick Film
1	C19	80-C0402T104K4RACTU	Multilayer Ceramic Capacitors MLCC - SMD/SMT .100UF 16.0V
2	R13, R34	YAG2998CT-ND	165 kOhms $\pm 1\%$ 0.063W, 1/16W Chip Resistor 0402 (1005 Metric) Moisture Resistant Thick Film
2	C11, C12	490-1318-1-ND	0.1 μ F $\pm 10\%$ 10V Ceramic Capacitor X5R 0402 (1005 Metric)
2	C9, C31	490-3160-1-ND	100pF $\pm 5\%$ 25V Ceramic Capacitor C0G, NPO 0201 (0603 Metric)
1	X2	1040310811	10 (8 + 2) Position Card Connector Secure Digital - microSD™ Surface Mount, Right Angle Gold
24	R16, R21, R22, R23, R24, R25, R26, R27, R28, R29, R30, R31, R32, R35, R40, R41, R53, R54, R55, R56, R3, R5, R6, R9	RC0805FR-0710KL	10 kOhms $\pm 1\%$ 0.125W, 1/8W Chip Resistor 0805 (2012 Metric) Moisture Resistant Thick Film
2	C22, C23	81-GRM1555C1H120GA1D	Multilayer Ceramic Capacitors MLCC - SMD/SMT 0402 12pF 50volts C0G 2%
2	R37, R38	311-120JRCT-ND	RESISTOR, American symbol
1	X18	12401548E4#2ACT-ND	USB - C USB 3.1 (USB 3.1 Gen 2, Superspeed+) Receptacle Connector 24 Position Surface Mount, Right Angle; Through Hole
1	R17	311-1.0KJRCT-ND	1 kOhms $\pm 5\%$ 0.063W, 1/16W Chip Resistor 0402 (1005 Metric) Moisture Resistant Thick Film
25	C1, C2, C3, C4, C5, C7, C8, C10, C13, C14, C15, C16, C17, C18, C24, C25, C26, C27, C28, C29, C30, C32, C33, C34, C35	1276-1946-1-ND	1 μ F $\pm 10\%$ 10V Ceramic Capacitor X7R 0603 (1608 Metric)

1	S2	218-6LPST	Dip Switch SPST 6 Position Surface Mount Slide (Standard) Actuator 25mA 24VDC
3	R4, R10, R11	311-2.00KHRCT-ND	2 kOhms $\pm 1\%$ 0.1W, 1/10W Chip Resistor 0603 (1608 Metric) Moisture Resistant Thick Film
2	R12, R15	667-ERA-2AEB362X	Thin Film Resistors - SMD 0402 1/16W 3.6Kohms
12	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12	36-3557-2-ND	Fuse Block 30A 500V 1 Circuit Blade PCB
4	R19, R33,R18, R20	311-4.7KJRCT-ND	4.7 kOhms $\pm 5\%$ 0.063W, 1/16W Chip Resistor 0402 (1005 Metric) Moisture Resistant Thick Film
4	X1, X15, X16, X17	5031480890	8 Position Receptacle Connector 0.059" (1.50mm) Surface Mount, Right Angle Tin
13	X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14,X9	5031481090	10 Position Receptacle Connector 0.059" (1.50mm) Surface Mount, Right Angle Tin
8	R57, R58, R59, R60, R61, R62, R63, R64	R-US_R0805	RESISTOR, American symbol
1	L1	7447715330	33 μ H Shielded Wirewound Inductor 2.3A 85 mOhm Max Nonstandard
1	U10	74LVC4245APW,118	Voltage Level Translator Bidirectional 1 Circuit 8 Channel 24-TSSOP
1	R14	TNP777DACT-ND	777 Ohms $\pm 0.1\%$ 0.1W, 1/10W Chip Resistor 0402 (1005 Metric) Anti-Sulfur, Automotive AEC-Q200, Moisture Resistant Thin Film
2	C20, C21	490-12681-1-ND	8pF ± 0.25 pF 25V Ceramic Capacitor C0G, NP0 0201 (0603 Metric)
1	U12	ACSA03-41EWA-F01	Character LED Display Module Red 7-Segment 1 Character Common Anode 2V 20mA 0.394" H x 0.287" W x 0.148" D (10.00mm x 7.30mm x 3.75mm) 10-SMD, No Lead
5	D2, D3, D4, D5, D6	APTD2012LSURCK	Red 630nm LED Indication - Discrete 1.75V 0805 (2012 Metric)
1	D1	B330B-13-F	Diode Schottky 30V 3A Surface Mount SMB
1	U17	CD4050BD	CMOS Hex Buffer/Converters
1	C6	EEE-1VA101XP	100 μ F 35V Aluminum Electrolytic Capacitors Radial, Can - SMD 2000 Hrs @ 85°C
1	Y1	FC-135_32.7680KA-A5	32.768kHz 720ppm Crystal 12.5pF 70 kOhms 2-SMD, No Lead
1	U2	LD1086DT33TR	Linear Voltage Regulator IC Positive Fixed 1 Output 3.3V 1.5A D-Pak
1	IC1	511-LD39100PU33R	LDO Voltage Regulators 1 A low quiescent current low noise voltage regulator

1	U5	LSM6DS3HTR	IMU ACCEL/GYRO/TEMP I2C/SPI LGA
1	U15	MAX490ESA+CSA	RS485 TRANSEIVER
13	Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12, Q13	MMBT3904LT3G	TRANS NPN 40V 0.2A SOT23
1	Y2	NX2520SA-24.000000MHZ	24MHz 710ppm Crystal 10pF 60 Ohms 4-SMD, No Lead
2	U4, U9	PCA9534APWR	I/O Expander 8 I ² C, SMBus 400kHz 16-TSSOP
1	U6	PIC32MZ2048EFG064-I/PT	IC MCU 32BIT 2MB FLASH 64TQFP
1	S1	PTS645SM43SMTR92	Tactile Switches
2	U3	R-629.0P	Non-Isolated DC/DC Converters DC/DC REG 11-32Vin 3.3-15Vout
1	U1	ROF-78E5.0-0.5SMD-R	Non-Isolated DC/DC Converters 5V 500MA OUT SMD
1	D16	SL23-E3/52T	Schottky Diodes & Rectifiers 2.0 Amp 30 Volt
2	U7, U8	SN74LVC1T45DCKR	Voltage Level Translator Bidirectional 1 Circuit 1 Channel 420Mbps SC-70-6

Table 31: Main Control Board Bill of Material

3.2.5 Main Control Board Implementation and Revisions

Because of the bulk capacitance on the PSBs the DC-DC converter on the MCB that powers the PSBs was not operating reliably. The 9-volt output of the DC-DC converter would occasionally oscillate during turn on. This unreliable behavior was corrected by the use of an external DC-DC converter that powered the PSBs. During the design of the MCB the on-board DC-DC converter, seen in Figure 17, was very close to its maximum output current rating. It was considered that the on-board DC-DC converter would not be sufficient and the ability to add an external DC-DC converter was supported as seen in Figure 14. This was accomplished with the use of a diode on the output of the on-board DC-DC converter so that an external DC-DC converter with higher output voltage could be used. This solution worked without any other complications. (MJH)

The micro-processor of the MCB, the PIC32MZ2048EFG064-I/PT, supports remappable pins that allows pins to change their function with software. The re-mapping capability is limited though, and this resulted in the UART RX pin needing corrected. The original design had UART RX connected to RB2 but this needed corrected to RF4. This was accomplished with a bodge (wire-wrap) wire. A wire was soldering from the pin of RF4 to via on the UART RX trace. The bodge wire was secured with a thick layer of conformal coating. After the fix no other issues were caused from this original design mistake. (MJH)

As with the MDB the micro-SD card holder was not populated on the MCB. This is because of the difficulty to properly populate without the proper equipment. With proper SMD population equipment and a solder-paste application stencil the micro-SD card could be populated the a micro-SD card could be used. (MJH)

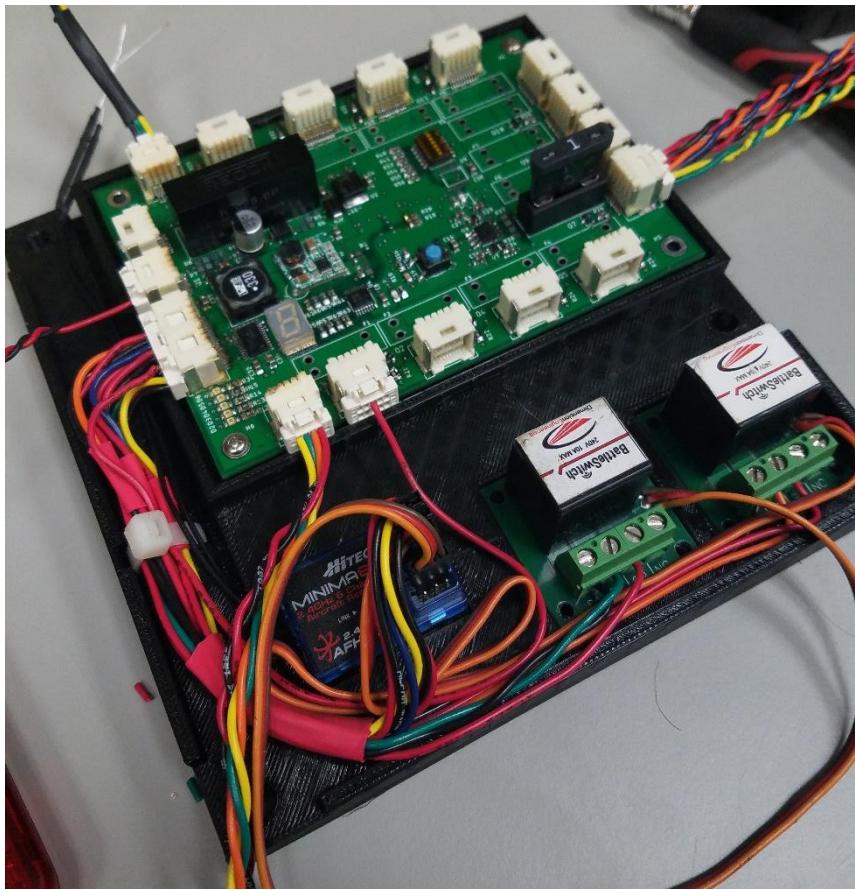


Figure 19: MCB Competed Board

3.2.6 Main Control Board Code

```
1 // ****
2 // ****
3 // Section: Included Files
4 // ****
5 // ****
6
7 #include "app.h"
8 #include "locationFunction/solveLocation.h"
9 #include "macros.h"
10 #include "motorControl.h"
11 #include "I2CFunctions.h"
12 #include "simulinkDebugger.h"
13 #include "NeuralNets/calculateSensorOne/solveSensorOne.h"
14 #include "detectEnemy.h"
15 #include "AutonomousMode.h"
16 #include "LIDARSensors.h"
17 #include "printfUART.h"
18 #include "sevenSegDigit.h"
19 #include "spinWeaponUp.h"
20 // ****
21 // ****
22 // Section: Configuration Defines, Alter code operation
23 // ****
24 // ****
25
26
27 //-----IMU CONFIG-----
28 //#define Z_AXIS_GYRO_ONLY
29 //#define GYRO_ONLY
30 #define IMU_FULL
31
32
33 //-----DEBUG CONFIG-----
34 //#define PRINT_GYRO_ANGLES
35 //#define PRINT_XL
36 //#define PRINT_LIDAR_DISTANCES
37 //#define PRINT_MOTOR_VALUES
38 #define PRINT_RC_INPUTS
39 //#define PRINT_ENEMY_SENSOR_LOCATION
40
41 //-----COMMS CONFIG-----
42 #define COMMS_ACTIVE
43 //#define COMMS_TEST
44 #define COMMS_TEST_TIMER_LENGTH_MS 10
45
46 //-----TEST CONFIG-----
47 //#define TEST_WEAPON_COMMAND_SIMPLE
48 //#define TEST_LOCOMOTION_COMMS
49 //#define TEST_MOTOR_COMMANDS_SIMPLE_RIGHT
50 //#define TEST_MOTOR_COMMANDS_SIMPLE_LEFT
51 //#define TEST_MOTOR_COMMANDS_SENSOR_FEEDBACK
52 //#define TEST_SENSOR_SYSTEM
53 //#define TEST_LETS_DO_TURN_90
54 //#define TEST_LETS_APPROACH_A_WALL_50
55 //#define TEST_RC_INPUTS
56 //#define TEST_MACRO_LIGHT
57
58
59 //# TEST_LETS_FIND_PERPENDICULAR
60 //#define TEST_LETS_RUN_WALLS
61 //#define TEST_LETS_RUN_BACK_AND_FORTH
62 //#define TEST_LETS_RUN_BACK_AND_FORTH_ENEMY
63
64
65 #define NUMBER_OF_SENSORS_TO_TEST 8
```

```

66 static int testAddressToSend= MDB_R_ADDRESS;
67
68 // ****
69 // Section: Global Data Definitions
70 // ****
71 // ****
72 // ****
73
74 // ****
75 /* Application Data
76
77 Summary:
78     Holds application data
79
80 Description:
81     This structure holds the application's data.
82
83 Remarks:
84     This structure should be initialized by the APP_Initialize function.
85
86     Application strings and buffers are defined outside this structure.
87 */
88
89 APP_DATA appData;
90
91 // ****
92 // ****
93 // Section: Application Callback Functions
94 // ****
95 // ****
96
97 /* TODO: Add any necessary callback functions.
98 */
99
100 // ****
101 // ****
102 // Section: Application Local Functions
103 // ****
104 // ****
105
106 void initializeICHandlees(void);
107 void initializeInterruptCallbacks(void);
108
109
110 // ****
111 // ****
112 // Section: Application Initialization and State Machine Functions
113 // ****
114 // ****
115
116
117 // ****
118 Function:
119     void APP_Tasks ( void )
120
121 Remarks:
122     See prototype in app.h.
123 */
124 int enemySensor=0;
125
126 void APP_Tasks ( void )
127 {
128     static timers_t taskTimer;
129     static timers_t secondTimer;
130     static timers_t tenMSTimer;

```

```

131     /* Check the application's current state. */
132     switch ( appData.state )
133     {
134         /* Application's initial state. */
135         case APP_STATE_INIT:
136         {
137
138             setupSpinUp();
139             //Init custom timer system (millis timers)
140             setTimerInterval(&taskTimer,100);
141             setTimerInterval(&secondTimer,10);
142             setTimerInterval(&tenMSTimer,10);
143             writeDigitSevenSeg(14,true);
144
145             //Finish app setup
146             bool appInitialized = true;
147             if (appInitialized)
148             {
149                 appData.state = APP_STATE_SERVICE_TASKS;
150                 writeDigitSevenSeg(15,true);
151             }
152
153             break;
154         }
155
156         case APP_STATE_SERVICE_TASKS:
157         {
158
159             //Blink the decimal at 1 second
160             blinkDecimal(1000);
161 #ifndef TEST_RC_INPUTS
162
163             //printf("ICTimer\r\n");
164             //Gather other RC input channels
165             if(updateRCInputs())
166             {
167                 setWeaponMotorSpeed((int)((getChannel6()+255)/5.0));
168                 //Capture joystick information from the input capture
169                 appData.verticalJoystick    =   getChannel12();
170                 appData.horizontalJoystick =   getChannel14();
171
172                 //Mix joystick info and store motor commands
173                 mixJoysticks(appData.verticalJoystick,appData.horizontalJoystick);
174             }
175
176 #endif
177
178             //Every 10ms
179             if(timerDone(&tenMSTimer))
180             {
181                 WDTCONbits.WDTCLRKEY=0x5743;
182                 //printf("Macros\r\n");
183                 updateMacros();
184             }
185
186             //Every 100ms
187             if(timerDone(&taskTimer))
188             {
189                 //printf("Debug\r\n");
190                 debugStuff();
191             }
192
193             if(timerDone(&secondTimer))
194             {
195                 //printf("OneSecond\r\n");

```

```

196     LATEbits.LATE7 ^=1;
197     //writeDigitSevenSegNoDec(8);
198
199     enemySensor=detectEnemy();
200
201     //double sensorOnePrediction = solveSensorOne((double
202     *)getSensorDistancesSubarray7(0));
203     //printf("Robot Angle: %f\n",robotAngle);
204     //printf("Sensor 1 prediction: %f\r\n",sensorOnePrediction);
205 }
206
207 //-----IMU Update system-----
208 #ifdef IMU_FULL
209     updateIMU();
210 #elif defined GYRO_ONLY
211     //All gyro-axis update
212     updateGyro();
213 #elif defined Z_AXIS_GYRO_ONLY
214     //Z-axis only update
215     //updateZAxis();
216 #endif
217 //-----Communications system-----
218 #ifdef COMMS_ACTIVE
219     #ifndef COMMS_TEST
220         updateCommunications();
221     #endif
222     #ifdef COMMS_TEST
223         static timers_t boardCommTestTimer;
224         static bool firstTime=true;
225         if(firstTime)
226         {
227             firstTime=false;
228
229             setTimerInterval(&boardCommTestTimer,COMMS_TEST_TIMER_LENGTH
230             TH_MS);
231         }
232         if(timerDone(&boardCommTestTimer))
233     {
234         #ifdef TEST_SENSOR_SYSTEM
235             if(testBoardComms(testAddressToSend))
236             {
237                 testAddressToSend++;
238
239                 if(testAddressToSend>=PSB_1_ADDRESS+NUMBER_OF_SENSORS_
240                 TO_TEST)
241                 {
242                     testAddressToSend= PSB_1_ADDRESS;
243                 }
244             #elif defined(TEST_LOCOMOTION_COMMs)
245                 if(testBoardComms(testAddressToSend))
246                 {
247                     if(testAddressToSend==MDB_R_ADDRESS)
248                     {
249                         testAddressToSend= MDB_L_ADDRESS;
250                     }
251                     else
252                     {
253                         testAddressToSend = MDB_R_ADDRESS;
254                     }
255                 }
256             }
257         }
258     }
259
260 #endif

```

```

256                         testBoardComms(testAddressToSend);
257     #endif
258     }
259     #endif
260     #endif
261     break;
262 }
263 /* The default state should never be executed. */
264 default:
265 {
266     /* TODO: Handle error in application's state machine. */
267     break;
268 }
269 }
270 */
271 ****
272 Function:
273 void APP_Initialize ( void )
274
275 Remarks:
276 See prototype in app.h.
277 */
278 void APP_Initialize ( void )
279 {
//    //MISC PIN SETS (PROB UNNECESSARY)
//    TRISBbits.TRISB2= 1;
//    LATBbits.LATB4 = 1;
//    LATBbits.LATB5 = 0;
//    /* Place the App state machine in its initial state. */
//    appData.state = APP_STATE_INIT;
//    //TIMERS
//    SYS_INT_SourceEnable(INT_SOURCE_TIMER_1);
//    //SYS_INT_SourceEnable(INT_SOURCE_TIMER_2);
//    PLIB_TMR_Start(TMR_ID_1);
//    PLIB_TMR_Start(TMR_ID_2);
//    PLIB_TMR_Start(TMR_ID_4);

293 WDTCONbits.WDTCLRKEY=0x5743;
294 //I2C
295 appData.I2CHandle = DRV_I2C_Open(sysObj.drvI2C0,DRV_IO_INTENT_READWRITE);
296
297 //Setup smart I2C handling timeouts
298 setupI2CFunctions();
299
300 WDTCONbits.WDTCLRKEY=0x5743;
301 //Run seven seg for mutex init
302 initSevenSeg();
303 writeDigitSevenSeg(0,true);
304
305 //UART
306 initializeUARTHandles();
307
308 WDTCONbits.WDTCLRKEY=0x5743;
309 //Setup printf redirect
310 setupPrintf(appData.USART1Handle);
311
312 WDTCONbits.WDTCLRKEY=0x5743;
313 //UART Buffer system init
314
315 initUARTBufferSystem(appData.USART0Handle,appData.USART1Handle,appData.USART2H
316 andle,appData.USART3Handle);
317 writeDigitSevenSeg(10,true);
318 WDTCONbits.WDTCLRKEY=0x5743;
319 //Initialize Input Capture Handles (RC Receiver)

```

```

319     initializeICHandlees();
320     writeDigitSevenSeg(11,true);
321
322     WDTCONbits.WDTCLRKEY=0x5743;
323     //Setup the FastTransfer communications system
324     setupCommunications();
325     writeDigitSevenSeg(12,true);
326     //    while(1)
327     //    {
328     //        sendDataSimulink();
329     //    }
330
331     WDTCONbits.WDTCLRKEY=0x5743;
332     //Initialize IMU
333 #if (defined Z_AXIS_GYRO_ONLY) || (defined GYRO_ONLY) || (defined IMU_FULL)
334     initIMU();
335     writeDigitSevenSeg(13,true);
336 #endif
337
338     WDTCONbits.WDTCLRKEY=0x5743;
339     //Solver neural net bootup
340     initLocationSolver();
341     initEnemySolver();
342     //initSensorOneSolver();
343     setupAutonomous();
344     //WDTCONbits.ON=0;
345 }
346
347 APP_STATES getApplicationState(void)
348 {
349     return appData.state;
350 }
351
352 void initializeICHandlees(void)
353 {
354     DRV_IC0_Start();
355     DRV_IC1_Start();
356     //setupTimerHandleRC();
357 }
358
359 timers_t RCDebugTimer;
360 bool firstTime=true;
361 void debugStuff(void)
362 {
363
364     //resetI2CSYSTEM();
365     #ifdef PRINT_MOTOR_VALUES
366         printf("lm: %d, rm: %d, weapon:
367             %d\r\n",getLeftMotorSpeed(),getRightMotorSpeed(),getWeaponMotorSpeed());
368     #endif
369     #ifdef TEST_MOTOR_COMMANDS_SIMPLE_RIGHT
370         testAddressToSend = MDB_R_ADDRESS;
371     #endif
372     #ifdef TEST_MOTOR_COMMANDS_SIMPLE_LEFT
373         testAddressToSend = MDB_L_ADDRESS;
374     #endif
375     #ifdef TEST_WEAPON_COMMAND_SIMPLE
376         testAddressToSend = WMCB_ADDRESS;
377     #endif
378     #ifdef TEST_SENSOR_SYSTEM
379         //testAddressToSend = PSB_1_ADDRESS;
380     #endif

```

```

383
384     #if defined(PRINT_GYRO_ANGLES) && defined(PRINT_XL_ANGLES)
385         printf("Ang %d, %d, %d ",getXAxisAngle(),getYAxisAngle(),getZAxisAngle());
386         printf("XL %4.3f, %4.3f,
387             %4.3f\n",getXAxisAccel(),getYAxisAccel(),getZAxisAccel());
388     #elif defined PRINT_GYRO_ANGLES
389         printf("Angles %4.3f, %4.3f,
390             %4.3f\r\n",getXAxisAngle(),getYAxisAngle(),getZAxisAngle());
391     #elif defined PRINT_XL
392         printf("XL %4.3f, %4.3f,
393             %4.3f\r\n\r\n",getXAxisAccel(),getYAxisAccel(),getZAxisAccel());
394     #endif
395
396     #ifdef PRINT_LIDAR_DISTANCES
397         double * sensorD;
398         sensorD=(double *)getSensorDistancesArray();
399         printf("Sensor Array: ");
400         int i;
401         for(i=0;i<8;i++)
402         {
403             printf("%4.3f",sensorD[i]);
404             printf(" ");
405         }
406         printf("\r\n\r\n");
407     #endif
408     #ifdef PRINT_ENEMY_SENSOR_LOCATION
409 //Printout enemy location
410
411         printf("X: %f, Y: %f\r\n", getXPos(),getYPos());
412         if(enemySensor>=0)
413         {
414             printf("Detected Enemy: %d\r\n\r\n", enemySensor+1);
415         }
416         else
417         {
418             printf("No Enemy\r\n\r\n");
419         }
420     #endif
421
422     #ifdef TEST_LETS_DO_TURN_90
423         doTurn(90);
424         while(1);
425     #endif
426     #ifdef TEST_LETS_APPROACH_A_WALL_50
427         doApproach(50);
428         while(1);
429     #endif
430     #ifdef TEST_LETS_FIND_PERPENDICULAR
431         printf("Finding Perp\r\n");
432         setMacroCommand(PERPENDICULAR);
433         findPerpendicular(2);
434
435         printf("Done Finding Perp\r\n");
436     #endif
437     #ifdef TEST_LETS_RUN_WALLS
438         setMacroCommand(PERPENDICULAR);
439         runWalls();
440         while(1);
441     #endif
442     #ifdef TEST_LETS_RUN_BACK_AND_FORTH
443         setMacroCommand(PERPENDICULAR);
444         runBackAndForth();
445         while(1);

```

```

445     #endif
446     #ifdef TEST_LETS_RUN_BACK_AND_FORTH_ENEMY
447         setMacroCommand(PERPENDICULAR);
448         runBackAndForthFindEnemy();
449         while(1);
450     #endif
451
452     #ifdef TEST_RC_INPUTS
453         if(firstTime)
454         {
455             setTimerInterval(&RCDebugTimer,1000);
456             firstTime=false;
457         }
458
459         if(timerDone(&RCDebugTimer))
460         {
461             //Gather other RC input channels
462             if(updateRCInputs())
463             {
464                 printf("RCInputs Active\r\n");
465                 printf("\r\n");
466             }
467             else
468             {
469                 printf("\r\n");
470                 printf("\r\n");
471             }
472         }
473
474
475     #endif
476     #ifdef PRINT_RC_INPUTS
477         printf("Chan 1: %d\r\n",getChannel1());
478         printf("Chan 2: %d\r\n",getChannel2());
479         printf("Chan 3: %d\r\n",getChannel3());
480         printf("Chan 4: %d\r\n",getChannel4());
481         printf("Chan 5: %d\r\n",getChannel5());
482         printf("Chan 6: %d\r\n\r\n",getChannel6());
483     #endif
484
485     #ifdef TEST_MACRO_LIGHT
486         TRISBbits.TRISB7 = 0;
487         static bool toggleLEDDD=true;
488         if(toggleLEDDD)
489         {
490             LATBbits.LATB7 =0;
491             toggleLEDDD=0;
492         }
493         else
494         {
495             LATBbits.LATB7 =1;
496             toggleLEDDD=1;
497         }
498     }
499
500     #endif
501 }
502
503
504
505
506 ****
507 End of File
508 */
509

```

```

1  #include "AutonomousMode.h"
2  #include "helperFunctions.h"
3  #include "IMU.h"
4  #include "communications.h"
5  #include "PID.h"
6  #include "motorControl.h"
7  #include "FastTransfer.h"
8  #include "driveAssistance.h"
9  #include "macros.h"
10 #include "RCInputs.h"
11 #include "LIDARSensors.h"
12 #include "MillisTimer.h"
13 #include "runBackAndForthFindEnemyBoth.h"
14 #include "spinWeaponUp.h"
15 #include "enemyAssumedToBeInFront.h"
16 #include "runWalls.h"
17 #include "detectEnemy.h"
18 #include "doTurn.h"
19
20 #define WEAPON_SPEED_SETTING 25
21
22 #define ATTACK_SPEED 40
23
24 #define MAX_SPEED_ATTACK 50
25 #define MIN_SPEED_ATTACK -50
26
27 #define CLOCKWISE true
28 #define COUNTERCLOCKWISE false
29
30 timers_t detectEnemyTimer;
31 timers_t enemyTimeout;
32 timers_t checkDetectTimer;
33 timers_t attackCycleTimeout;
34 timers_t macroLightFoundYouTimer;
35 timers_t macroLightAttackTimer;
36 timers_t debugTimer;
37 timers_t holdEnemyMemoryTimer;
38
39 bool cw_ccw=CLOCKWISE;
40 bool blinkLight=true;
41 int enemySensor=-1;
42 int numberOfShimmies=0;
43
44 int boundMotorCommandsAttack(int val);
45
46 void setupAutonomous(void)
47 {
48     setTimerInterval(&detectEnemyTimer,10);
49     setTimerInterval(&enemyTimeout,2000);
50     setTimerInterval(&checkDetectTimer,500);
51     setTimerInterval(&attackCycleTimeout,5000);
52     setTimerInterval(&macroLightFoundYouTimer,100);
53     setTimerInterval(&macroLightAttackTimer,50);
54     setTimerInterval(&debugTimer,100);
55     setTimerInterval(&holdEnemyMemoryTimer,500);
56 }
57
58 void runAutonomous(void)
59 {
60     spinWeaponUp(WEAPON_SPEED_SETTING);
61
62     while(checkMacroExists())
63     {
64         MACRO_LIGHT=1;
65         runWallsFindEnemy(WEAPON_SPEED_SETTING);

```

```

66
67     bool enemyFound=false;
68     resetTimer(&attackCycleTimeout);
69     while(!timerDone(&attackCycleTimeout) && checkMacroExists())
70     {
71         if(timerDone(&macroLightFoundYouTimer))
72         {
73             MACRO_LIGHT=blinkLight;
74             blinkLight=!blinkLight;
75         }
76
77         resetTimer(&enemyTimeout);
78         numberOShimmies=0;
79
80         while(!timerDone(&enemyTimeout) && checkMacroExists())
81         {
82             if(timerDone(&macroLightFoundYouTimer))
83             {
84                 MACRO_LIGHT=blinkLight;
85                 blinkLight=!blinkLight;
86             }
87
88             updateCommunications();
89             updateRCInputs();
90             //Update the gyroscope values
91             updateIMU();
92             //
93             //
94             //
95             //
96
97             uint8_t speed = (int)((getChannel16() + 255)/5.0);
98             if(speed>WEAPON_SPEED_SETTING)
99             {
100                 setWeaponMotorSpeed(speed);
101             }
102             else
103             {
104                 setWeaponMotorSpeed(WEAPON_SPEED_SETTING);
105             }
106
107             enemySensor=getEnemyLocation();
108             if(timerDone(&checkDetectTimer))
109             {
110                 if(enemySensor==0)
111                 {
112                     enemyFound=true;
113                     break;
114                 }
115                 else
116                 {
117                     if(cw_ccw == CLOCKWISE)
118                     {
119                         doTurn(15+numberOShimmies);
120                         cw_ccw=COUNTERCLOCKWISE;
121                     }
122                     else
123                     {
124                         doTurn(-15-numberOShimmies);
125                         cw_ccw=CLOCKWISE;
126                         numberOShimmies+=2;
127                     }
128                 }
129             }
130         }
    }

```

```

131
132
133 //ATTACK
134 resetTimer(&holdEnemyMemoryTimer);
135 cw_ccw=!cw_ccw;
136
137 setGyroRelativeAngle();
138 if(enemyFound)
139 {
140     while((enemySensor==0 || !timerDone(&holdEnemyMemoryTimer)) &&
141         checkMacroExists())
142     {
143         if(timerDone(&macroLightAttackTimer))
144         {
145             MACRO_LIGHT=blinkLight;
146             blinkLight=!blinkLight;
147         }
148         updateCommunications();
149         updateRCInputs();
150         //Update the gyroscope values
151         updateIMU();
152
153         uint8_t speed = (int)((getChannel6()+255)/5.0);
154         if(speed>WEAPON_SPEED_SETTING)
155         {
156             setWeaponMotorSpeed(speed);
157         }
158         else
159         {
160             setWeaponMotorSpeed(WEAPON_SPEED_SETTING);
161         }
162
163         enemySensor=getEnemyLocation();
164
165         int _lm = ATTACK_SPEED;
166         int _rm = ATTACK_SPEED;
167
168         motorGyroFeedbackStraight(&_lm, &_rm);
169         //Send to the motor system to be sent by the communications
170         library
171
172         setLeftMotorSpeed(boundMotorCommands(boundMotorCommandsAttack(
173             _lm)));
174
175         setRightMotorSpeed(boundMotorCommands(boundMotorCommandsAttack(
176             _rm)));
177     }
178
179 }
180 void runAutonomous2(void)
181 {
182     spinWeaponUp(WEAPON_SPEED_SETTING);
183
184     while(checkMacroExists())
185     {
186         MACRO_LIGHT=1;
187         runWallsFindEnemy(WEAPON_SPEED_SETTING);
188
189         bool enemyFound=false;

```

```

190     resetTimer(&attackCycleTimeout);
191     while(!timerDone(&attackCycleTimeout) && checkMacroExists())
192     {
193         if(timerDone(&macroLightFoundYouTimer))
194         {
195             MACRO_LIGHT=blinkLight;
196             blinkLight=!blinkLight;
197         }
198
199         resetTimer(&enemyTimeout);
200         numberOfWorkShimmies=0;
201
202         while(!timerDone(&enemyTimeout) && checkMacroExists())
203         {
204             if(timerDone(&macroLightFoundYouTimer))
205             {
206                 MACRO_LIGHT=blinkLight;
207                 blinkLight=!blinkLight;
208             }
209
210             updateCommunications();
211             updateRCInputs();
212             //Update the gyroscope values
213             updateIMU();
214
215             //if(timerDone(&debugTimer))
216             //{
217                 //printf("Looking, see: %d\r\n", enemySensor);
218             //}
219             uint8_t speed = (int)((getChannel16() + 255) / 5.0);
220             if(speed>WEAPON_SPEED_SETTING)
221             {
222                 setWeaponMotorSpeed(speed);
223             }
224             else
225             {
226                 setWeaponMotorSpeed(WEAPON_SPEED_SETTING);
227             }
228
229             enemySensor=getEnemyLocation();
230             if(timerDone(&checkDetectTimer))
231             {
232                 if(enemySensor==0)
233                 {
234                     enemyFound=true;
235                     break;
236                 }
237                 else
238                 {
239                     if(cw_ccw == CLOCKWISE)
240                     {
241                         doTurn(15+numberOfWorkShimmies);
242                         cw_ccw=COUNTERCLOCKWISE;
243                     }
244                     else
245                     {
246                         doTurn(-15-numberOfWorkShimmies);
247                         cw_ccw=CLOCKWISE;
248                         numberOfWorkShimmies+=2;
249                     }
250                 }
251                 resetTimer(&checkDetectTimer);
252             }
253         }
254     } //ATTACK

```

```

255     resetTimer(&holdEnemyMemoryTimer);
256     cw_ccw=!cw_ccw;
257
258     setGyroRelativeAngle();
259     if(enemyFound)
260     {
261         while((enemySensor==0 || !timerDone(&holdEnemyMemoryTimer)) &&
262             checkMacroExists())
263         {
264             if(timerDone(&macroLightAttackTimer))
265             {
266                 MACRO_LIGHT=blinkLight;
267                 blinkLight=!blinkLight;
268             }
269             updateCommunications();
270             updateRCInputs();
271
272             //Update the gyroscope values
273             updateIMU();
274
275             uint8_t speed = (int)((getChannel6()+255)/5.0);
276             if(speed>WEAPON_SPEED_SETTING)
277             {
278                 setWeaponMotorSpeed(speed);
279             }
280             else
281             {
282                 setWeaponMotorSpeed(WEAPON_SPEED_SETTING);
283             }
284
285             enemySensor=getEnemyLocation();
286
287             int _lm = ATTACK_SPEED;
288             int _rm = ATTACK_SPEED;
289
290             motorGyroFeedbackStraight(&_lm, &_rm);
291
292             //Send to the motor system to be sent by the communications
293             library
294
295             setLeftMotorSpeed(boundMotorCommands(boundMotorCommandsAttack(
296                 _lm)));
297
298             setRightMotorSpeed(boundMotorCommands(boundMotorCommandsAttack(
299                 _rm)));
300         }
301     }
302     int boundMotorCommandsAttack(int val)
303     {
304         if(val>MAX_SPEED_ATTACK)
305         {
306             val=MAX_SPEED_ATTACK;
307         }
308         else if(val<MIN_SPEED_ATTACK)
309         {
310             val=MIN_SPEED_ATTACK;
311         }
312         return val;
313     }

```

```

1 // ****
2 // ****
3 // Communications.c
4 // -- Handles Communications between MCB and other boards
5 // ****
6 // ****
7
8
9 #include "communications.h"
10 #include "LIDARSensors.h"
11 #include "FastTransfer.h"
12 #include "uartHandler.h"
13 #include "MillistTimer.h"
14 #include "motorControl.h"
15 #include "detectEnemy.h"
16
17 // #define DISABLE_WMCB
18 // #define DISABLE_SENSORS
19 // #define IGNORE_MDB_RESPONSE
20 // #define DISABLE_MOTORS
21 #define NUMBER_OF_SENSORS      8
22 #define MIN_WEAPON_SPEED      50
23 #define WEIGHT_OLD             90.0
24 #define WEIGHT_NEW              10.0
25
26 int receiveArray[4];
27 //timer_t sendTimer;
28 timers_t resendTimer;
29 timers_t receivedResendTimer;
30 timers_t receiveTimer;
31 timers_t sendMotorCommand,debugSevenTimer;
32 timers_t detectEnemyTimer;
33
34 unsigned char slaveAddress = 0;
35 bool receivedResponse = true;
36 uint16_t numberofRetries = 0;
37 int enemyLocation=0;
38
39 void retrySystem(void);
40 void populateSlaveDataFT(unsigned char slaveNum);
41 bool receiveDataFT(unsigned char slaveNum);
42 void incrementSlaveCounter(void);
43
44 bool weaponUpToSpeed(void)
45 {
46 //    if(weaponSpeed>MIN_WEAPON_SPEED)
47 //    {
48 //        return true;
49 //    }
50 //    else
51 //    {
52 //        return false;
53 //    }
54    return true;
55 }
56
57 void setupCommunications(void)
58 {
59     setTimerInterval(&debugSevenTimer,100);           //Timer to control debugging
60     display
61     setTimerInterval(&resendTimer,30);                //Resend timer during comms
62     dropout
63     setTimerInterval(&receivedResendTimer,3);         //How quickly to resend after
64     receiving
65     setTimerInterval(&sendMotorCommand,5);            //When open loop comms to MDBs

```

```

63     setTimerInterval(&detectEnemyTimer,10);      //When open loop comms to MDBs
64     begin(receiveArray, sizeof(receiveArray), MCU_ADDRESS, false,
65     writeByteUART0, readByteUART2, rxBytesAvailable2, peekByteUART2);
66     //stopAllMotors();
67 }
68 int getEnemyLocation(void)
69 {
70     return enemyLocation;
71 }
72 void updateCommunications(void)
73 {
74     if(timerDone(&detectEnemyTimer))
75     {
76         enemyLocation=detectEnemy();
77     }
78     WDTCONbits.WDTCLRKEY=0x5743;
79     if(!receivedResponse)
80     {
81         //int recPacket=0;
82         while(receiveData())
83         {
84             //printf("Heard packet\r\n");
85             //        recPacket++;
86             //        if(recPacket>1)
87             //        {
88             //            printf("RecPacket: %d\r\n", recPacket);
89             //        }
90             //        resetTimer(&recievedResendTimer);
91             //        resetTimer(&resendTimer);
92             //        if(timerDone(&debugSevenTimer))
93             //            writeDigitSevenSeg(2,true);
94             //        if(receiveDataFT(slaveAddress))
95             {
96                 //printf("Got it\r\n");
97                 //break;
98             }
99         }
100    }
101   //    if(receiveData())
102   //    {
103   //        //printf("Rec\r\n");
104   //        resetTimer(&recievedResendTimer);
105   //        if(timerDone(&debugSevenTimer))
106   //            writeDigitSevenSeg(2,true);
107   //        receiveDataFT(slaveAddress);
108   //    }
109   //    else
110   //    {
111   //        //if(timerDone(&debugSevenTimer))
112   //        //    writeDigitSevenSeg(1,true);
113   //    }
114 }
115
116
117
118 //If its time to send a request for data/data out
119 if(receivedResponse && timerDone(&recievedResendTimer))
120 {
121     //Send the data
122     //wipeBuffer2();
123     populateSlaveDataFT(addressList[slaveAddress]);
124     sendData(addressList[slaveAddress]);
125
126     //printf("sendingOut %d\r\n", slaveAddress);

```

```

127         //Reset the resend timer for retries
128         resetTimer(&resendTimer);
129
130         //writeDigitSevenSeg(slaveAddress,true);
131         //Mark that we are waiting for a response
132         receivedResponse=false;
133
134         //Clear the retry counter
135         numberOfRetries      = 0;
136     }
137 #ifdef IGNORE_MDB_RESPONSE
138 //    else if(timerDone(&sendMotorCommand))
139 //    {
140 //        static uint8_t motorCounter=0;
141 //        motorCounter++;
142 //        if(motorCounter>1)
143 //        {
144 //            motorCounter=0;
145 //        }
146 //        populateSlaveDataFT(addressList[motorCounter]);
147 //        sendData(addressList[motorCounter]);
148 //    }
149 #endif
150
151     //Retry slaves if not responding
152     retrySystem();
153 }
154
155
156 bool receiveDataFT(unsigned char slaveNum)
157 {
158     //printf("Response from %d\r\n", addressList[slaveNum]);
159     //printf("%d\r\n", receiveArray[0]);
160 //    if((receiveArray[0]==addressList[slaveNum])||(addressList[slaveNum] ==
161 //MDB_L_ADDRESS)|| (addressList[slaveNum] == MDB_R_ADDRESS))
162 //    {
163         switch(receiveArray[0]//addressList[slaveNum])
164         {
165             case MDB_L_ADDRESS:
166                 //printf("MDB_L\r\n");
167                 break;
168             case MDB_R_ADDRESS:
169                 //printf("MDB_R\r\n");
170                 break;
171             case WMCB_ADDRESS:
172                 //printf("WMCB\r\n");
173                 break;
174             case PSB_1_ADDRESS:
175                 setSensorDistance(0,(receiveArray[1]*WEIGHT_OLD +
176                     getSensorDistance(0)*WEIGHT_NEW)/100.0);
177                 break;
178             case PSB_2_ADDRESS:
179                 setSensorDistance(1,(receiveArray[1]*WEIGHT_OLD +
180                     getSensorDistance(1)*WEIGHT_NEW)/100.0);
181                 break;
182             case PSB_3_ADDRESS:
183                 setSensorDistance(2,(receiveArray[1]*WEIGHT_OLD +
184                     getSensorDistance(2)*WEIGHT_NEW)/100.0);
185                 break;
186             case PSB_4_ADDRESS:
187                 setSensorDistance(3,(receiveArray[1]*WEIGHT_OLD +
188                     getSensorDistance(3)*WEIGHT_NEW)/100.0);
189                 break;
190             case PSB_5_ADDRESS:
191                 setSensorDistance(4,(receiveArray[1]*WEIGHT_OLD +

```

```

187         getSensorDistance(4)*WEIGHT_NEW)/100.0);
188         break;
189     case PSB_6_ADDRESS:
190         setSensorDistance(5,(receiveArray[1]*WEIGHT_OLD +
191             getSensorDistance(5)*WEIGHT_NEW)/100.0);
192         break;
193     case PSB_7_ADDRESS:
194         setSensorDistance(6,(receiveArray[1]*WEIGHT_OLD +
195             getSensorDistance(6)*WEIGHT_NEW)/100.0);
196         break;
197     case PSB_8_ADDRESS:
198         setSensorDistance(7,(receiveArray[1]*WEIGHT_OLD +
199             getSensorDistance(7)*WEIGHT_NEW)/100.0);
200         break;
201     }
202
203     //mark that we have heard back
204     receivedResponse = true;
205
206     //Since we have heard back, lets move onto the next slave
207     incrementSlaveCounter();
208
209     //resetTimer(&resendTimer); //TESTING ONLY
210     receivedResponse = false;
211
212     return false;
213 }
214
215 void populateSlaveDataFT(unsigned char slaveNum)
216 {
217     switch(slaveNum)
218     {
219         case MDB_L_ADDRESS:
220             ToSend(SPEED_SETTING_DATA_ADDRESS,getLeftMotorSpeed());
221             break;
222         case MDB_R_ADDRESS:
223             ToSend(SPEED_SETTING_DATA_ADDRESS,getRightMotorSpeed());
224             break;
225         case WMCB_ADDRESS:
226             //printf("WMCB send\r\n");
227             ToSend(SPEED_SETTING_DATA_WEAPON_ADDRESS,getWeaponMotorSpeed());
228             break;
229         case PSB_1_ADDRESS:
230             ToSend(0,0);
231             //printf("Sending to PSB\r\n");
232             break;
233         case PSB_2_ADDRESS:
234             ToSend(0,0);
235             break;
236         case PSB_3_ADDRESS:
237             ToSend(0,0);
238             break;
239         case PSB_4_ADDRESS:
240             ToSend(0,0);
241             break;
242         case PSB_5_ADDRESS:
243             ToSend(0,0);
244             break;
245     }
246 }
247

```

```

248         break;
249     case PSB_6_ADDRESS:
250         ToSend(0,0);
251         break;
252     case PSB_7_ADDRESS:
253         ToSend(0,0);
254         break;
255     case PSB_8_ADDRESS:
256         ToSend(0,0);
257         break;
258     }
259     break;
260 }
261 }
262 }
263
264 void retrySystem(void)
265 {
266     //If you havent received a response
267     if(!receivedResponse )
268     {
269         //And it has been some time
270         if(timerDone(&resendTimer))
271         {
272             //printf("missed: %d\r\n",addressList[slaveAddress]);
273             //while(readByteUART2()!=0);
274             //and you havent retried too many times
275             if(numberOfRetries < MAX_RETRY_PER_ADDRESS)
276             {
277                 //Send again
278                 populateSlaveDataFT(addressList[slaveAddress]);
279                 sendData(addressList[slaveAddress]);
280                 numberOfRetries++;
281             }
282             else    //You tried too much
283             {
284                 numberOfRetries=0;
285                 incrementSlaveCounter();
286                 //Allow the send routine to send data
287                 receivedResponse = true;
288             }
289         }
290     }
291 }
292
293 bool testBoardComms(unsigned char slaveNum)
294 {
295     static bool testReceivedData = true, firstpass=true;
296     static timers_t timeout;
297
298     //First time through, init timer
299     if(firstpass)
300     {
301         firstpass=false;
302         setTimerInterval(&timeout,150);
303     }
304
305     //If you receive data
306     if(receiveData())
307     {
308         writeDigitSevenSeg(15,true);
309         switch(slaveNum)
310         {
311             case MDB_L_ADDRESS:
312                 printf("MDBL\r\n");

```

```

313         break;
314     case MDB_R_ADDRESS:
315         printf("MDBR\r\n");
316         break;
317     case WMCB_ADDRESS:
318         printf("WMCB\r\n");
319         break;
320     case PSB_1_ADDRESS:
321         setSensorDistance(0,receiveArray[1]);
322         break;
323     case PSB_2_ADDRESS:
324         setSensorDistance(1,receiveArray[1]);
325         break;
326     case PSB_3_ADDRESS:
327         setSensorDistance(2,receiveArray[1]);
328         break;
329     case PSB_4_ADDRESS:
330         setSensorDistance(3,receiveArray[1]);
331         break;
332     case PSB_5_ADDRESS:
333         setSensorDistance(4,receiveArray[1]);
334         break;
335     case PSB_6_ADDRESS:
336         setSensorDistance(5,receiveArray[1]);
337         break;
338     case PSB_7_ADDRESS:
339         setSensorDistance(6,receiveArray[1]);
340         break;
341     case PSB_8_ADDRESS:
342         setSensorDistance(7,receiveArray[1]);
343         break;
344     }
345
346     if(slaveNum>=PSB_1_ADDRESS && slaveNum <= PSB_8_ADDRESS)
347     {
348         //printf("%d\r\n",receiveArray[0]);
349         //printf("Sensor # %d: ",receiveArray[0]);
350         //int i;
351         //for(i=0;i<8;i++)
352         //{
353             //    printf("%3.3f",getSensorDistance(i));
354             //    printf(" ");
355         //}
356         //    printf("\r\r\n");
357     }
358
359     testReceivedData=true;
360     resetTimer(&timeout);
361     return true;
362 }
363
364
365     if(testReceivedData)
366     {
367         switch(slaveNum)
368         {
369             case MDB_L_ADDRESS:
370                 //testLeftMotorControl();
371                 ToSend(SPEED_SETTING_DATA_ADDRESS,getLeftMotorSpeed());
372                 //printf("Sending left motor command: %d to MDB\r\n",
373                 getLeftMotorSpeed());
374                 break;
375             case MDB_R_ADDRESS:
376                 //testRightMotorControl();
377                 ToSend(SPEED_SETTING_DATA_ADDRESS,getRightMotorSpeed());

```

```

377         //printf("Sending right motor command: %d to MDB\r\n",
378         getRightMotorSpeed());
379         break;
380     case WMCB_ADDRESS:
381         testWeaponMotorControl();
382         ToSend(1,getWeaponMotorSpeed());
383         printf("Sending weapon motor command: %d to
384             WMCB\r\n",getWeaponMotorSpeed());
385         break;
386     case PSB_1_ADDRESS:
387     case PSB_2_ADDRESS:
388     case PSB_3_ADDRESS:
389     case PSB_4_ADDRESS:
390     case PSB_5_ADDRESS:
391     case PSB_6_ADDRESS:
392     case PSB_7_ADDRESS:
393     case PSB_8_ADDRESS:
394         ToSend(0,1);
395         //printf("Sending PSB %d data\r\n", slaveNum);
396         break;
397     }
398     sendData(slaveNum);
399     testReceivedData=false;
400     resetTimer(&timeout);
401     return false;
402 }
403 if(!testReceivedData && timerDone(&timeout))
404 {
405     printf("%d\r\n",slaveNum);
406     testReceivedData=true;
407     return true;
408 }
409
410 void incrementSlaveCounter(void)
411 {
412     //Move on
413     slaveAddress++;
414
415     //IF WE SKIP WMCB
416     #ifdef DISABLE_MOTORS
417         if(slaveAddress==MDB_L_ADDRESS)
418         {
419             slaveAddress+=2;
420         }
421
422     #endif
423     #if defined(DISABLE_WMCB)
424         if(slaveAddress==WMCB_ADDRESS_INDEX)
425         {
426             slaveAddress++;
427         }
428     #endif
429
430
431     //Rollover conditions
432     #if defined( DISABLE_SENSORS) && defined(DISABLE_WMCB)
433         //If sensors and weapon disabled, talk to just the MDBs
434         if(slaveAddress>MDB_R_ADDRESS_INDEX)
435         {
436     #elif defined (DISABLE_SENSORS)
437         //If we are talking to slaves, increment until we reach the
438         last slave
439         if(slaveAddress>(WMCB_ADDRESS_INDEX))

```

```

439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
    {
    #else
        if(slaveAddress>(2+NUMBER_OF_SENSORS))
        {
        #endif
        //Restart conditions
        #if defined(IGNORE_MDB_RESPONSE) && defined(DISABLE_WMCB)
            slaveAddress=PSB_1_ADDRESS_INDEX;//3 is currently using only
            the first sensor board ()
        }
        #elif defined(IGNORE_MDB_RESPONSE)
            slaveAddress=WMCB_ADDRESS_INDEX;//3 is currently using only
            the first sensor board ()
        }
        #else
            slaveAddress=0;//3 is currently using only the first sensor
            board ()
        }
    #endif
}
*****
End of File
*/

```

```

1  /* **** Demonstration Macro Systems **** */
2  /** Demonstration Macro Systems
3
4  @Company
5      Zac Kilburn
6
7  @File Name
8      demonstrationSetup.c
9
10 @Summary
11     Brief description of the file.
12
13 @Description
14     Turning and wall approach system for demonstration purposes.
15 */
16 /* **** Demonstration Macro Systems **** */
17
18 #include "demonstrationSetup.h"
19 #include "helperFunctions.h"
20 #include "IMU.h"
21 #include "communications.h"
22 #include "PID.h"
23 #include "motorControl.h"
24 #include "FastTransfer.h"
25 #include "driveAssistance.h"
26 #include "macros.h"
27 #include "RCInputs.h"
28 #include "LIDARSensors.h"
29 #include "MillisTimer.h"
30
31 // #define IGNORE_MACRO_KILL
32
33 //-----WALL APPROACH-----
34 #define MIN_SPEED_APPROACH -17
35 #define MAX_SPEED_APPROACH 17
36
37 #define KP_APPROACH .5
38 #define KI_APPROACH 0
39 #define KD_APPROACH 0
40 //-----WALL AVOIDANCE-----
41 #define MIN_SPEED_AVOID -20
42 #define MAX_SPEED_AVOID 20
43
44 #define WALL_AVOID_SPEED          10
45 #define MINIMUM_WALL_DISTANCE_DRIVE 25
46 #define FORWARD_WALL_APPROACH_DIST 100
47
48 #define FORWARD_FACING_SENSOR_NUMBER 0
49 #define LEFT_SIDE_SENSOR_NUMBER    1
50 #define RIGHT_SIDE_SENSOR_NUMBER   2
51 #define ADJUST_RIGHT              5
52 #define ADJUST_LEFT                -5
53 #define DRIVE_STRAIGHT            0
54
55 //-----PERPENDICULAR-----
56 #define ANGLE_TO_TURN             3
57 #define NOISE_MARGIN               0.5
58
59 //-----
60
61 #define DEBUG_PID_UART
62
63 timers_t debugTHIS;
64 timers_t decisionTiming, sampleTiming;
65

```

```

66     int boundMotorCommandsApproach(int val);
67     int boundMotorCommandsWallAvoid(int val);
68
69 void findPerpendicular(uint8_t sensorToUse)
70 {
71     //Prep data values
72     static timers_t updateTimer;
73     double previousDistance = getSensorDistance(sensorToUse);
74     uint8_t foundMultipleTimes=0;
75     //double angleOfCenter,angleCapture;
76     //bool senseDistances=true;
77     bool foundOneWay = false;
78     bool localMinFound =false;
79
80     //Prepare a timer for making control decisions and sampling
81     setTimerInterval(&decisionTiming,50);
82     setTimerInterval(&updateTimer,50);
83     setTimerInterval(&debugTHIS,100);
84
85     //Reset the angle tracking for IMU correction
86     resetXAngle();
87     setGyroRelativeAngle();
88
89     //Do method until local minimum found or told to stop
90     while(!localMinFound && (getMacroCommand() !=0))
91     {
92         //Update the gyroscope values
93         updateIMU();
94
95         //Update the communications systems (Motors/Sensors)
96         updateCommunications();
97
98 #ifndef IGNORE_MACRO_KILL
99         //Update the RC input to cancel
100        updateRCInputs();
101
102    #endif
103    //Do sample processing and input to PID system
104    if(timerDone(&updateTimer))
105    {
106        if(!foundOneWay)
107        {
108            setLeftMotorSpeed(6);
109            setRightMotorSpeed(-6);
110        }
111        else
112        {
113
114            setLeftMotorSpeed(-6);
115            setRightMotorSpeed(6);
116        }
117    //    if(senseDistances)
118    //    {
119        //If the distance has gotten bigger
120        if(previousDistance- getSensorDistance(sensorToUse)+NOISE_MARGIN
121        < 0)
122        {
123            //Distance is getting bigger
124            if(foundOneWay)
125            {
126                if(foundMultipleTimes>4)
127                    localMinFound = true;
128            }
129            //angleOfCenter=(getXAxisAngle()+angleCapture)/2;

```

```

130                     foundMultipleTimes++;
131                     foundOneWay=false;
132                     previousDistance = getSensorDistance(sensorToUse);
133                 }
134             }
135         }
136     }
137     //angleCapture=getXAxisAngle();
138     previousDistance = getSensorDistance(sensorToUse);
139     foundOneWay = true;
140   }
141 }
142 else
143 {
144     //continue turning
145 }
146 if(previousDistance>getSensorDistance(sensorToUse))
147 {
148     previousDistance = getSensorDistance(sensorToUse);
149 }
150 //senseDistances = false;
151 resetTimer(&updateTimer);
152 //
153 //
154 //
155 //        if(!foundOneWay)
156 //            doTurn(ANGLE_TO_TURN);
157 //        else
158 //            doTurn(-ANGLE_TO_TURN);
159 //        senseDistances = true;
160 //        resetTimer(&updateTimer);
161 //
162 }
163
164 #ifdef DEBUG_PID_UART
165     if(timerDone(&debugTHIS))
166     {
167         printf("Current: %f ,Prev: %f\r\n",
168               getSensorDistance(sensorToUse),
169               previousDistance);
170         printf("Macro: %d \r\n", getMacroCommand());
171     }
172 #endif
173 }
174
175
176
177 //printf("Perp Done\r\n");
178
179 completeMacro();
180 //doTurn(getXAxisAngle()-angleOfCenter);
181
182 }
183
184 void driveStraight(void)
185 {
186     //Prep data values
187
188     //The critical piece of this algorithm, using a nudge back to center, this
189     //value will shove the robot, IMU will handle variations that this causes.
190     int driveAdjust=0;
191
192     //Prepare a timer for making control decisions and sampling
193     setTimerInterval(&decisionTiming,50);
194     setTimerInterval(&sampleTiming,5);

```

```

195     setTimerInterval(&debugTHIS,100);
196
197     //Setup the PID system with tuning and a target
198     //resetPIDController(targetDistance, KP_APPROACH, KI_APPROACH, KD_APPROACH);
199
200     //Reset the angle tracking for IMU correction
201     resetXAngle();
202     setGyroRelativeAngle();
203
204     //Do method until distance is close to the wall/NON-ZERO or told to stop
205
206     while(((getSensorDistance(FORWARD_FACING_SENSOR_NUMBER)>FORWARD_WALL_APPROACH_
207     DIST)
208             ||(getSensorDistance(FORWARD_FACING_SENSOR_NUMBER)==0))
209             && (getMacroCommand() !=0))
210     {
211         //Update the communications systems (Motors/Sensors)
212         updateCommunications();
213
214         //Update the RC input to cancel
215         updateRCInputs();
216
217         //Do sample processing and input to PID system
218         if(timerDone(&sampleTiming))
219         {
220             if(getSensorDistance(LEFT_SIDE_SENSOR_NUMBER) <
221             MINIMUM_WALL_DISTANCE_DRIVE)
222             {
223                 driveAdjust = ADJUST_RIGHT;
224             }
225             else if(getSensorDistance(RIGHT_SIDE_SENSOR_NUMBER) <
226             MINIMUM_WALL_DISTANCE_DRIVE)
227             {
228                 driveAdjust = ADJUST_LEFT;
229             }
230         }
231
232         //Allow the PID to output a control value
233         if(timerDone(&decisionTiming))
234         {
235             //Driving straight is the goal
236             int _lm = (int) WALL_AVOID_SPEED;
237             int _rm = (int) WALL_AVOID_SPEED;
238
239             //Correct the motor values using the IMU
240             motorGyroFeedbackStraight(&_lm, &_rm);
241
242             _lm += driveAdjust;
243             _rm -= driveAdjust;
244
245             //Send to the motor system to be sent by the communications library
246
247             setLeftMotorSpeed(boundMotorCommands(boundMotorCommandsWallAvoid(_lm)))
248             );
249             setRightMotorSpeed(boundMotorCommands(boundMotorCommandsWallAvoid(_rm))
250             );
251         }
252     #ifdef DEBUG_PID_UART
253         if(timerDone(&debugTHIS))

```

```

252     {
253         printf("SForward: %f ,SL: %f ,SR: %f , lm: %d, rm: %d\n",
254             getSensorDistance(FORWARD_FACING_SENSOR_NUMBER),
255             getSensorDistance(LEFT_SIDE_SENSOR_NUMBER),
256             getSensorDistance(RIGHT_SIDE_SENSOR_NUMBER),
257             getLeftMotorSpeed(),
258             getRightMotorSpeed());
259
260         //printf("E: %3f, P: %3f, I: %3f, O: %3f\n",
261         //getErrorComponent(),getProportionalComponent(),
262         //getIntegralComponent(), getDerivativeComponent(), getPIDOutput());
263     }
264
265     //CLEANUP WHEN WE ARE DONE
266     setLeftMotorSpeed(0);
267     setRightMotorSpeed(0);
268
269     ToSend(SPEED_SETTING_DATA_ADDRESS,0);
270     sendData(MDB_L_ADDRESS);
271     ToSend(SPEED_SETTING_DATA_ADDRESS,0);
272     sendData(MDB_R_ADDRESS);
273
274 }
275
276
277 void doApproach(double targetDistance)
278 {
279     //Prep data values
280
281     //Prepare a timer for making control decisions and sampling
282     setTimerInterval(&decisionTiming,20);
283     setTimerInterval(&sampleTiming,2);
284     setTimerInterval(&debugTHIS,100);
285
286     //Setup the PID system with tuning and a target
287     resetPIDController(targetDistance, KP_APPROACH, KI_APPROACH, KD_APPROACH);
288
289     //Reset the angle tracking for IMU correction
290     resetXAngle();
291     setGyroRelativeAngle();
292     int _rm;
293     int _lm;
294
295     //Do method until turn is complete or told to stop
296     while(!isAboutDouble(getSensorDistance(0),targetDistance,5) &&
297           (getMacroCommand() !=0))
298     {
299         //Update the gyroscope values
300         updateIMU();
301
302         //Update the communications systems (Motors/Sensors)
303         updateCommunications();
304
305         //Update the RC input to cancel
306         updateRCInputs();
307
308         //Do sample processing and input to PID system
309         if(timerDone(&sampleTiming))
310         {
311             updatePIDOutput(getSensorDistance(0));
312         }
313
314         //Allow the PID to output a control value

```

```

314     if(timerDone(&decisionTiming))
315     {
316         //Condition the output from the PID so its appropriate to drive the
317         //motors
318         double outputToMotor = getPIDOutput();
319         _lm = (int) -outputToMotor;
320         _rm = (int) -outputToMotor;
321
322         //Correct the left and right motor for the gyro error based on
323         //percentage throttle
324         _lm=boundMotorCommandsApproach(_lm);
325         _rm=boundMotorCommandsApproach(_rm);
326
327         //Correct the motor values using the IMU
328         motorGyroFeedbackStraight(&_lm, &_rm);
329
330         //Send to the motor system to be sent by the communications library
331         setLeftMotorSpeed(boundMotorCommandsApproach(_lm));
332         setRightMotorSpeed(boundMotorCommandsApproach(_rm));
333     }
334
335     #ifdef DEBUG_PID_UART
336     if(timerDone(&debugTHIS))
337     {
338         printf("Sensor: %f , lm: %d, rm:
339             %d\r\n",getSensorDistance(0),getLeftMotorSpeed(),getRightMotorSpeed());
340         ;
341         printf("Angle: %f, lraw: %d, rraw: %d\r\n",getXAxisAngle(),_lm,_rm);
342         //printf("E: %3f, P: %3f, I: %3f, D: %3f, O: %3f\r\n",
343         //getErrorComponent(),getProportionalComponent(),
344         //getIntegralComponent(), getDerivativeComponent(), getPIDOutput());
345     }
346
347     completeMacro();
348
349     int boundMotorCommandsWallAvoid(int val)
350     {
351         if(val>MAX_SPEED_AVOID)
352         {
353             val=MAX_SPEED_AVOID;
354         }
355         else if(val<MIN_SPEED_AVOID)
356         {
357             val=MIN_SPEED_AVOID;
358         }
359         return val;
360     }
361
362     int boundMotorCommandsApproach(int val)
363     {
364         if(val>MAX_SPEED_APPROACH)
365         {
366             val=MAX_SPEED_APPROACH;
367         }
368         else if(val<MIN_SPEED_APPROACH)
369         {
370             val=MIN_SPEED_APPROACH;
371         }
372         return val;
373     }

```

```

1  #include "detectEnemy.h"
2  #include "locationFunction/solveLocation.h"
3  #include "LIDARSensors.h"
4  #include <stdbool.h>
5  #include <stdio.h>
6
7  #define ENEMY_DETECTION_LENGTH 50
8  int samples[ENEMY_DETECTION_LENGTH];
9  int counterIndex=0;
10 int sensorDetected[9]={0,0,0,0,0,0,0,0,0};
11
12 double x,y;
13
14
15 void initEnemySolver(void)
16 {
17     /* Initialize the application.
18     You do not need to do this more than one time. */
19     enemyDetection_initialize();
20 }
21
22 double getXPos(void)
23 {
24     return x;
25 }
26
27 double getYPos(void)
28 {
29     return y;
30 }
31
32 int detectEnemy(void)
33 {
34     double locationSet[3];
35     double enemyDetectionInputs[10];
36
37     //Solve Location of Robot
38     int i=0;
39     solveLocation(getSensorDistancesArrayCenterOfRobot(), (double*)locationSet);
40
41     //Construct enemy detection
42     for(i=0;i<8;i++)
43     {
44         enemyDetectionInputs[i]=getSensorDistanceCenterOfRobot(i);
45     }
46     x=locationSet[1] * (cos(locationSet[0]*(3.14159/180)));
47     y=locationSet[1] * (sin(locationSet[0]*(3.14159/180)));
48
49
50 //    printf("X: %f, Y: %f\r\n", x,y);
51 //    printf("Heading: %f\r\n",locationSet[2]);
52     enemyDetectionInputs[8]=x;
53     enemyDetectionInputs[9]=y;
54     //Store this sample of enemy location
55     samples[counterIndex] = locateEnemy((double*)enemyDetectionInputs)-1;
56
57     //Move pointer to next location
58     counterIndex++;
59     if(counterIndex>=ENEMY_DETECTION_LENGTH)
60     {
61         counterIndex=0;
62     }
63
64     //Wipe the sum of the sensor counters
65     for(i=0;i<9;i++)

```

```

66     {
67         sensorDetected[i]=0;
68     }
69
70     //Sum up the samples per sensor
71     for(i=0;i<ENEMY_DETECTION_LENGTH;i++)
72     {
73         sensorDetected[samples[i]+1]++;
74     }
75     int maxConfidence=0;
76     int locationEnemy=0;
77     //Find the most confident sensor that enemy is present on
78     for(i=0;i<9;i++)
79     {
80         if(sensorDetected[i] > maxConfidence)
81         {
82             maxConfidence = sensorDetected[i];
83             locationEnemy = i;
84         }
85     }
86     return locationEnemy-1;
87 }
88
89 int locateEnemy(double * inputs)
90 {
91     int i=0;
92     double maxConf = 50;
93     int locationEnemy=0;
94     double enemyHere[9];
95     //printf("B4: %d", millis());
96     enemyDetection(inputs,(double*)enemyHere);
97     for(i=0;i<9;i++)
98     {
99         if(enemyHere[i] > maxConf)
100         {
101             maxConf = enemyHere[i];
102             locationEnemy=i;
103         }
104         //printf("index %d: %f\r\n",i,enemyHere[i]);
105     }
106     //printf("\r\n");
107     //printf("After: %d \n",millis());
108     return locationEnemy;
109 }
110
111 void closeEnemySolver(void)
112 {
113     /* Terminate the application.
114     You do not need to do this more than one time. */
115     enemyDetection_terminate();
116 }
```

```

1  /* **** Do Turn Macro System **** */
2  /** Do Turn Macro System
3
4  @Company
5      Zac Kilburn
6
7  @File Name
8      doTurn.c
9
10 @Summary
11     Brief description of the file.
12
13 @Description
14     Turning system for autonomous purposes.
15 */
16 #include "doTurn.h"
17 #include "PID.h"
18
19 #include "holdRotationWeaponStopped.h"
20 #include "motorControl.h"
21 #include "MillisTimer.h"
22 #include "driveAssistance.h"
23 #include "macros.h"
24 #include "helperFunctions.h"
25
26 //----- TURN -----
27 #define MIN_SPEED_TURN -30 //30
28 #define MAX_SPEED_TURN 30 //30
29
30 #define KP_TURN 5
31 #define KI_TURN 0.01//.1
32 #define KD_TURN .5
33
34 timers_t debugTHIS;
35 timers_t decisionTiming, sampleTiming;
36
37 int boundMotorCommandsTurn(int val);
38
39
40 void doTurn(double angle)
41 {
42     //Prep data values
43
44     //Prepare a timer for making control decisions and sampling
45     setTimerInterval(&decisionTiming,50);
46     setTimerInterval(&sampleTiming,5);
47     setTimerInterval(&debugTHIS,100);
48     resetXAngle();      //Relative turning (zero before we turn)
49
50     resetPIDController(angle, KP_TURN, KI_TURN, KD_TURN);
51
52     printf("Turn\r\n");
53     //Do method until turn is complete or told to stop
54     while(!isAtAngleStable(angle,getXAxisAngle(),2) && (getMacroCommand() !=0))
55     {
56         //Update the gyroscope values
57         updateIMU();
58
59         //Update the communications systems (Motors/Sensors)
60         updateCommunications();
61
62 #ifndef IGNORE_MACRO_KILL
63         //Update the RC input to cancel
64         updateRCInputs();
65     #endif

```

```

66     //Do sample processing and input to PID system
67     if(timerDone(&sampleTiming))
68     {
69         updatePIDOutput(getXAxisAngle());
70     }
71
72     //Allow the PID to output a control value
73     if(timerDone(&decisionTiming))
74     {
75         //Condition the output from the PID so its appropriate to drive the
76         //motors
77         float outputToMotor = getPIDOutput();
78         int _lm = -outputToMotor;
79         int _rm = outputToMotor;
80
81         //Send to the motor system to be sent by the communications library
82         setLeftMotorSpeed(boundMotorCommands(boundMotorCommandsTurn(_lm)));
83         setRightMotorSpeed(boundMotorCommands(boundMotorCommandsTurn(_rm)));
84     }
85
86 #ifdef DEBUG_PID_UART
87     if(timerDone(&debugTHIS))
88     {
89         //printf("Chan 1: %d\r\n",getChannel1());
90         //printf("angle: %f, lm: %d, rm:
91         // %d\r\n",getXAxisAngle(),getLeftMotorSpeed(),getRightMotorSpeed());
92         //printf("MacroCommand %d\r\n", getMacroCommand());
93         //printf("E: %3f, P: %3f, I: %3f, D: %3f, O: %3f\r\n",
94         //getErrorComponent(),getProportionalComponent(),
95         //getIntegralComponent(), getDerivativeComponent(), getPIDOutput());
96     }
97 #endif
98
99 }
100
101 bool isAtAngleStable(double target, double currentAngle, double
102 acceptableAngleRange)
103 {
104     static timers_t stableEndTimer;
105     static bool setup = false;
106     if(!setup)
107     {
108         setTimerInterval(&stableEndTimer,100);
109         setup=true;
110     }
111
112     if(isAboutDouble(target,currentAngle,acceptableAngleRange) &&
113     timerDone(&stableEndTimer))
114     {
115         return true;
116     }
117     else if(!isAboutDouble(target,currentAngle,acceptableAngleRange))
118     {
119         resetTimer(&stableEndTimer);
120         return false;
121     }
122     else
123     {
124         return false;
125     }
126 }
```

```
125 }
126
127 int boundMotorCommandsTurn(int val)
128 {
129     if(val>MAX_SPEED_TURN)
130     {
131         val=MAX_SPEED_TURN;
132     }
133     else if(val<MIN_SPEED_TURN)
134     {
135         val=MIN_SPEED_TURN;
136     }
137     return val;
138 }
```

```

1  /* **** */
2  /** Drive Assistance Systems
3
4  @Company
5      Zac Kilburn
6
7  @File Name
8      driveAssistance.c
9
10 @Summary
11     An attempt to provide gyro control feedback to manual control.
12
13 @Description
14     The functions will modify the output to the motors based on
15     gyroscope feedback and ensure straight driving when the robot
16     is not being told to turn.
17
18     May also try to have the robot turn with an angular rate
19
20 /* **** */
21
22 #include "IMU.h"
23
24 #define GYRO_COMPENSATION          0.5
25 #define GYRO_COMPENSATION_AUTO    0.5
26 #define GYRO_COMPENSATION_ZERO_INPUT 1
27 #define TURNING_RATE_MAX      5000
28
29 bool stabilizationActive=true;
30 float gyroRelativeAngle=0;
31
32 bool getGyroStabilizationActive()
33 {
34     return stabilizationActive;
35 }
36
37 void setGyroStabilizationActive(bool b)
38 {
39     stabilizationActive=b;
40 }
41
42 void setGyroRelativeAngle(void)
43 {
44     gyroRelativeAngle = getXAxisAngle();
45 }
46
47 void setGyroRelativeAngleValue(float val)
48 {
49     gyroRelativeAngle = val;
50 }
51
52 float getGyroRelativeAngleValue(void)
53 {
54     return gyroRelativeAngle;
55 }
56
57 void motorGyroFeedbackStraightAuto(int *lm, int *rm)
58 {
59     if(getGyroStabilizationActive())
60     {
61         //Calculate the error of the angle versus the relative 0 angle
62         float angleError =  getXAxisAngle()-gyroRelativeAngle ;
63
64         //Calculate total throttle input
65         int totalMag = (*lm+*rm) /2;

```

```

66         //Convert to a percentage
67         float percentageThrottle = totalMag/255.0;
68     //     if(!isInverted())
69     //
70         //Correct the left and right motor for the gyro error based on
71         //percentage throttle
72         *lm+=(angleError*GYRO_COMPENSATION_AUTO *(1.0-percentageThrottle));
73         *rm-=(angleError*GYRO_COMPENSATION_AUTO*(1.0-percentageThrottle));
74     //
75     //
76     //          //Correct the left and right motor for the gyro error based on
77         //percentage throttle
78         *lm-=(angleError*GYRO_COMPENSATION *(1.0-percentageThrottle));
79         *rm+=(angleError*GYRO_COMPENSATION*(1.0-percentageThrottle));
80     }
81 }
82
83
84
85 void motorGyroFeedbackStraight(int *lm, int *rm)
86 {
87     if(getGyroStabilizationActive())
88     {
89         //Calculate the error of the angle versus the relative 0 angle
90         float angleError = getXAxisAngle()-gyroRelativeAngle;
91         //Calculate total throttle input
92         int totalMag = (*lm+*rm) /2;
93         //Convert to a percentage
94         float percentageThrottle = totalMag/255.0;
95
96         //This attempts to only add a value to a motor, so that moving forward
97         //is the priority, while also correcting the angle
98
99         if(abs(angleError)>5)
100     {
101         //if the command is positive
102         if(*lm>0 && *rm>0)
103     {
104         //if the error is positive
105         if(angleError<0)
106         {
107             *rm-=(int)(angleError*GYRO_COMPENSATION
108                         *(1.0-percentageThrottle));
109         }
110         else
111         {
112             *lm+=(int)(angleError*GYRO_COMPENSATION*(1.0-percentageThrott
113                         le));
114         }
115         //if the command is negative
116         else if(*lm<0 && *rm<0)
117     {
118         //if the error is positive
119         if(angleError>0)
120         {
121             *rm-=(int)(angleError*GYRO_COMPENSATION
122                         *(1.0-percentageThrottle));
123         }
124         else
125         {

```

```

                *lm+=(int) (angleError*GYRO_COMPENSATION*(1.0-percentageThrottle));
125            }
126        }
127        //if the inputs are zero - allows for the robot to stabilize when
128        //inputs are zero
129        else
130        {
131            //Correct the left and right motor for the gyro error based on
132            //percentage throttle
133            //if(abs(angleError)>2)
134            //{
135                //Correct the left and right motor for the gyro error
136                //based on percentage throttle
137                *lm+=(int) (angleError*(GYRO_COMPENSATION_ZERO_INPUT));
138                *rm+=(int) (angleError*(GYRO_COMPENSATION_ZERO_INPUT));
139            }
140            //if(originalL>0)
141            //{
142                //if(*lm<5)
143                //{
144                    *lm=5;
145                }
146                //else
147                //{
148                    if(*lm>-5)
149                    {
150                        *lm=5;
151                    }
152                }
153                //if(originalR>0)
154                //{
155                    if(*rm<5)
156                    {
157                        *rm=5;
158                    }
159                }
160                //else
161                //{
162                    if(*rm>-5)
163                    {
164                        *rm=5;
165                    }
166                }
167            }
168        }
169    }
170    void motorGyroFeedbackTurning(int lm, int rm)
171    {
172        int16_t turningRate = getXDPS();
173        if(abs(turningRate > TURNING_RATE_MAX))
174        {
175            if(turningRate > 0)           //POSITIVE TURNING RATE
176            {
177                if(lm>rm)           //INSTRUCTIONS ARE CW
178                {
179
180                }
181                else                  //INSTRUCTIONS ARE CCW
182                {
183
184                }

```

```
185     }
186     else //NEGATIVE TURNING RATE
187     {
188         if(lm>rm) //INSTRUCTIONS ARE CW
189         {
190             }
191         else //INSTRUCTIONS ARE CCW
192         {
193             }
194         }
195     }
196 }
197 }
198 */
199
200 /* **** End of File ****
201 */
202 */
203
```

```

1 #include "helperFunctions.h"
2
3 bool isAboutInt(int value, int referenceValue, int range)
4 {
5     return ((value+range)>referenceValue) && ((value-range)<referenceValue);
6 }
7
8 bool isAboutFloat(float value, float referenceValue, float range)
9 {
10    return ((value+range)>referenceValue) && ((value-range)<referenceValue);
11 }
12 bool isAboutDouble(double value, double referenceValue, double range)
13 {
14    return ((value+range)>referenceValue) && ((value-range)<referenceValue);
15 }
16
17
18 bool isWithinInt(int value, int lowValue, int highValue)
19 {
20    return ((value>lowValue) && (value<highValue));
21 }
22
23
24 void I2CsweepAddresses(void)
25 {
26     int state=0,addressToSend=0;
27     bool addressesIncomplete=true;
28
29     while(addressesIncomplete)
30     {
31         //Wait for idle bus
32         while (! PLIB_I2C_BusIsIdle ( I2C_ID_1 ) );
33         switch(state)
34         {
35             case 0:
36                 PLIB_I2C_MasterStart(I2C_ID_1);
37                 state++;
38                 break;
39             case 1:
40                 PLIB_I2C_TransmitterByteSend(I2C_ID_1,addressToSend);
41                 while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
42                 state++;
43                 break;
44             case 2:
45                 PLIB_I2C_MasterStop(I2C_ID_1);
46                 addressToSend++;
47                 if(addressToSend>=255)
48                 {
49                     addressesIncomplete=false;
50                 }
51                 state=0;
52                 break;
53         }
54     }
55 }
56

```

```

1  #include "holdRotationWeaponStopped.h"
2  #include "helperFunctions.h"
3  #include "IMU.h"
4  #include "communications.h"
5  #include "PID.h"
6  #include "motorControl.h"
7  #include "FastTransfer.h"
8  #include "driveAssistance.h"
9  #include "macros.h"
10 #include "RCInputs.h"
11 #include "LIDARSensors.h"
12 #include "MillisTimer.h"
13
14 timers_t sampleTimingHold, decisionTimingHold;
15
16 #define MAX_SPEED_HOLD 30
17 #define MIN_SPEED_HOLD -30
18
19 int boundMotorCommandsHold(int val);
20
21 void setupHoldPosition(void)
22 {
23     setTimerInterval(&sampleTimingHold,5);
24     setTimerInterval(&decisionTimingHold,50);
25 }
26
27 void holdPosition(void)
28 {
29     //Do sample processing and input to PID system
30     if(timerDone(&sampleTimingHold))
31     {
32         updatePIDOutput(getXAxisAngle());
33     }
34
35     //Allow the PID to output a control value
36     if(timerDone(&decisionTimingHold))
37     {
38         //Condition the output from the PID so its appropriate to drive the
39         //motors
40         float outputToMotor = getPIDOutput();
41         int _lm = -outputToMotor;
42         int _rm = outputToMotor;
43
44         //Send to the motor system to be sent by the communications library
45         setLeftMotorSpeed(boundMotorCommandsHold(_lm));
46         setRightMotorSpeed(boundMotorCommandsHold(_rm));
47     }
48 }
49
50
51 int boundMotorCommandsHold(int val)
52 {
53     if(val>MAX_SPEED_HOLD)
54     {
55         val=MAX_SPEED_HOLD;
56     }
57     else if(val<MIN_SPEED_HOLD)
58     {
59         val=MIN_SPEED_HOLD;
60     }
61     return val;
62 }

```

```

1  /* **** Descriptive File Name **** */
2  /** Descriptive File Name
3
4      @Company
5          Company Name
6
7      @File Name
8          filename.c
9
10     @Summary
11         Brief description of the file.
12
13     @Description
14         Describe the purpose of this file.
15 */
16 /* **** **** **** */
17 #include "I2CFunctions.h"
18 #include "../../../../../framework/driver/i2c/drv_i2c.h"
19 #include "MillisTimer.h"
20 #include "MicrosTimer.h"
21
22 timers_micros_t timeout;
23 bool mutexControl = false;
24
25 void setupI2CFunctions(void)
26 {
27     setTimerIntervalMicros(&timeout, 100);
28 }
29
30 bool WaitReceiverByteAck(void)
31 {
32     resetTimerMicros(&timeout);
33     while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1))
34     {
35         if(timerDoneMicros(&timeout))
36         {
37             return false;
38         }
39     }
40     return true;
41 }
42
43 bool WaitRxByteAvailable(void)
44 {
45     resetTimerMicros(&timeout);
46     while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1))
47     {
48         if(timerDoneMicros(&timeout))
49         {
50             return false;
51         }
52     }
53     return true;
54 }
55
56 bool WaitTransmitterWriteCompleted(void)
57 {
58     resetTimerMicros(&timeout);
59     while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1))
60     {
61         if(timerDoneMicros(&timeout))
62         {
63             return false;
64         }
65     }

```

```

66     return true;
67 }
68
69 //According to errata, there is a silicon issue
70 void resetI2CSystem(void)
71 {
72     PLIB_I2C_Disable(I2C_ID_1);
73
74     //Wait 4 instruction clock cycles by clearing the lines
75     //CLEAR I2C handstate by toggling lines
76     TRISDbits.TRISD10 =0;    //SCL
77     TRISDbits.TRISD9  =0;   //SDA
78
79     LATDbits.LATD9   =0;    //SDA
80     LATDbits.LATD10 ^=1;   //SCL
81     LATDbits.LATD10 ^=1;   //SCL
82     LATDbits.LATD10 ^=1;   //SCL
83
84     //    TRISDbits.TRISD9  =1;
85     //    TRISDbits.TRISD10 =1;
86
87
88     PLIB_I2C_Enable(I2C_ID_1);
89 }
//=====
90 =====
91 void I2cStart(void)
92 {
93     PLIB_I2C_MasterStart(I2C_ID_1);
94 }
//=====
95 =====
96 void I2cStop (void)
97 {
98     PLIB_I2C_MasterStop(I2C_ID_1);
99 }
100
101 bool getMutex(void)
102 {
103 //    if(mutexControl)
104 //    {
105 //        if(timerDone(&mutexTimer)) //Mandatory wait after mutex is used
106 //        {
107 //            return mutexControl;
108 //        }
109 //    }
110 //    else
111 //    {
112 //        return false;
113 //    }
114 }
115
116 void setMutex(bool mu)
117 {
118 //    if(mu==false)
119 //    {
120 //        resetTimer(&mutexTimer);
121 //    }
122 //    mutexControl=mu;
123 }
/* **** End of File ****
124 */
125 */
126 */
127

```

```

1  #include "IMU.h"
2  #include "MillisTimer.h"
3  #include "sevenSegDigit.h"
4  #include <stdio.h>
5  #include "I2CFunctions.h"
6  #include "../../../../../framework/driver/i2c/drv_i2c.h"
7  #include "helperFunctions.h"
8
9  #define NUMBER_SAMPLES_CALIBRATE      100.0
10 #define CORRECTION_FACTOR_GYRO       1.3//1.0715
11 #define SENSATIVITY_GYRO             1000
12 #define SCALING_GYRO                (SENSATIVITY_GYRO/65535.0)*CORRECTION_FACTOR_GYRO
13
14 #define CORRECTION_FACTOR_XL         1
15 #define SENSATIVITY_XL              32
16 #define SCALING_ACCEL               (SENSATIVITY_XL/65535.0)*CORRECTION_FACTOR_XL
17
18
19 //Variables to hold the values read from the gyro
20 join_t combineX_gyro, combineY_gyro, combineZ_gyro;
21
22 //Variables to hold the values read from the gyro
23 join_t combineX_accel, combineY_accel, combineZ_accel;
24
25 //Offset and deadzone variables gyro
26 int16_t offsetG_X=0, highG_x=-65535/2, lowG_x=65535/2;
27 int16_t offsetG_Y=0, highG_y=-65535/2, lowG_y=65535/2;
28 int16_t offsetG_Z=0, highG_z=-65535/2, lowG_z=65535/2;
29
30 //Offset and deadzone variables accel
31 int16_t offsetXL_X=0, highXL_x=-65535/2, lowXL_x=65535/2;
32 int16_t offsetXL_Y=0, highXL_y=-65535/2, lowXL_y=65535/2;
33 int16_t offsetXL_Z=0, highXL_z=-65535/2, lowXL_z=65535/2;
34
35 int16_t gyroXDPS, gyroYDPS, gyroZDPS;
36
37 unsigned long lastMillis=0;
38 timers_t IMU_UpdateTimer;
39 double xAngle,yAngle,zAngle;
40 double x_XL_raw, y_XL_raw, z_XL_raw;
41 double x_XL, y_XL, z_XL;
42
43 timers_t invertedTimer;
44 bool wasInverted=false;
45 bool isInverted(void)
46 {
47     if(x_XL>0.5 && !wasInverted)
48     {
49         resetTimer(&invertedTimer);
50         return true;
51     }
52     else if(!wasInverted)
53     {
54         if(timerDone(&invertedTimer))
55         {
56             printf("switch inverted");
57             wasInverted=true;
58             return false;
59         }
60         return true;
61     }
62     else if(x_XL<0.5 && wasInverted)
63     {
64         resetTimer(&invertedTimer);

```

```

65         return false;
66     }
67     else if(wasInverted)
68     {
69         if(timerDone(&invertedTimer))
70         {
71             printf("switch non-inverted");
72             wasInverted=false;
73             return true;
74         }
75         return false;
76     }
77     else
78     {
79         return wasInverted;
80     }
81 // 82 //  if(x_XL>0)
83 // 84 //      printf("inverted\r\n");
85 // 86 //      return false;
87 // 88 //      printf("non-inverted\r\n");
89 // 90 //      return true;
91 // 92 }
93
94 void updateGyro(void)
{
    static bool firstTime=true;
//First time through
95     if(firstTime)
96     {
97         firstTime=false;
98         lastMillis=millis();
99     }
//I2c Is available to use now
100    if(getMutex()==false)
101    {
102        if(timerDone(&IMU_UpdateTimer))
103        {
104
105            //printf("Gyro\r\n");
106            if(lastMillis!=millis())
107            {
108
109                //Read from the device
110                readIMU3AxisG();
111
112                //Process offsets
113                gyroXDPs = combineX_gyro.integer - offsetG_X;
114                gyroYDPs = combineY_gyro.integer - offsetG_Y;
115                gyroZDPs = combineZ_gyro.integer - offsetG_Z;
116
117                //Handle Deadzones
118                if(isWithinInt(gyroXDPs,lowG_x*1.2,highG_x*1.2))
119                    gyroXDPs=0;
120                if(isWithinInt(gyroYDPs,lowG_y,highG_y))           gyroYDPs=0;
121                if(isWithinInt(gyroZDPs,lowG_z,highG_z))
122                    gyroZDPs=0;
123
124                xAngle += (gyroXDPs*SCALING_GYRO *
125                            ((millis()-lastMillis)/1000.0));
126                yAngle += (gyroYDPs*SCALING_GYRO *

```

```

127         ((millis()-lastMillis)/1000.0));
128     zAngle += (gyroZDPS*SCALING_GYRO *
129     ((millis()-lastMillis)/1000.0));
130     lastMillis = millis();
131   }
132 }
133
134 void updateZAxis(void)
135 {
136   static bool firstTime=true;
137   //First time through
138   if(firstTime)
139   {
140     firstTime=false;
141     lastMillis=millis();
142   }
143
144   //I2c Is available to use now
145   if(getMutex()==false)
146   {
147     if(timerDone(&IMU_UpdateTimer))
148     {
149       if(lastMillis!=millis())
150       {
151         readIMU(Z_AXIS_GYRO);
152         gyroZDPS = combineZ_gyro.integer - offsetG_Z;
153         //printf("Offsetted value: %d\r\n",gyroRead);
154         if(isWithinInt(gyroZDPS,lowG_z,highG_z))
155           gyroZDPS=0;
156
157         zAngle += (gyroZDPS*SCALING_GYRO *
158         ((millis()-lastMillis)/1000.0));
159         lastMillis = millis();
160       }
161     }
162   }
163
164 void updateXL(void)
165 {
166
167 }
168
169 void updateIMU(void)
170 {
171   static bool firstTime=true;
172   //First time through
173   if(firstTime)
174   {
175     firstTime=false;
176     lastMillis=millis();
177   }
178   //I2c Is available to use now
179   if(getMutex()==false)
180   {
181     if(timerDone(&IMU_UpdateTimer))
182     {
183       if(lastMillis!=millis())
184       {
185         //Read from the device
186         readIMU6Axis();
187         //Process offsets

```

```

189     gyroXDPS = combineX_gyro.integer - offsetG_X;
190     gyroYDPS = combineY_gyro.integer - offsetG_Y;
191     gyroZDPS = combineZ_gyro.integer - offsetG_Z;
192
193     //Handle Deadzones
194     if(isWithinInt(gyroXDPS,lowG_x*1.2,highG_x*1.2))
195         gyroXDPS=0;
196     if(isWithinInt(gyroYDPS,lowG_y,highG_y))           gyroYDPS=0;
197     if(isWithinInt(gyroZDPS,lowG_z,highG_z))
198         gyroZDPS=0;
199
200     xAngle += (gyroXDPS*SCALING_GYRO *
201     ((millis()-lastMillis)/1000.0));
202     yAngle += (gyroYDPS*SCALING_GYRO *
203     ((millis()-lastMillis)/1000.0));
204     zAngle += (gyroZDPS*SCALING_GYRO *
205     ((millis()-lastMillis)/1000.0));
206
207     //printf("Raw: %d\n",combineX_accel.integer);
208     x_XL_raw = ((double)combineX_accel.integer) * SCALING_ACCEL;
209     y_XL_raw = ((double)combineY_accel.integer) * SCALING_ACCEL;
210     z_XL_raw = ((double)combineZ_accel.integer) * SCALING_ACCEL;
211
212     //printf("double: %f\n",x_XL);
213
214     lastMillis = millis();
215 }
216 }
217 }
218
219 void resetXAngle(void)
220 {
221     xAngle = 0;
222 }
223
224 void resetZAngle(void)
225 {
226     zAngle = 0;
227 }
228
229 uint16_t getXDPS(void)
230 {
231     return gyroXDPS;
232 }
233
234 uint16_t getYDPS(void)
235 {
236     return gyroYDPS;
237 }
238
239 uint16_t getZDPS(void)
240 {
241     return gyroZDPS;
242 }
243
244 double getXAxisAngle(void)
245 {
246     return xAngle;
247 }
248

```

```

249     double getYAxisAngle(void)
250     {
251         return yAngle;
252     }
253
254     double getZAxisAngle(void)
255     {
256         return zAngle;
257     }
258
259     double getXAxisAccel(void)
260     {
261         return x_XL_raw;
262     }
263
264     double getYAxisAccel(void)
265     {
266         return y_XL_raw;
267     }
268
269     double getZAxisAccel(void)
270     {
271         return z_XL_raw;
272     }
273
274     void zeroIMUAxisGyro(void)
275     {
276         double sumG_X=0,sumG_Y=0,sumG_Z=0;
277         int i=0;
278
279         //Wait for power on bootup
280         for(i=0;i<NUMBER_SAMPLES_CALIBRATE;i++)
281         {
282             while(!timerDone(&IMU_UpdateTimer));
283             //blinkDecimal(100);
284
285             WDTCONbits.WDTCLRKEY=0x5743;
286         }
287
288         //Calibrate using a number of samples defined above
289         for(i=0;i<NUMBER_SAMPLES_CALIBRATE;i++)
290         {
291
292             WDTCONbits.WDTCLRKEY=0x5743;
293             //Wait for 10ms
294             while(!timerDone(&IMU_UpdateTimer));
295
296             //Read the values
297             readIMU3AxisG();
298
299             //Sum all values up for an average
300             sumG_X+=combineX_gyro.integer;
301             sumG_Y+=combineY_gyro.integer;
302             sumG_Z+=combineZ_gyro.integer;
303
304             //Keep track of the highest values
305             if(combineX_gyro.integer>highG_x)
306                 highG_x=combineX_gyro.integer;
307             if(combineY_gyro.integer>highG_y)
308                 highG_y=combineY_gyro.integer;
309             if(combineZ_gyro.integer>highG_z)
310                 highG_z=combineZ_gyro.integer;
311
312             //Keep track of the lowest values
313             if(combineX_gyro.integer<lowG_x)
314                 lowG_x=combineX_gyro.integer;
315             if(combineY_gyro.integer<lowG_y)
316                 lowG_y=combineY_gyro.integer;
317             if(combineZ_gyro.integer<lowG_z)
318                 lowG_z=combineZ_gyro.integer;

```

```

314         blinkDecimal(100);
315     }
316
317     //CALIBRATE X
318     offsetG_X=(int16_t)(sumG_X/NUMBER_SAMPLES_CALIBRATE);
319     highG_x -= offsetG_X;
320     lowG_x  -= offsetG_X;
321     //printf("TotalX: %f\n",sumX);
322     printf("OffsetX: %d\n",offsetG_X);
323     printf("HighX: %d\n",highG_x);
324     printf("LowX: %d\r\n",lowG_x);
325
326     //CALIBRATE Y
327     offsetG_Y=(int16_t)(sumG_Y/NUMBER_SAMPLES_CALIBRATE);
328     highG_y -= offsetG_Y;
329     lowG_y  -= offsetG_Y;
330     //printf("TotalY: %f\n",sumY);
331     printf("OffsetY: %d\n",offsetG_Y);
332     printf("HighY: %d\n",highG_y);
333     printf("LowY: %d\r\n",lowG_y);
334
335     //CALIBRATE Z
336     offsetG_Z=(int16_t)(sumG_Z/NUMBER_SAMPLES_CALIBRATE);
337     highG_z -= offsetG_Z;
338     lowG_z  -= offsetG_Z;
339     //printf("TotalZ: %f\n",sumZ);
340     printf("OffsetZ: %d\n",offsetG_Z);
341     printf("HighZ: %d\n",highG_z);
342     printf("LowZ: %d\r\n",lowG_z);
343
344     //Lemme read the damn output
345     for(i=0;i<300;i++)
346     {
347         while(!timerDone(&IMU_UpdateTimer));
348         //blinkDecimal(100);
349         WDTCONbits.WDTCLRKEY=0x5743;
350     }
351 }
352
353
354 void zeroIMUAxisAll(void)
355 {
356     double sumG_X=0,sumG_Y=0,sumG_Z=0;
357     double sumXL_X=0,sumXL_Y=0,sumXL_Z=0;
358     int i=0;
359
360     //Wait for power on bootup
361     for(i=0;i<NUMBER_SAMPLES_CALIBRATE;i++)
362     {
363         while(!timerDone(&IMU_UpdateTimer));
364         //blinkDecimal(100);
365     }
366
367     //Calibrate using a number of samples defined above
368     for(i=0;i<NUMBER_SAMPLES_CALIBRATE;i++)
369     {
370         WDTCONbits.WDTCLRKEY=0x5743;
371         //Wait for 10ms
372         while(!timerDone(&IMU_UpdateTimer));
373
374         //Read the values
375         readIMU6Axis();
376
377         //Sum all values up for an average GYRO
378         sumG_X+=combineX_gyro.integer;

```

```

379     sumG_Y+=combineY_gyro.integer;
380     sumG_Z+=combineZ_gyro.integer;
381
382     //Sum all values up for an average XL
383     sumXL_X+=combineX_accel.integer;
384     sumXL_Y+=combineY_accel.integer;
385     sumXL_Z+=combineZ_accel.integer;
386
387     //Keep track of the highest values GYRO
388     if(combineX_gyro.integer>highG_x)      highG_x=combineX_gyro.integer;
389     if(combineY_gyro.integer>highG_y)      highG_y=combineY_gyro.integer;
390     if(combineZ_gyro.integer>highG_z)      highG_z=combineZ_gyro.integer;
391
392     //Keep track of the highest values XL
393     if(combineX_accel.integer>highXL_x)      highXL_x=combineX_accel.integer;
394     if(combineY_accel.integer>highXL_y)      highXL_y=combineY_accel.integer;
395     if(combineZ_accel.integer>highXL_z)      highXL_z=combineZ_accel.integer;
396
397     //Keep track of the lowest values GYRO
398     if(combineX_gyro.integer<lowG_x)          lowG_x=combineX_gyro.integer;
399     if(combineY_gyro.integer<lowG_y)          lowG_y=combineY_gyro.integer;
400     if(combineZ_gyro.integer<lowG_z)          lowG_z=combineZ_gyro.integer;
401
402     //Keep track of the lowest values XL
403     if(combineX_accel.integer<lowXL_x)          lowXL_x=combineX_accel.integer;
404     if(combineY_accel.integer<lowXL_y)          lowXL_y=combineY_accel.integer;
405     if(combineZ_accel.integer<lowXL_z)          lowXL_z=combineZ_accel.integer;
406
407     blinkDecimal(100);
408 }
409
410 //CALIBRATE X GYRO
411 offsetG_X=(int16_t)(sumG_X/NUMBER_SAMPLES_CALIBRATE);
412 highG_x -= offsetG_X;
413 lowG_x  -= offsetG_X;
414 //printf("TotalX: %f\n",sumX);
415 printf("OffsetX: %d\n",offsetG_X);
416 printf("HighX: %d\n",highG_x);
417 printf("LowX: %d\r\n",lowG_x);
418
419 //CALIBRATE Y GYRO
420 offsetG_Y=(int16_t)(sumG_Y/NUMBER_SAMPLES_CALIBRATE);
421 highG_y -= offsetG_Y;
422 lowG_y  -= offsetG_Y;
423 //printf("TotalY: %f\n",sumY);
424 printf("OffsetY: %d\n",offsetG_Y);
425 printf("HighY: %d\n",highG_y);
426 printf("LowY: %d\r\n",lowG_y);
427
428 //CALIBRATE Z GYRO
429 offsetG_Z=(int16_t)(sumG_Z/NUMBER_SAMPLES_CALIBRATE);
430 highG_z -= offsetG_Z;
431 lowG_z  -= offsetG_Z;
432 //printf("TotalZ: %f\n",sumZ);
433 printf("OffsetZ: %d\n",offsetG_Z);
434 printf("HighZ: %d\n",highG_z);
435 printf("LowZ: %d\r\n",lowG_z);
436
437 // //CALIBRATE X XL
438 // offsetG_X=(int16_t)(sumG_X/NUMBER_SAMPLES_CALIBRATE);
439 // highG_x -= offsetG_X;
440 // lowG_x  -= offsetG_X;
441 // //printf("TotalX: %f\n",sumX);
442 // printf("OffsetX: %d\n",offsetG_X);
443 // printf("HighX: %d\n",highG_x);

```

```

444 //     printf("LowX: %d\r\n",lowG_x);
445 //
446 //     //CALIBRATE Y XL
447 //     offsetG_Y=(int16_t)(sumG_Y/NUMBER_SAMPLES_CALIBRATE);
448 //     highG_y -= offsetG_Y;
449 //     lowG_y  -= offsetG_Y;
450 //     //printf("TotalY: %f\n",sumY);
451 //     printf("OffsetY: %d\n",offsetG_Y);
452 //     printf("HighY: %d\n",highG_y);
453 //     printf("LowY: %d\r\n",lowG_y);
454 //
455 //     //CALIBRATE Z XL
456 //     offsetG_Z=(int16_t)(sumG_Z/NUMBER_SAMPLES_CALIBRATE);
457 //     highG_z -= offsetG_Z;
458 //     lowG_z  -= offsetG_Z;
459 //     //printf("TotalZ: %f\n",sumZ);
460 //     printf("OffsetZ: %d\n",offsetG_Z);
461 //     printf("HighZ: %d\n",highG_z);
462 //     printf("LowZ: %d\r\n",lowG_z);
463
464 //Lemme read the damn output
465 //    for(i=0;i<300;i++)
466 //    {
467 //        while(!timerDone(&IMU_UpdateTimer));
468 //        //blinkDecimal(100);
469 //    }
470 }
471
472 void initIMU(void)
473 {
474     //Setup timer
475     setTimerInterval(&IMU_UpdateTimer,IMU_UPDATE_TIMER_VALUE);
476     setTimerInterval(&invertedTimer,250);
477     IMU_STATES_CONFIG_t states = START_BIT_CONFIG;
478     bool configIncomplete=true;
479     int configStep=0;
480     unsigned char controlRegister,dataRegister;
481     //Make sure another thread is not using the I2C
482     if(getMutex()==false)
483     {
484         setMutex(true);
485         while(configIncomplete)
486         {
487             switch(configStep)
488             {
489                 case GYRO_REGISTER:
490                     controlRegister      = CTRL2_G;
491                     dataRegister        = CTRL2_G_SETTING;
492                     break;
493                 case XL_REGISTER:
494                     controlRegister      = CTRL1_XL;
495                     dataRegister        = CTRL1_XL_SETTING;
496                     break;
497             }
498             bool writeIncomplete=true;
499             while(writeIncomplete)
500             {
501                 //Wait for idle bus
502                 while (! PLIB_I2C_BusIsIdle ( I2C_ID_1 ) );
503                 switch(states)
504                 {
505                     case START_BIT_CONFIG:
506                         PLIB_I2C_MasterStart(I2C_ID_1);
507                         states++;
508                         break;

```

```

509         case ADDR_BYTE_CONFIG:
510             PLIB_I2C_TransmitterByteSend(I2C_ID_1,IMU_ADDRESS_WRITE);
511             //while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
512             if(WaitTransmitterWriteCompleted())
513             {
514                 states++;
515             }
516             else
517             {
518
519                 resetI2CSystem();
520
521             break;
522         case COMMAND_BYTE_CONFIG:
523             PLIB_I2C_TransmitterByteSend(I2C_ID_1,controlRegister);
524             //while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
525             if(WaitTransmitterWriteCompleted())
526             {
527                 states++;
528             }
529             else
530             {
531
532                 resetI2CSystem();
533
534             break;
535         case DATA_BYTE_CONFIG:
536             PLIB_I2C_TransmitterByteSend(I2C_ID_1,dataRegister);
537             //while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
538             if(WaitTransmitterWriteCompleted())
539             {
540                 states++;
541             }
542             else
543             {
544
545                 resetI2CSystem();
546
547             break;
548         case STOP_BIT_CONFIG:
549             PLIB_I2C_MasterStop(I2C_ID_1);
550             states=START_BIT_CONFIG;
551             writeIncomplete=false;
552             break;
553         }
554     }
555
556     configStep++;
557     if(configStep>NUMBER_OF_CONFIG_STEPS)
558     {
559         configIncomplete=false;
560     }
561 }

```

```

562         setMutex(false);
563     }
564
565     //Calibrate zero for axis
566     resetTimer(&IMU_UpdateTimer);
567     zeroIMUAxisGyro();
568     resetTimer(&IMU_UpdateTimer);
569 }
570
571 void readIMU(int axis)
572 {
573     join_t combineStuff;
574     unsigned char registerToRead=OUTX_L_G;
575     switch(axis)
576     {
577         case X_AXIS_GYRO:
578             registerToRead=OUTX_L_G;
579             break;
580         case Y_AXIS_GYRO:
581             registerToRead=OUTY_L_G;
582             break;
583         case Z_AXIS_GYRO:
584             registerToRead=OUTZ_L_G;
585             break;
586     }
587
588     static IMU_STATES_READ_t states = START_BIT_READ;
589
590     bool writeIncomplete=true;
591     if(getMutex()==false)
592     {
593         setMutex(true);
594         while(writeIncomplete)
595         {
596             //Wait for idle bus
597             while (! PLIB_I2C_BusIsIdle ( I2C_ID_1 ) );
598             switch(states)
599             {
600                 case START_BIT_READ:
601                     PLIB_I2C_MasterStart(I2C_ID_1);
602                     states++;
603                     break;
604                 case ADDR_BYTE_WRITE:
605                     PLIB_I2C_TransmitterByteSend(I2C_ID_1,IMU_ADDRESS_WRITE);
606                     //while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
607                     if(WaitTransmitterWriteCompleted())
608                     {
609                         states++;
610                     }
611                     else
612                     {
613                         resetI2CSYSTEM();
614                     }
615                     states=START_BIT_READ;
616             }
617             break;
618             case REGISTER_BYTE_READ:
619                 PLIB_I2C_TransmitterByteSend(I2C_ID_1,registerToRead);
620                 //while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
621                 if(WaitTransmitterWriteCompleted())
622                 {
623                     states++;
624                 }
625             }
626         }
627     }
628 }

```

```

625
626
627     {
628         resetI2CSys tem();
629         states=START_BIT_READ;
630     }
631     break;
632 case RESTART_BIT:
633     PLIB_I2C_MasterStartRepeat(I2C_ID_1);
634     states++;
635     break;
636 case ADDR_BYTE_READ:
637     PLIB_I2C_TransmitterByteSend(I2C_ID_1,IMU_ADDRESS_READ);
638     //while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
639     if(WaitTransmitterWriteCompleted())
640     {
641         states++;
642     }
643     else
644     {
645         resetI2CSys tem();
646         states=START_BIT_READ;
647     }
648     break;
649 case CLOCK_BYTE1_READ:
650     PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
651     //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
652     if(WaitRxByteAvailable())
653     {
654         states++;
655     }
656     else
657     {
658         resetI2CSys tem();
659         states=START_BIT_READ;
660     }
661     break;
662 case GRAB_BYTE_L_READ:
663     combineStuff.parts[0]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
664     //printf("Low: %d",readStuff.parts[1]);
665     //while(!PLIB_I2C_MasterReceiverReadyToAcknowledge(I2C_ID_1));
666     states++;
667     break;
668 case ACK_BYTE_L_READ:
669     PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
670     //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1));
671     if(WaitReceiverByteAck())
672     {
673         states++;
674     }
675     else
676     {
677         resetI2CSys tem();
678         states=START_BIT_READ;
679     }
680     break;

```

```

680     case CLOCK_BYTE2_READ:
681         PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
682         //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
683         if(WaitRxByteAvailable())
684         {
685             states++;
686         }
687         else
688         {
689             resetI2CSysytem();
690             states=START_BIT_READ;
691         }
692         break;
693     case GRAB_BYTE_H_READ:
694         combineStuff.parts[1]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
695         //printf(" High: %d\r\n",readStuff.parts[0]);
696         //while(!PLIB_I2C_MasterReceiverReadyToAcknowledge(I2C_ID_1));
697         states++;
698         break;
699     case NACK_BYTE_H_READ:
700         PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,false);
701         //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1));
702         if(WaitReceiverByteAck())
703         {
704             states++;
705         }
706         else
707         {
708             resetI2CSysytem();
709             states=START_BIT_READ;
710         }
711         break;
712     case STOP_BIT_READ:
713         PLIB_I2C_MasterStop(I2C_ID_1);
714         states=START_BIT_READ;
715         writeIncomplete=false;
716         break;
717     }
718 }
719 setMutex(false);
720 //printf("Raw: %d\r\n",readStuff.integer);
721 switch(axis)
722 {
723     case X_AXIS_GYRO:
724         combineX_gyro.integer=combineStuff.integer;
725         break;
726     case Y_AXIS_GYRO:
727         combineY_gyro.integer=combineStuff.integer;
728         break;
729     case Z_AXIS_GYRO:
730         combineZ_gyro.integer=combineStuff.integer;
731         break;
732     }
733 }
734 }
735
736 void readIMU3AxisG(void)
737 {

```

```

739     static IMU_STATES_READ_3_t states = START_BIT_READ_3;
740
741     bool writeIncomplete=true;
742     if(getMutex()==false)
743     {
744         setMutex(true);
745         while(writeIncomplete)
746         {
747             //Wait for idle bus
748             while (! PLIB_I2C_BusIsIdle ( I2C_ID_1 ) );
749             switch(states)
750             {
751                 case START_BIT_READ_3:
752                     PLIB_I2C_MasterStart(I2C_ID_1);
753                     states++;
754                     break;
755                 case ADDR_BYTE_WRITE_3:
756                     PLIB_I2C_TransmitterByteSend(I2C_ID_1,IMU_ADDRESS_WRITE);
757                     //while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
758                     if(WaitTransmitterWriteCompleted())
759                     {
760                         states++;
761                     }
762                     else
763                     {
764                         resetI2CSysyem();
765
766                         states=START_BIT_READ_3;
767                     }
768                     break;
769                 case REGISTER_BYTE_READ_3:
770                     PLIB_I2C_TransmitterByteSend(I2C_ID_1,OUTX_L_G);
771                     //while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
772                     if(WaitTransmitterWriteCompleted())
773                     {
774                         states++;
775                     }
776                     else
777                     {
778                         resetI2CSysyem();
779
780                         states=START_BIT_READ_3;
781                     }
782                     break;
783                 case RESTART_BIT_3:
784                     PLIB_I2C_MasterStartRepeat(I2C_ID_1);
785                     states++;
786                     break;
787                 case ADDR_BYTE_READ_3:
788                     PLIB_I2C_TransmitterByteSend(I2C_ID_1,IMU_ADDRESS_READ);
789                     //while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
790                     if(WaitTransmitterWriteCompleted())
791                     {
792                         states++;
793                     }
794                     else
795                     {
796                         resetI2CSysyem();
797
798                         states=START_BIT_READ_3;
799                     }
800                     break;

```

```

798
799
800     case CLOCK_BYTE1_X_READ_3:
801         PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
802         //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
803         if(WaitRxByteAvailable())
804         {
805             states++;
806         }
807         else
808         {
809             resetI2CSysytem();
810             states=START_BIT_READ_3;
811         }
812         break;
813     case GRAB_BYTE_X_L_READ_3:
814         combineX_gyro.parts[0]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
815         states++;
816         break;
817     case ACK_BYTE_X_L_READ_3:
818         PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
819         //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1));
820         if(WaitReceiverByteAck())
821         {
822             states++;
823         }
824         else
825         {
826             resetI2CSysytem();
827             states=START_BIT_READ_3;
828         }
829         break;
830     case CLOCK_BYTE2_X_READ_3:
831         PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
832         //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
833         if(WaitRxByteAvailable())
834         {
835             states++;
836         }
837         else
838         {
839             resetI2CSysytem();
840             states=START_BIT_READ_3;
841         }
842         break;
843     case GRAB_BYTE_X_H_READ_3:
844         combineX_gyro.parts[1]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
845         states++;
846         break;
847     case ACK_BYTE_X_H_READ_3:
848         PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
849         //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1));
850         if(WaitReceiverByteAck())
851         {
852             states++;

```

```

853     }
854     else
855     {
856         resetI2CSys tem();
857
858         states=START_BIT_READ_3;
859
860     }
861     break;
862
863     case CLOCK_BYTE1_Y_READ_3:
864         PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
865         //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
866         if(WaitRxByteAvailable())
867         {
868             states++;
869         }
870         else
871         {
872             resetI2CSys tem();
873
874             states=START_BIT_READ_3;
875
876         }
877         break;
878
879     case GRAB_BYTE_Y_L_READ_3:
880         combineY_gyro.parts[0]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
881         states++;
882         break;
883
884     case ACK_BYTE_Y_L_READ_3:
885         PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
886
887         //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1)
888         //);
889         if(WaitReceiverByteAck())
890         {
891             states++;
892         }
893         else
894         {
895             resetI2CSys tem();
896
897             states=START_BIT_READ_3;
898
899         }
900         break;
901
902     case CLOCK_BYTE2_Y_READ_3:
903         PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
904         //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
905         if(WaitRxByteAvailable())
906         {
907             states++;
908         }
909         else
910         {
911             resetI2CSys tem();
912
913             states=START_BIT_READ_3;
914
915         }
916         break;
917
918     case GRAB_BYTE_Y_H_READ_3:
919         combineY_gyro.parts[1]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
920         states++;
921         break;

```

```

908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962

```

```

    case ACK_BYTE_Y_H_READ_3:
        PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
        //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1));
        if(WaitReceiverByteAck())
        {
            states++;
        }
        else
        {
            resetI2CSysytem();
            states=START_BIT_READ_3;
        }
        break;

    case CLOCK_BYTE1_Z_READ_3:
        PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
        //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
        if(WaitRxByteAvailable())
        {
            states++;
        }
        else
        {
            resetI2CSysytem();
            states=START_BIT_READ_3;
        }
        break;

    case GRAB_BYTE_Z_L_READ_3:
        combineZ_gyro.parts[0]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
        states++;
        break;

    case ACK_BYTE_Z_L_READ_3:
        PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
        //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1));
        if(WaitReceiverByteAck())
        {
            states++;
        }
        else
        {
            resetI2CSysytem();
            states=START_BIT_READ_3;
        }
        break;

    case CLOCK_BYTE2_Z_READ_3:
        PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
        //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
        if(WaitRxByteAvailable())
        {
            states++;
        }
        else
        {
            resetI2CSysytem();
        }

```

```

                states=START_BIT_READ_3;

        }
        break;
    case GRAB_BYTE_Z_H_READ_3:
        combineZ_gyro.parts[1]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
        states++;
        break;
    case NACK_BYTE_Z_H_READ_3:
        PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,false);
        //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1));
        if(WaitReceiverByteAck())
        {
            states++;
        }
        else
        {
            resetI2CSysytem();
        }
        states=START_BIT_READ_3;

    }
    break;
case STOP_BIT_READ_3:
    PLIB_I2C_MasterStop(I2C_ID_1);
    states=START_BIT_READ_3;
    writeIncomplete=false;
    break;
}
setMutex(false);
}
}

void readIMU6Axis(void)
{
    static IMU_STATES_READ_6_t states = START_BIT_READ_6;
    bool writeIncomplete=true;
    if(getMutex()==false)
    {
        setMutex(true);
        while(writeIncomplete)
        {
            //Wait for idle bus
            while (! PLIB_I2C_BusIsIdle ( I2C_ID_1 ) );
            switch(states)
            {
                case START_BIT_READ_6:
                    PLIB_I2C_MasterStart(I2C_ID_1);
                    states++;
                    break;
                case ADDR_BYTE_WRITE_6:
                    PLIB_I2C_TransmitterByteSend(I2C_ID_1,IMU_ADDRESS_WRITE);
                    //while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
                    if(WaitTransmitterWriteCompleted())
                    {
                        states++;
                    }
                    else
                    {
                        resetI2CSysytem();
                    }
                    break;
            }
        }
    }
}

```

```

1022
1023         states=START_BIT_READ_6;
1024
1025     }
1026     break;
1027 case REGISTER_BYTE_READ_6:
1028     PLIB_I2C_TransmitterByteSend(I2C_ID_1,OUTX_L_G);
1029     //while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
1030     if(WaitTransmitterWriteCompleted())
1031     {
1032         states++;
1033     }
1034     else
1035     {
1036         resetI2CSysytem();
1037
1038     states=START_BIT_READ_6;
1039
1040     }
1041     break;
1042 case RESTART_BIT_6:
1043     PLIB_I2C_MasterStartRepeat(I2C_ID_1);
1044     states++;
1045     break;
1046 case ADDR_BYTE_READ_6:
1047     PLIB_I2C_TransmitterByteSend(I2C_ID_1,IMU_ADDRESS_READ);
1048     //while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
1049     if(WaitTransmitterWriteCompleted())
1050     {
1051         states++;
1052     }
1053     else
1054     {
1055         resetI2CSysytem();
1056
1057     states=START_BIT_READ_6;
1058
1059     }
1060     break;
1061 case CLOCK_BYTE1_X_G_READ_6:
1062     PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
1063     //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
1064     if(WaitRxByteAvailable())
1065     {
1066         states++;
1067     }
1068     else
1069     {
1070         resetI2CSysytem();
1071
1072     states=START_BIT_READ_6;
1073
1074     }
1075     break;
1076 case GRAB_BYTE_X_G_L_READ_6:
1077     combineX_gyro.parts[0]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
1078     states++;
1079     break;
1080 case ACK_BYTE_X_G_L_READ_6:
1081     PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
1082
1083     //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1
1084     ));

```

```

1077         if(WaitReceiverByteAck())
1078     {
1079         states++;
1080     }
1081     else
1082     {
1083         resetI2CSysytem();
1084         states=START_BIT_READ_6;
1085     }
1086     break;
1087 case CLOCK_BYTE2_X_G_READ_6:
1088     PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
1089     //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
1090     if(WaitRxByteAvailable())
1091     {
1092         states++;
1093     }
1094     else
1095     {
1096         resetI2CSysytem();
1097         states=START_BIT_READ_6;
1098     }
1099     break;
1100 case GRAB_BYTE_X_G_H_READ_6:
1101     combineX_gyro.parts[1]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
1102     states++;
1103     break;
1104 case ACK_BYTE_X_G_H_READ_6:
1105     PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
1106     //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1));
1107     if(WaitReceiverByteAck())
1108     {
1109         states++;
1110     }
1111     else
1112     {
1113         resetI2CSysytem();
1114         states=START_BIT_READ_6;
1115     }
1116     break;
1117 case CLOCK_BYTE1_Y_G_READ_6:
1118     PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
1119     //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
1120     if(WaitRxByteAvailable())
1121     {
1122         states++;
1123     }
1124     else
1125     {
1126         resetI2CSysytem();
1127         states=START_BIT_READ_6;
1128     }
1129     break;
1130 case GRAB_BYTE_Y_G_L_READ_6:

```

```

1132     combineY_gyro.parts[0]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
1133     states++;
1134     break;
1135   case ACK_BYTE_Y_G_L_READ_6:
1136     PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
1137
1138     //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1)
1139     //);
1140     if(WaitReceiverByteAck())
1141     {
1142       states++;
1143     }
1144     else
1145     {
1146       resetI2CSYSTEM();
1147
1148       states=START_BIT_READ_6;
1149
1150     }
1151     break;
1152   case CLOCK_BYTE2_Y_G_READ_6:
1153     PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
1154     //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
1155     if(WaitRxByteAvailable())
1156     {
1157       states++;
1158     }
1159     else
1160     {
1161       resetI2CSYSTEM();
1162
1163       states=START_BIT_READ_6;
1164
1165     }
1166     break;
1167   case GRAB_BYTE_Y_G_H_READ_6:
1168     combineY_gyro.parts[1]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
1169     states++;
1170     break;
1171   case ACK_BYTE_Y_G_H_READ_6:
1172     PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
1173
1174     //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1)
1175     //);
1176     if(WaitReceiverByteAck())
1177     {
1178       states++;
1179     }
1180     else
1181     {
1182       resetI2CSYSTEM();
1183
1184       states=START_BIT_READ_6;
1185
1186     }
1187     break;
1188
1189   case CLOCK_BYTE1_Z_G_READ_6:
1190     PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
1191     //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
1192     if(WaitRxByteAvailable())

```

```

1183             {
1184                 states++;
1185             }
1186         else
1187         {
1188             resetI2CSYSTEM();
1189             states=START_BIT_READ_6;
1190         }
1191         break;
1192     case GRAB_BYTE_Z_G_L_READ_6:
1193         combineZ_gyro.parts[0]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
1194         states++;
1195         break;
1196     case ACK_BYTE_Z_G_L_READ_6:
1197         PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
1198         //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1))
1199         ());
1200         if(WaitReceiverByteAck())
1201         {
1202             states++;
1203         }
1204         else
1205         {
1206             resetI2CSYSTEM();
1207         }
1208         break;
1209     case CLOCK_BYTE2_Z_G_READ_6:
1210         PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
1211         //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
1212         if(WaitRxByteAvailable())
1213         {
1214             states++;
1215         }
1216         else
1217         {
1218             resetI2CSYSTEM();
1219         }
1220         states=START_BIT_READ_6;
1221     }
1222     break;
1223     case GRAB_BYTE_Z_G_H_READ_6:
1224         combineZ_gyro.parts[1]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
1225         states++;
1226         break;
1227     case ACK_BYTE_Z_G_H_READ_6:
1228         PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
1229         //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1))
1230         ());
1231         if(WaitReceiverByteAck())
1232         {
1233             states++;
1234         }
1235         else

```

```

        resetI2CSYSTEM();

1236         states=START_BIT_READ_6;

1237     }
1238     break;
1239
1240
1241     case CLOCK_BYTE1_X_XL_READ_6:
1242         PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
1243         //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
1244         if(WaitRxByteAvailable())
1245         {
1246             states++;
1247         }
1248         else
1249         {
1250             resetI2CSYSTEM();
1251
1252             states=START_BIT_READ_6;
1253
1254         }
1255         break;
1256     case GRAB_BYTE_X_XL_L_READ_6:
1257         combineX_accel.parts[0]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
1258         states++;
1259         break;
1260     case ACK_BYTE_X_XL_L_READ_6:
1261         PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
1262
1263         //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1));
1264         if(WaitReceiverByteAck())
1265         {
1266             states++;
1267         }
1268         else
1269         {
1270             resetI2CSYSTEM();
1271
1272         }
1273         break;
1274     case CLOCK_BYTE2_X_XL_READ_6:
1275         PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
1276         //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
1277         if(WaitRxByteAvailable())
1278         {
1279             states++;
1280         }
1281         else
1282         {
1283             resetI2CSYSTEM();
1284
1285             states=START_BIT_READ_6;
1286
1287         }
1288         break;
1289     case GRAB_BYTE_X_XL_H_READ_6:
1290         combineX_accel.parts[1]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
1291         states++;

```

```

1287         break;
1288     case ACK_BYTE_X_XL_H_READ_6:
1289         PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
1290         //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1))
1291         //{
1292             if(WaitReceiverByteAck())
1293             {
1294                 states++;
1295             }
1296             else
1297             {
1298                 resetI2CSysytem();
1299                 states=START_BIT_READ_6;
1300             }
1301             break;
1302     case CLOCK_BYTE1_Y_XL_READ_6:
1303         PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
1304         //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
1305         if(WaitRxByteAvailable())
1306         {
1307             states++;
1308         }
1309         else
1310         {
1311             resetI2CSysytem();
1312             states=START_BIT_READ_6;
1313         }
1314         break;
1315     case GRAB_BYTE_Y_XL_L_READ_6:
1316         combineY_accel.parts[0]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
1317         states++;
1318         break;
1319     case ACK_BYTE_Y_XL_L_READ_6:
1320         PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
1321         //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1))
1322         //{
1323             if(WaitReceiverByteAck())
1324             {
1325                 states++;
1326             }
1327             else
1328             {
1329                 resetI2CSysytem();
1330                 states=START_BIT_READ_6;
1331             }
1332             break;
1333     case CLOCK_BYTE2_Y_XL_READ_6:
1334         PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
1335         //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
1336         if(WaitRxByteAvailable())
1337         {
1338             states++;
1339         }
1340         else
1341             {
1342                 resetI2CSysytem();

```

```

1342                         states=START_BIT_READ_6;
1343
1344                     }
1345                     break;
1346                 case GRAB_BYTE_Y_XL_H_READ_6:
1347                     combineY_accel.parts[1]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
1348                     states++;
1349                     break;
1350                 case ACK_BYTE_Y_XL_H_READ_6:
1351                     PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
1352
1353                     //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1)
1354                     //));
1355                     if(WaitReceiverByteAck())
1356                     {
1357                         states++;
1358                     }
1359                     else
1360                     {
1361                         resetI2CSystem();
1362
1363                         states=START_BIT_READ_6;
1364
1365                     }
1366                     break;
1367
1368                 case CLOCK_BYTE1_Z_XL_READ_6:
1369                     PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
1370                     //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
1371                     if(WaitRxByteAvailable())
1372                     {
1373                         states++;
1374                     }
1375                     break;
1376                 case GRAB_BYTE_Z_XL_L_READ_6:
1377                     combineZ_accel.parts[0]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
1378                     states++;
1379                     break;
1380                 case ACK_BYTE_Z_XL_L_READ_6:
1381                     PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1,true);
1382
1383                     //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1)
1384                     //));
1385                     if(WaitReceiverByteAck())
1386                     {
1387                         states++;
1388                     }
1389                     else
1390                     {
1391                         resetI2CSystem();
1392
1393                         states=START_BIT_READ_6;
1394
1395                     }
1396                     break;
1397                 case CLOCK_BYTE2_Z_XL_READ_6:
1398                     PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);

```

```

1395 //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
1396 if(WaitRxByteAvailable())
1397 {
1398     states++;
1399 }
1400 else
1401 {
1402     resetI2CSysytem();
1403     states=START_BIT_READ_6;
1404 }
1405 break;
1406 case GRAB_BYTE_Z_XL_H_READ_6:
1407     combineZ_accel.parts[1]=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
1408     states++;
1409     break;
1410
1411
1412
1413 case NACK_BYTE_Z_XL_H_READ_6:
1414     PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1, false);
1415     //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1));
1416     if(WaitReceiverByteAck())
1417     {
1418         states++;
1419     }
1420     else
1421     {
1422         resetI2CSysytem();
1423         states=START_BIT_READ_6;
1424     }
1425     break;
1426
1427 case STOP_BIT_READ_6:
1428     PLIB_I2C_MasterStop(I2C_ID_1);
1429     states=START_BIT_READ_6;
1430     writeIncomplete=false;
1431     break;
1432 }
1433 setMutex(false);
1434 }
1435 }
1436 }
```

```

1  #include "LIDARSensors.h"
2
3  /* ====== Data from PSU ===== */
4  uint16_t rawSensorDistances[8] = {0,0,0,0,0,0,0,0};
5  double sensorDistances[8]
6  ={0,0,0,0,0,0,0}; // {483.6599, 929.33, 859.0, 856.6, 791.8, 450.5, 416.4, 523.2}; // {21.2
7  836, 22.0676, 17.0268, 17.6540, 21.2836, 22.0676, 25.5403, 26.4811}; // Result should be
8  20
9
10 void setSensorDistance(uint8_t sensorNumber, double sensorValue)
11 {
12     sensorDistances[sensorNumber]=sensorValue;
13 }
14
15 double getSensorDistance(uint8_t sensorNumber)
16 {
17     return sensorDistances[sensorNumber];
18 }
19 double getSensorDistanceCenterOfRobot(uint8_t sensorNumber)
20 {
21     return sensorDistances[sensorNumber]+38;
22 }
23
24 double * getSensorDistancesArray(void)
25 {
26     return sensorDistances;
27 }
28
29     double sensorDistancesOffset[8];
30 double * getSensorDistancesArrayCenterOfRobot(void)
31 {
32     int i=0;
33     for(i=0;i<8;i++)
34     {
35         sensorDistancesOffset[i]=sensorDistances[i]+38;
36     }
37     return (double*)sensorDistancesOffset;
38 }
39
40 double distancesSubarray[7];
41
42 double * getSensorDistancesSubarray7(uint8_t indexToSkip)
43 {
44     int i=0,j=0;
45     for(i=0;i<8;i++)
46     {
47         if(i==indexToSkip)//if this is the index to skip
48         {
49             //Do not load the index
50             //Do not increment the subarray index pointer
51         }
52     else
53     {
54         distancesSubarray[j]=sensorDistances[i];
55         j++;
56     }

```

```
57
58      }
59      return distancesSubarray;
60  }
61
```

```

1  /* **** **** **** **** **** **** **** **** **** **** **** **** **** */
2  /** Descriptive File Name
3
4      @Company
5          Company Name
6
7      @File Name
8          filename.c
9
10     @Summary
11         Brief description of the file.
12
13     @Description
14         Describe the purpose of this file.
15     */
16    /* **** **** **** **** **** **** **** **** **** **** **** **** */
17
18 #include "macros.h"
19 #include "demonstrationSetup.h"
20 #include "AutonomousMode.h"
21 #include "driveAssistance.h"
22 #include "IMU.h"
23 #include "runWalls.h"
24 #include "MillistTimer.h"
25 #include "runBackAndForthFindEnemyBoth.h"
26 #include "spinWeaponUp.h"
27 #include "FastTransfer.h"
28 #include "motorControl.h"
29 #include "doTurn.h"
30
31
32 uint8_t macroCommand=0;
33
34 void updateMacros(void)
35 {
36     if(macroCommand!=0)
37     {
38         MACRO_LIGHT = 1;
39         switch(macroCommand)
40         {
41             case TURN:
42                 // writeDigitSevenSeg(1, false);
43                 doTurn(180,0);
44                 //writeDigitSevenSeg(0, false);
45                 macroCommand=0;
46                 break;
47             case APPROACH:
48                 //writeDigitSevenSeg(1, false);
49                 doApproach(50);
50                 //writeDigitSevenSeg(0, false);
51                 macroCommand=0;
52                 break;
53             case PERPENDICULAR:
54                 //writeDigitSevenSeg(1, false);
55                 findPerpendicular(2);
56                 //writeDigitSevenSeg(0, false);
57                 macroCommand=0;
58                 break;
59             case WALLS:
60                 //writeDigitSevenSeg(1, false);
61                 runWalls();
62                 //writeDigitSevenSeg(0, false);
63                 macroCommand=0;
64                 break;
65         }
66     }
67 }

```

```

66     case BAF:
67         //writeDigitSevenSeg(1, false);
68         runBackAndForth();
69         //writeDigitSevenSeg(0, false);
70         macroCommand=0;
71         break;
72     case BAF_ENEMY:
73         //writeDigitSevenSeg(1, false);
74         runBackAndForthFindEnemyBoth();
75         //writeDigitSevenSeg(0, false);
76         macroCommand=0;
77         break;
78     case AUTONOMOUS:
79         runAutonomous();
80         macroCommand=0;
81         break;
82     case AUTONOMOUS2:
83         runAutonomous2();
84         macroCommand=0;
85         break;
86     case SPIN_UP:
87         //writeDigitSevenSeg(1, false);
88         spinWeaponUp(25);
89         //writeDigitSevenSeg(0, false);
90         macroCommand=0;
91         break;
92     case SPIN_AND_MOVE:
93         //writeDigitSevenSeg(1, false);
94         testMovementSpinning();
95         //writeDigitSevenSeg(0, false);
96         macroCommand=0;
97         break;
98     case WALLS_ENEMY:
99         //writeDigitSevenSeg(1, false);
100        runWallsFindEnemy(25);
101        //writeDigitSevenSeg(0, false);
102        macroCommand=0;
103
104        break;
105    }
106}
107
108
109     MACRO_LIGHT = 0;
110 }
111 }
112
113 void setMacroCommand(uint8_t mc)
114 {
115     macroCommand = mc;
116 }
117
118 uint8_t getMacroCommand(void)
119 {
120     return macroCommand;
121 }
122
123 bool checkMacroExists(void)
124 {
125     return macroCommand!=0;
126 }
127
128 void completeMacro(void)
129 {
130     resetXAngle();

```

```
131     setGyroRelativeAngle();
132     //CLEANUP WHEN WE ARE DONE
133     setLeftMotorSpeed(0);
134     setRightMotorSpeed(0);
135     //setWeaponMotorSpeed(0);
136
137
138 //    ToSend(SPEED_SETTING_DATA_ADDRESS,0);
139 //    sendData(MDB_L_ADDRESS);
140 //    ToSend(SPEED_SETTING_DATA_ADDRESS,0);
141 //    sendData(MDB_R_ADDRESS);
142
143 }
144
145 void checkWDT(void)
146 {
147
148     WDTCONbits.WDTCLRKEY=0x5743;
149 }
150
151 /* **** End of File ****
152 */
153
154
```

```

1  ****
2      MPLAB Harmony Project Main Source File
3
4  Company:
5      Microchip Technology Inc.
6
7  File Name:
8      main.c
9
10 Summary:
11     This file contains the "main" function for an MPLAB Harmony project.
12
13 Description:
14     This file contains the "main" function for an MPLAB Harmony project. The
15     "main" function calls the "SYS_Initialize" function to initialize the state
16     machines of all MPLAB Harmony modules in the system and it calls the
17     "SYS_Tasks" function from within a system-wide "super" loop to maintain
18     their correct operation. These two functions are implemented in
19     configuration-specific files (usually "system_init.c" and "system_tasks.c")
20     in a configuration-specific folder under the "src/system_config" folder
21     within this project's top-level folder. An MPLAB Harmony project may have
22     more than one configuration, each contained within its own folder under
23     the "system_config" folder.
24 ****
25
26 // DOM-IGNORE-BEGIN
27 ****
28 Copyright (c) 2013-2014 released Microchip Technology Inc. All rights reserved.
29
30 //Microchip licenses to you the right to use, modify, copy and distribute
31 Software only when embedded on a Microchip microcontroller or digital signal
32 controller that is integrated into your product or third party product
33 (pursuant to the sublicense terms in the accompanying license agreement).
34
35 You should refer to the license agreement accompanying this Software for
36 additional information regarding your rights and obligations.
37
38 SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
39 EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
40 MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
41 IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
42 CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
43 OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
44 INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
45 CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
46 SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
47 (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
48 ****
49 // DOM-IGNORE-END
50
51
52 // ****
53 // ****
54 // Section: Included Files
55 // ****
56 // ****
57
58 #include <stddef.h>           // Defines NULL
59 #include <stdbool.h>          // Defines true
60 #include <stdlib.h>            // Defines EXIT_FAILURE
61 #include "system/common/sys_module.h" // SYS function prototypes
62 #include "TimerlessDelay.h"
63
64 // ****
65 // ****

```

```

66 // Section: Main Entry Point
67 // ****
68 // ****
69
70 int main ( void )
{
    //CLEAR I2C handstate by toggling lines
73 TRISDbits.TRISD10 =0;
74 TRISDbits.TRISD9 =0;
75
76 hardDelay_cycles(10);
77
78 LATDbits.LATD9 =0;
79 hardDelay_cycles(10);
80 LATDbits.LATD9 =0;
81 hardDelay_cycles(10);
82 LATDbits.LATD9 =0;
83 hardDelay_cycles(10);
84 LATDbits.LATD10 ^=1;
85 hardDelay_cycles(10);
86 LATDbits.LATD10 ^=1;
87 hardDelay_cycles(10);
88 LATDbits.LATD10 ^=1;
89 hardDelay_cycles(10);
90 LATDbits.LATD10 ^=1;
91 hardDelay_cycles(10);
92 LATDbits.LATD10 ^=1;
93 hardDelay_cycles(10);
94 LATDbits.LATD10 ^=1;
95 hardDelay_cycles(10);
96 LATDbits.LATD10 ^=1;
97 hardDelay_cycles(10);
98 LATDbits.LATD10 ^=1;
99 hardDelay_cycles(10);
100 LATDbits.LATD9 =1;
101 hardDelay_cycles(10);
102
103 TRISDbits.TRISD9 =1;
104 TRISDbits.TRISD10 =1;
105
106 hardDelay_ms(100);
107
108 /* Initialize all MPLAB Harmony modules, including application(s). */
109 SYS_Initialize ( NULL );
110
111
112 while ( true )
{
113     /* Maintain state machines of all polled MPLAB Harmony modules. */
114     SYS_Tasks ( );
115
116 }
117
118
119 /* Execution should not come here during normal operation */
120
121     return ( EXIT_FAILURE );
122 }
123
124
125 /* ****
126 End of File
127 */
128
129

```

```

1  /* **** */
2  /** Descriptive File Name
3
4  @Company
5      Company Name
6
7  @File Name
8      filename.c
9
10 @Summary
11     Brief description of the file.
12
13 @Description
14     Describe the purpose of this file.
15 */
16 /* **** */
17
18 #include "MicrosTimer.h"
19 #include "../src/system_config/default/system_definitions.h"
20
21
22 unsigned long micros(void)
23 {
24     return PLIB_TMR_Counter32BitGet(TMR_ID_4);
25 }
26
27 bool timerDoneMicros(timers_micros_t * t)
28 {
29     if((abs(micros())-t->lastMicros)/12) > t->timerInterval)
30     {
31         t->lastMicros=micros();
32         return true;
33     }
34     else
35     {
36         return false;
37     }
38 }
39
40 void setTimerIntervalMicros(timers_micros_t * t, unsigned long interval)
41 {
42     t->timerInterval= interval;
43 }
44
45 void resetTimerMicros(timers_micros_t * t)
46 {
47     t->lastMicros=micros();
48 }
49
50
51 /* **** */
52 End of File
53 */
54

```

```

1  #include "MillisTimer.h"
2
3 //Each interrupt is 100us so we count to 10x and thats a millisecond
4 #define MILLIS_INTERVAL 10
5
6 unsigned long globalTimeMillis;
7 unsigned short millisCounter=0;
8
9
10 unsigned long millis(void)
11 {
12     return globalTimeMillis;
13 }
14
15 bool timerDone(timers_t * t)
16 {
17     if(abs(millis()-t->lastMillis)>t->timerInterval)
18     {
19         t->lastMillis=millis();
20         return true;
21     }
22     else
23     {
24         return false;
25     }
26 }
27
28 void setTimerInterval(timers_t * t, unsigned long interval)
29 {
30     t->timerInterval= interval;
31     resetTimer(t);
32 }
33
34 void resetTimer(timers_t * t)
35 {
36     t->lastMillis=millis();
37 }
38
39 void globalTimerTracker( )
40 {
41     //Wait a number of interrupts to count a millisecond
42 //     if(millisCounter>MILLIS_INTERVAL)
43 //     {
44 //         globalTimeMillis++;
45 //         millisCounter = 0;
46 //     }
47 //     else
48 //     {
49 //         millisCounter++;
50 //     }
51 }
52 }
```



```

1  #include "motorControl.h"
2  #include "IMU.h"
3  #include "FastTransfer.h"
4  #include "driveAssistance.h"
5
6  #define HORIZONTAL_COMPENSATION_NORMAL_MODE_STOPPED 0.3
7  #define HORIZONTAL_COMPENSATION_NORMAL_MODE_DRIVING 0.2
8  #define HORIZONTAL_COMPENSATION_BBC_MODE_STOPPED 0.3
9  #define HORIZONTAL_COMPENSATION_BBC_MODE_DRIVING 0.1
10 #define VERTICAL_COMPENSATION_NORMAL_MODE 0.5
11 #define VERTICAL_COMPENSATION_BBC_MODE 0.3
12
13 #define USE_GYRO_FEEDBACK_CORRECTION
14
15 bool wasHorizontal=false;
16 int boundMotorCommands(int command);
17 int boundWeaponMotorCommands(int command);
18 int leftSpeed=STOPPED_MOTOR_COMMAND, rightSpeed=STOPPED_MOTOR_COMMAND,
weaponSpeed = STOPPED_MOTOR_COMMAND;
19
20 bool firstTime=true;
21 timers_t debugTimer;
22 //Mix the joystick inputs into a motor speed for left and right
23 void mixJoysticks(int vert, int horizon)
24 {
25     if(firstTime)
26     {
27         setTimerInterval (&debugTimer,100);
28         firstTime=false;
29         setGyroRelativeAngle();
30     }
31     int lm=0, rm=0;
32     switch(getOperationalMode())
33     {
34         case NORMAL_MODE:
35 #ifdef USE_GYRO_FEEDBACK_CORRECTION
36             if(!isInverted())
37             {
38                 if(vert==0)
39                 {
40                     if(horizon==0)
41                     {
42                         motorGyroFeedbackStraight (&lm,&rm);
43                     }
44                     else
45                     {
46                         lm =
47                             (int)vert*VERTICAL_COMPENSATION_NORMAL_MODE+horizon*HORIZONTAL_COMPENSATION_NORMAL_MODE_STOPPED;
48                         rm =
49                             (int)vert*VERTICAL_COMPENSATION_NORMAL_MODE-horizon*HORIZONTAL_COMPENSATION_NORMAL_MODE_STOPPED;
50                         setGyroRelativeAngle();
51                     }
52                 }
53                 lm =
54                     (int)vert*VERTICAL_COMPENSATION_NORMAL_MODE+(horizon*HORIZONTAL_COMPENSATION_NORMAL_MODE_DRIVING* (255.0-abs(vert))/255.0);
55                 rm =
56                     (int)vert*VERTICAL_COMPENSATION_NORMAL_MODE-(horizon*HORIZONTAL_COMPENSATION_NORMAL_MODE_DRIVING* (255.0-abs(vert))/255.0);
57                 if(horizon==0)
58                 {

```

```

57             if(wasHorizontal)
58             {
59                 wasHorizontal=false;
60                 setGyroRelativeAngle();
61             }
62             motorGyroFeedbackStraight(&lm,&rm);
63         }
64     else
65     {
66         wasHorizontal=true;
67         setGyroRelativeAngle();
68     }
69 }
70 }
71 else
72 {
73     if(vert==0)
74     {
75         if(horizon==0)
76         {
77             motorGyroFeedbackStraight(&lm,&rm);
78         }
79         else
80         {
81             lm =
82                 vert*VERTICAL_COMPENSATION_NORMAL_MODE-horizon*HORIZONTAL_
83                 COMPENSATION_NORMAL_MODE_STOPPED;
84             rm =
85                 vert*VERTICAL_COMPENSATION_NORMAL_MODE+horizon*HORIZONTAL_
86                 COMPENSATION_NORMAL_MODE_STOPPED;
87             setGyroRelativeAngle();
88         }
89     }
90     else
91     {
92         lm =
93             vert*VERTICAL_COMPENSATION_NORMAL_MODE-(horizon*HORIZONTAL_COM_
94                 PENSATION_NORMAL_MODE_DRIVING*(255.0-abs(vert))/255.0);
95         rm =
96             vert*VERTICAL_COMPENSATION_NORMAL_MODE+(horizon*HORIZONTAL_COM_
97                 PENSATION_NORMAL_MODE_DRIVING*(255.0-abs(vert))/255.0);
98         if(horizon==0)
99         {
100             if(wasHorizontal)
101             {
102                 wasHorizontal=false;
103                 setGyroRelativeAngle();
104             }
105             motorGyroFeedbackStraight(&lm,&rm);
106         }
107     }
108 #else
109 if(vert==0)
110 {
111     lm =
112         vert*VERTICAL_COMPENSATION_NORMAL_MODE+horizon*HORIZONTAL_COMPENSA-
113             TION_NORMAL_MODE_STOPPED;

```

```

112     rm =
113     vert*VERTICAL_COMPENSATION_NORMAL_MODE-horizon*HORIZONTAL_COMPENSA-
114     TION_NORMAL_MODE_STOPPED;
115   }
116   else
117   {
118     lm =
119     vert*VERTICAL_COMPENSATION_NORMAL_MODE+(horizon*HORIZONTAL_COMPENS-
120     ATION_NORMAL_MODE_DRIVING* (255.0-abs(vert))/255.0);
121     rm =
122     vert*VERTICAL_COMPENSATION_NORMAL_MODE-(horizon*HORIZONTAL_COMPENS-
123     ATION_NORMAL_MODE_DRIVING* (255.0-abs(vert))/255.0);
124   }
125   #endif
126   break;
127 case BABY_CAKES_MODE:
128 if(!isInverted())
129 {
130   if(vert==0)
131   {
132     lm =
133     vert*VERTICAL_COMPENSATION_BBC_MODE+horizon*HORIZONTAL_COMPENSATIO-
134     N_BBC_MODE_STOPPED;
135     rm =
136     vert*VERTICAL_COMPENSATION_BBC_MODE-horizon*HORIZONTAL_COMPENSATIO-
137     N_BBC_MODE_STOPPED;
138   }
139   else
140   {
141     if(vert==0)
142     {
143       lm =
144       vert*VERTICAL_COMPENSATION_BBC_MODE-horizon*HORIZONTAL_COMPENSATIO-
145       N_BBC_MODE_STOPPED;
146       rm =
147       vert*VERTICAL_COMPENSATION_BBC_MODE+horizon*HORIZONTAL_COMPENSATIO-
148       N_BBC_MODE_STOPPED;
149     }
150   }
151 }
152 //  if(lm!=rm)
153 //    //if(timerDone(&debugTimer))
154 //    {

```

```

155 //      printf("v: %3d, h: %3d, lm: %3d, rm: %3d\r\n",vert,horizon,lm,rm);
156 //    }
157     setLeftMotorSpeed(boundMotorCommands(lm));
158     setRightMotorSpeed(boundMotorCommands(rm));
159   }
160
161 //Bound the motor commands to the max and min allowable values
162 int boundMotorCommands(int command)
163 {
164   //Maximums
165   if(command>MAX_POS_MOTOR_COMMAND)
166   {
167     command=MAX_POS_MOTOR_COMMAND;
168   }
169   if(command<MAX_NEG_MOTOR_COMMAND)
170   {
171     command=MAX_NEG_MOTOR_COMMAND;
172   }
173
174   //Minimums
175 //  if(command!=0)
176 //  {
177 //    if(abs(command)<30)
178 //    {
179 //      if(command<0)
180 //      {
181 //        command=-30;
182 //      }
183 //      else
184 //      {
185 //        command=30;
186 //      }
187 //    }
188 //  }
189   return command;
190 }
191
192 int boundWeaponMotorCommands(int command)
193 {
194   //Maximums
195   if(command>MAX_WEAPON_COMMAND)
196   {
197     command=MAX_WEAPON_COMMAND;
198   }
199   if(command<MIN_WEAPON_COMMAND)
200   {
201     command=MIN_WEAPON_COMMAND;
202   }
203
204   return command;
205 }
206
207 //Set the motor speed variable (for external code usage)
208 void setRightMotorSpeed(int rm)
209 {
210   if(!isInverted())
211     rightSpeed=rm;
212   else
213     rightSpeed=-rm;
214
215 }
216
217 //Get the motor speed variable (for external code usage)
218 int getRightMotorSpeed(void)
219 {

```

```

220     return rightSpeed;
221 }
222
223 //Set the motor speed variable (for external code usage)
224 void setLeftMotorSpeed(int lm)
{
225     if(!isInverted())
226         leftSpeed=lm;
227     else
228         leftSpeed=-lm;
229 }
230
231
232 //Get the motor speed variable (for external code usage)
233 int getLeftMotorSpeed(void)
{
234     return leftSpeed;
235 }
236
237
238 //Set the motor speed variable (for external code usage)
239 void setWeaponMotorSpeed(int wm)
{
240     weaponSpeed=wm;
241 }
242
243
244 //Get the motor speed variable (for external code usage)
245 int getWeaponMotorSpeed(void)
{
246     return weaponSpeed;
247 }
248
249 timers_t commDelay;
250 void stopAllMotors(void)
{
251     setTimerInterval(&commDelay,5);
252     setLeftMotorSpeed(0);
253     setRightMotorSpeed(0);
254
255     ToSend(SPEED_SETTING_DATA_ADDRESS,0);
256     sendData(MDB_L_ADDRESS);
257     while(!timerDone(&commDelay));
258
259     ToSend(SPEED_SETTING_DATA_ADDRESS,0);
260     sendData(MDB_R_ADDRESS);
261     while(!timerDone(&commDelay));
262 }
263
264
265 void testRightMotorControl(void)
{
266     static int ramper=0;
267     static bool rampingUp=true;
268     setRightMotorSpeed(ramper);
269     if(rampingUp && ramper<MAX_POS_MOTOR_COMMAND)
270     {
271         ramper++;
272     }
273     else if(rampingUp)
274     {
275         rampingUp = false;
276     }
277     else if(!rampingUp && ramper > MAX_NEG_MOTOR_COMMAND)
278     {
279         ramper--;
280     }
281     else
282     {
283         rampingUp=true;
284     }
}

```

```

285     }
286 }
287
288 void testLeftMotorControl(void)
289 {
290     static int ramper=0;
291     static bool rampingUp=true;
292     setLeftMotorSpeed(ramper);
293     if(rampingUp && ramper<MAX_POS_MOTOR_COMMAND)
294     {
295         ramper++;
296     }
297     else if(rampingUp)
298     {
299         rampingUp = false;
300     }
301     else if(!rampingUp && ramper > MAX_NEG_MOTOR_COMMAND)
302     {
303         ramper--;
304     }
305     else
306     {
307         rampingUp=true;
308     }
309 }
310
311
312 void testWeaponMotorControl(void)
313 {
314     static int ramper=0;
315     static bool rampingUp=true;
316     setWeaponMotorSpeed(ramper);
317     if(rampingUp && ramper<MAX_WEAPON_COMMAND)
318     {
319         ramper++;
320     }
321     else if(rampingUp)
322     {
323         rampingUp = false;
324     }
325     else if(!rampingUp && ramper > MIN_WEAPON_COMMAND)
326     {
327         ramper--;
328     }
329     else
330     {
331         rampingUp=true;
332     }
333 }

```

```

1  #include "PID.h"
2  #include "MillisTimer.h"
3
4  double target;
5  double kp,ki,kd;
6  double proportional, derivative, integral;
7  double errorSum, error, lastError, derror;
8  double output;
9  double dt;
10 unsigned long lastMillisPID;
11
12
13 void resetPIDController(double _target, double _kp, double _ki, double _kd)
14 {
15     //Set a new target
16     target=_target;
17
18     //Accept the tuning settings
19     kp=_kp;
20     ki=_ki;
21     kd=_kd;
22
23     //Wipe state variables
24     error      =0;
25     derror     =0;
26     lastError   =0;
27     errorSum   =0;
28     proportional=0;
29     derivative  =0;
30     integral    =0;
31     output      =0;
32
33     //Record current milliseconds
34     lastMillisPID = millis();
35 }
36
37 //Update the state of the PID controller
38 void updatePIDOutput(double sample)
39 {
40     //Calculate the time between updates (seconds)
41     dt           = (millis()-lastMillisPID) / 1000.0;
42
43     //Move the error from last time to the old error
44     lastError    = error;
45
46     //Update the error now
47     error        = target - sample;
48
49     //Calculate the change in error
50     derror       = error - lastError;
51
52     //Calculate the sum of the error
53     errorSum    += error;
54
55     //Update the state control vars
56     proportional = error      * kp;
57     derivative   = derror/dt  * kd;
58     integral     = errorSum*dt * ki;
59
60     //Calculate the output
61     output       = proportional + integral + derivative;
62
63     lastMillisPID = millis();
64 }
65

```

```
66 //Used for the PID controller
67 double getPIDOutput(void)
68 {
69     return output;
70 }
71
72 //-----DEBUGGING GET VARS-----
73 double getProportionalComponent(void)
74 {
75     return proportional;
76 }
77 double getDerivativeComponent(void)
78 {
79     return derivative;
80 }
81 double getIntegralComponent(void)
82 {
83     return integral;
84 }
85 double getErrorComponent(void)
86 {
87     return error;
88 }
```

```
1  #include "printfUART.h"
2  #include <stdio.h>
3  #include "uartHandler.h"
4
5
6  void _mon_putc(char ch);
7
8  DRV_HANDLE           USART_HANDLER_PRINTF;
9
10 void setupPrintf(DRV_HANDLE uartHandle)
11 {
12     //setbuf(stdout, NULL);
13     USART_HANDLER_PRINTF=uartHandle;
14 }
15
16 void _mon_putc(char ch)
17 {
18     writeByteUART1(ch);
19 }
20
21
22
```

```

1  #include "RCInputs.h"
2  #include "demonstrationSetup.h"
3  #include "driveAssistance.h"
4  #include "macros.h"
5  #include "MillistTimer.h"
6  #include <stdio.h>
7  #include "helperFunctions.h"
8  #include "motorControl.h"
9
10 // #define PRINT_RAW_INPUT_1
11 // #define PRINT_RAW_INPUT_2
12 // #define PRINT_RAW_INPUT_3
13 // #define PRINT_RAW_INPUT_4
14 // #define PRINT_RAW_INPUT_5
15 // #define PRINT_RAW_INPUT_6
16
17 uint8_t operationalMode=NORMAL_MODE;
18
19 uint16_t IC1CaptureEvents[IC_FIFO_LENGTH],IC3CaptureEvents[IC_FIFO_LENGTH];
20 DRV_HANDLE IC1Handle, IC3Handle, TIMERHandle;
21
22 timers_t RCTimeout;
23
24 int channel1Value=-253, channel2Value, channel3Value, channel4Value,
channel5Value=-255, channel6Value=-255;
25 int lastRB11State, lastRB9State, lastRB12State, lastRB13State;
26
27 //Manual Input Capture
28 unsigned int channel12Start, channel12Raw;
29 unsigned int channel13Start, channel13Raw;
30 unsigned int channel14Start, channel14Raw;
31 unsigned int channel16Start, channel16Raw;
32
33 bool channel5DigitalRead,channel1DigitalRead;
34
35 bool lockSystemOutputs=false;
36
37 int processChannel2(void);
38 int processChannel3(void);
39 int processChannel4(void);
40 int processChannel6(void);
41
42 timers_t ICTimer;
43
44 void initRCInputs(void)
45 {
46 }
47 bool updateRCInputs(void)
48 {
49     static bool firstTime=true;
50     if(firstTime)
51     {
52         setTimerInterval(&RCTimeout,100);
53         firstTime=false;
54         setTimerInterval(&ICTimer,3);
55     }
56
57     if(timerDone(&ICTimer))
58     {
59         //printf("Capture\r\n");
60         inputCapture1PWMCalculator();
61         inputCapture3PWMCalculator();
62         processChannel2();
63         processChannel3();
64         processChannel4();

```

```

65         processChannel6();
66     //      if(isAboutInt(channel1Value,-253,2)
67     //          && isAboutInt(channel2Value,0,2)
68     //          && isAboutInt(channel3Value,0,2)
69     //          && isAboutInt(channel4Value,0,2)
70     //          && isAboutInt(channel5Value,-255,2)
71     //          &&
72     //          isAboutInt(channel6Value,-255,2))//isAboutInt(channel1Value,-78,2) &&
73     //          isAboutInt(channel2Value,0,1) && isAboutInt(channel3Value,-254,2) &&
74     //          isAboutInt(channel5Value,0,1))
75     //      {
76     //          if(timerDone(&RCTimeout))
77     //          {
78     //              //printf("Controller Off\r\n");
79     //              lockSystemOutputs=true;
80
81             channel1Value=-254;
82             channel2Value=0;
83             channel4Value=0;
84             channel5Value=-255;
85             channel6Value=-255;
86             channel12Raw=0;
87             channel14Raw=0;
88             channel16Raw=0;
89             setLeftMotorSpeed(0);
90             setRightMotorSpeed(0);
91             setWeaponMotorSpeed(0);
92             setMacroCommand(0);
93             return false;
94         }
95     else
96     {
97         //FLAPS
98         if(channel5Value>DIGITAL_HIGH_INDICATION && !channel5DigitalRead)
99         {
100            channel5DigitalRead = true;
101            //WHEN THE PIN CHANGES TO HIGH
102            setMacroCommand(AUTONOMOUS);
103        }
104        else if(channel5Value<DIGITAL_LOW_INDICATION && channel5DigitalRead)
105        {
106            //WHEN THE PIN CHANGES TO HIGH
107            channel5DigitalRead = false;
108            setMacroCommand(MACRO_STOP);
109        }
110
111         //GEAR
112         if(channel1Value>DIGITAL_HIGH_INDICATION && !channel1DigitalRead)
113         {
114            //WHEN THE PIN CHANGES TO HIGH
115            channel1DigitalRead = true;
116            getGyroStabilizationActive(true);
117        }
118        else if(channel1Value<DIGITAL_LOW_INDICATION && channel1DigitalRead)
119        {
120            //WHEN THE PIN CHANGES TO HIGH
121            channel1DigitalRead = false;
122            getGyroStabilizationActive(false);
123        }
124    else
125    {
126        return false;

```

```

127     }
128 }
129 //RB14 is IC1 (RC_PWM_CH1)
130 int inputCapture1PWMCalculator(void)
131 {
132     int i=0;
133     while(!DRV_IC0_BufferIsEmpty())
134     {
135         IC1CaptureEvents[i]=DRV_IC0_Capture16BitDataRead();
136         i++;
137     }
138     int k=0;
139     for(k=1;k<i;k++)
140     {
141         int difference = (IC1CaptureEvents[k]-IC1CaptureEvents[k-1]);
142         //printf("%d\n",difference);
143         if(isWithinInt(difference,MIN_LENGTH_CHANNEL1,MAX_LENGTH_CHANNEL1))
144         {
145             channel1Value=difference;
146
147             #ifdef PRINT_RAW_INPUT_1
148             printf("Ch1: %d\r\n",channel1Value);
149             #endif
150             channel1Value-=CHANNEL1_OFFSET;
151             channel1Value*=CHANNEL1_SCALE;
152             if(isAboutInt(channel1Value,0,CHANNEL1_DEADBAND) )
153             {
154                 channel1Value=0;
155             }
156             else
157             {
158
159             }
160             if(channel1Value>255)
161             {
162                 channel1Value=255;
163             }
164             if(channel1Value<-255)
165             {
166                 channel1Value=-255;
167             }
168             //printf("Ch1: %d\r\n",channel1Value);
169             return channel1Value;
170         }
171     }
172     return channel1Value;
173 }
174
175 //RB10 is IC3 (RC_PWM_CH5)
176 int inputCapture3PWMCalculator(void)
177 {
178     int i=0;
179     while(!DRV_IC1_BufferIsEmpty())
180     {
181         IC3CaptureEvents[i]=DRV_IC1_Capture16BitDataRead();
182         i++;
183     }
184     int k=0;
185     for(k=1;k<i;k++)
186     {
187         int difference = (IC3CaptureEvents[k]-IC3CaptureEvents[k-1]);
188         //printf("%d\n",difference);
189         if(isWithinInt(difference,MIN_LENGTH_CHANNELS5,MAX_LENGTH_CHANNELS5))
190         {
191             channel5Value=difference;

```

```

192     #ifdef PRINT_RAW_INPUT_5
193     printf("Ch5: %d\r\n",channel5Value);
194     #endif
195     channel5Value-=CHANNELS5_OFFSET;
196     channel5Value*=CHANNELS5_SCALE;
197     if(isAboutInt(channel5Value,0,CHANNELS5_DEADBAND) )
198     {
199         channel5Value=0;
200     }
201     else
202     {
203     }
204     if(channel5Value>255)
205     {
206         channel5Value=255;
207     }
208     if(channel5Value<-255)
209     {
210         channel5Value=-255;
211     }
212     //printf("Ch3: %d\r\n",channel3Value);
213     return channel5Value;
214 }
215 }
216 return channel5Value;
217 }
218 }
219
220
221
222 //24-RB11 is PORTB ChangeNotification (PWM Input) (RC_PWM_CH4)
223 //22-RB9  is PORTB ChangeNotification (PWM Input) (RC_PWM_CH6)
224 //27-RB12 is PORTB ChangeNotification (PWM Input) (RC_PWM_CH3)
225 //28-RB13 is PORTB ChangeNotification (PWM Input) (RC_PWM_CH2)
226
227 void pinChangeNotificationCallback(void)
228 {
229     if(getApplicationState()==APP_STATE_SERVICE_TASKS)
230     {
231         //CHANNEL 4
232         if(PORTBbits.RB11 != lastRB11State)
233         {
234             if(PORTBbits.RB11)
235             {
236                 //WHEN THE PIN CHANGES TO HIGH (Mark Timer Value)
237                 channel4Start = PLIB_TMR_Counter16BitGet(TMR_ID_2);
238             }
239             else
240             {
241                 //WHEN THE PIN CHANGES TO LOW (Calculate Timer Duty)
242                 channel4Raw =
243                 PLIB_TMR_Counter16BitGet(TMR_ID_2)-channel4Start;
244
245             }
246             lastRB11State = PORTBbits.RB11;
247         }
248         //CHANNEL 6
249         if(PORTBbits.RB9 != lastRB9State)
250         {
251             if(PORTBbits.RB9)
252             {
253                 //WHEN THE PIN CHANGES TO HIGH (Mark Timer Value)
254                 channel6Start = PLIB_TMR_Counter16BitGet(TMR_ID_2);
255             }
256             else

```

```

255     {
256         //WHEN THE PIN CHANGES TO LOW (Calculate Timer Duty)
257         channel6Raw =
258             PLIB_TMR_Counter16BitGet(TMR_ID_2)-channel6Start;
259
260     }
261
262     //CHANNEL 3
263     if(PORTBbits.RB12 != lastRB12State)
264     {
265         if(PORTBbits.RB12)
266         {
267             //WHEN THE PIN CHANGES TO HIGH (Mark Timer Value)
268             channel3Start = PLIB_TMR_Counter16BitGet(TMR_ID_2);
269         }
270         else
271         {
272             //WHEN THE PIN CHANGES TO LOW (Calculate Timer Duty)
273             channel3Raw =
274                 PLIB_TMR_Counter16BitGet(TMR_ID_2)-channel3Start;
275         }
276         lastRB12State = PORTBbits.RB12;
277     }
278
279     //CHANNEL 2
280     if(PORTBbits.RB13 != lastRB13State)
281     {
282         resetTimer(&RCTimeout);
283         if(PORTBbits.RB13)
284         {
285             //WHEN THE PIN CHANGES TO HIGH (Mark Timer Value)
286             channel2Start =
287                 PLIB_TMR_Counter16BitGet(TMR_ID_2);
288         }
289         else
290         {
291             //WHEN THE PIN CHANGES TO LOW (Calculate Timer
292             //Duty)
293             channel2Raw =
294                 PLIB_TMR_Counter16BitGet(TMR_ID_2)-channel2Start;
295         }
296     }
297     int processChannel2(void)
298     {
299         //printf("Ch2: %d\n",channel2Raw);
300         if(isWithinInt(channel2Raw,MIN_LENGTH_CHANNEL2,MAX_LENGTH_CHANNEL2))
301         {
302             channel2Value=channel2Raw;
303             #ifdef PRINT_RAW_INPUT_2
304             printf("Ch2: %d\r\n",channel2Raw);
305             #endif
306             channel2Value-=CHANNEL2_OFFSET;
307             channel2Value*=CHANNEL2_SCALE;
308             if(isAboutInt(channel2Value,0,CHANNEL2_DEADBAND) )
309             {
310                 channel2Value=0;
311             }
312             else
313             {

```

```

314         //channel2Value=channel2Raw;
315     }
316
317     if(channel2Value>255)
318     {
319         channel2Value=255;
320     }
321     if(channel2Value<-255)
322     {
323         channel2Value=-255;
324     }
325     //printf("Ch2: %d\r\n",channel2Value);
326     return channel2Value;
327 }
328 return channel2Value;
}
329
330 int processChannel3(void)
331 {
332     //printf("Ch3: %d\r\n",channel3Raw);
333     if(isWithinInt(channel3Raw,MIN_LENGTH_CHANNEL3,MAX_LENGTH_CHANNEL3))
334     {
335         channel3Value = channel3Raw;
336         #ifdef PRINT_RAW_INPUT_3
337         printf("Ch3: %d\r\n",channel3Raw);
338         #endif
339         channel3Value-=CHANNEL3_OFFSET;
340         channel3Value*=CHANNEL3_SCALE;
341         if(isAboutInt(channel3Value,0,CHANNEL3_DEADBAND) )
342         {
343             channel3Value=0;
344         }
345         else
346         {
347
348         }
349
350         if(channel3Value>255)
351         {
352             channel3Value=255;
353         }
354         if(channel3Value<-255)
355         {
356             channel3Value=-255;
357         }
358         //printf("Ch3: %d\r\n",channel3Value);
359         return channel3Value;
360     }
361     return channel3Value;
362 }
363
364
365
366 int processChannel4(void)
367 {
368     //printf("Ch4: %d\r\n",channel4Raw);
369     if(isWithinInt(channel4Raw,MIN_LENGTH_CHANNEL4,MAX_LENGTH_CHANNEL4))
370     {
371         channel4Value = channel4Raw;
372         #ifdef PRINT_RAW_INPUT_4
373         printf("Ch4: %d\r\n",channel4Raw);
374         #endif
375         channel4Value-=CHANNEL4_OFFSET;
376         channel4Value*=CHANNEL4_SCALE;
377         if(isAboutInt(channel4Value,0,CHANNEL4_DEADBAND) )
378         {

```

```

379         channel4Value=0;
380     }
381     else
382     {
383     }
384     if(channel4Value>255)
385     {
386         channel4Value=255;
387     }
388     if(channel4Value<-255)
389     {
390         channel4Value=-255;
391     }
392     //printf("Ch4: %d\r\n",channel4Value);
393     return channel4Value;
394 }
395     return channel4Value;
396 }
397
398
399
400
401 int processChannel6(void)
402 {
403     //printf("Ch6: %d\r\n",channel6Raw);
404     if(isWithinInt(channel6Raw,MIN_LENGTH_CHANNEL6,MAX_LENGTH_CHANNEL6))
405     {
406         channel6Value = channel6Raw;
407         #ifdef PRINT_RAW_INPUT_6
408         printf("Ch6: %d\r\n",channel6Raw);
409         #endif
410         channel6Value-=CHANNEL6_OFFSET;
411         channel6Value*=CHANNEL6_SCALE;
412         if(isAboutInt(channel6Value,0,CHANNEL6_DEADBAND) )
413         {
414             channel6Value=0;
415         }
416         else
417         {
418         }
419     }
420     if(channel6Value>255)
421     {
422         channel6Value=255;
423     }
424     if(channel6Value<-255)
425     {
426         channel6Value=-255;
427     }
428     //printf("Ch6: %d\r\n",channel6Value);
429     return channel6Value;
430 }
431     return channel6Value;
432 }
433
434
435 uint8_t getOperationalMode(void)
436 {
437     return operationalMode;
438 }
439
440 void setupIC1Handle(DRV_HANDLE handle)
441 {
442     IC1Handle = handle;
443 }

```

```
444
445 void setupIC3Handle(DRV_HANDLE handle)
446 {
447     IC3Handle = handle;
448 }
449
450 void setupTimerHandleRC()//DRV_HANDLE handle)
451 {
452     //TIMERHandle = handle;
453 }
454
455 int getChannel1(void)
456 {
457     return channel1Value;
458 }
459
460 int getChannel2(void)
461 {
462     return channel2Value;
463 }
464
465 int getChannel3(void)
466 {
467     return channel3Value;
468 }
469
470 int getChannel4(void)
471 {
472     return channel4Value;
473 }
474
475 int getChannel5(void)
476 {
477     return channel5Value;
478 }
479 int getChannel6(void)
480 {
481     return channel6Value;
482 }
```

```

1  #include "runBackAndForthFindEnemyBoth.h"
2  #include "helperFunctions.h"
3  #include "IMU.h"
4  #include "communications.h"
5  #include "PID.h"
6  #include "motorControl.h"
7  #include "FastTransfer.h"
8  #include "driveAssistance.h"
9  #include "macros.h"
10 #include "RCInputs.h"
11 #include "LIDARSensors.h"
12 #include "MillisTimer.h"
13 #include "doTurn.h"
14
15 #define DEBUG_PID_UART
16
17 //-----BACK AND FORTH-----
18 #define FORWARD           true
19 #define BACKWARD          false
20 #define ENEMY_LOCATION_SENSATIVITY 25
21 #define BACK_AND_FORTH_SPEED    20
22 #define MAX_SPEED_BAF        25
23 #define MIN_SPEED_BAF       -25
24
25 timers_t debugTHIS;
26 timers_t decisionTiming, sampleTiming;
27
28 int boundMotorCommandsBAF(int val);
29
30 void runBackAndForth(void)
31 {
32     //Prep data values
33     double angleOffset=0;
34     bool direction = FORWARD;
35
36     //Prepare a timer for making control decisions and sampling
37     setTimerInterval(&decisionTiming,50);
38     setTimerInterval(&sampleTiming,5);
39     setTimerInterval(&debugTHIS,100);
40
41     resetXAngle();      //Relative turning (zero before we turn)
42
43     setGyroRelativeAngle();
44
45     printf("BAF\r\n");
46     //Do method until forwards sensor is at wall, left sensor is at wall or told
47     //to stop
48     while(getSensorDistance(FORWARD_SENSOR)>25 && getSensorDistance(2)>25 &&
49         (getMacroCommand() !=0))
50     {
51         //Update the gyroscope values
52         updateIMU();
53
54         //Update the communications systems (Motors/Sensors)
55         updateCommunications(true);
56
57 #ifndef IGNORE_MACRO_KILL
58         //Update the RC input to cancel
59         updateRCInputs();
60 #endif
61
62         //Allow the PID to output a control value
63         if(timerDone(&decisionTiming))
64         {
65             int _lm;

```

```

64     int _rm;
65     if(direction==FORWARD)
66     {
67         _lm =  20;
68         _rm =  20;
69     }
70     else
71     {
72         _lm = -20;
73         _rm = -20;
74     }
75
76     //Using the two 45 degree sensors for wall orientation
77     if(getSensorDistance(1)<getSensorDistance(3)-25)
78     {
79         if(angleOffset<10)
80         {
81             angleOffset++;
82             setGyroRelativeAngleValue(getGyroRelativeAngleValue()+1);
83         }
84     }
85     else if(getSensorDistance(1)>getSensorDistance(3)+25)
86     {
87         if(angleOffset>-10)
88         {
89             setGyroRelativeAngleValue(getGyroRelativeAngleValue()-1);
90             angleOffset--;
91         }
92     }
93     else
94     {
95         angleOffset=0;
96         setGyroRelativeAngle();
97     }
98
99
100
101    if(((getSensorDistance(FORWARD_SENSOR)<75)&&(direction==FORWARD))|||((getSensorDistance(BACKWARD_SENSOR)<75)&&(direction==BACKWARD)))
102    {
103        //ToSend(SPEED_SETTING_DATA_ADDRESS,0);
104        //sendData(MDB_L_ADDRESS);
105        //ToSend(SPEED_SETTING_DATA_ADDRESS,0);
106        //sendData(MDB_R_ADDRESS);
107
108        //Reset the offset for getting yourself straight
109        angleOffset=0;
110        //Reverse Direction
111        direction = !direction;
112        //Set the relative angle
113        resetXAngle();
114        setGyroRelativeAngle();
115
116        //Correct the motor values using the IMU
117        motorGyroFeedbackStraight(&_lm, &_rm);
118
119        //Send to the motor system to be sent by the communications library
120        setLeftMotorSpeed(boundMotorCommands(boundMotorCommandsBAF(_lm)));
121        setRightMotorSpeed(boundMotorCommands(boundMotorCommandsBAF(_rm)));
122
123 #ifdef DEBUG_PID_UART
124     if(timerDone(&debugTHIS))
125     {
126         if(direction==FORWARD)

```

```

127         printf("Moving forward:
128             %f\r\n",getSensorDistance(FORWARD_SENSOR));
129     else
130         printf("Moving backward:
131             %f\r\n",getSensorDistance(BACKWARD_SENSOR));
132     }
133 }
134
135     completeMacro();
136 }
137
138 void runBackAndForthFindEnemy(void)
139 {
140     //Prep data values
141     double angleOffset=0;
142     double deltaSensor=0, lastSensor = getSensorDistance(6);
143     bool direction = FORWARD;
144     bool foundEnemy = false;
145     //Prepare a timer for making control decisions and sampling
146     setTimerInterval(&decisionTiming,50);
147     setTimerInterval(&sampleTiming,5);
148     setTimerInterval(&debugTHIS,100);
149     resetXAngle();      //Relative turning (zero before we turn)
150
151     setGyroRelativeAngle();
152     //resetPIDController(angle, KP_TURN, KI_TURN, KD_TURN);
153
154     //Do method until forwards sensor is at wall, left sensor is at wall or told
155     //to stop
156     while(!foundEnemy && (getMacroCommand() !=0))//&& getSensorDistance(0)>15 &&
157     getSensorDistance(2)>15
158     {
159         //Update the gyroscope values
160         updateIMU();
161
162         //Update the communications systems (Motors/Sensors)
163         updateCommunications(true);
164
165     #ifndef IGNORE_MACRO_KILL
166         //Update the RC input to cancel
167         updateRCInputs();
168     #endif
169
170         //Check to see if the change in sensor value is beyond the threshold,
171         //meaning we saw a non steady slope (not a wall)
172         if(timerDone(&sampleTiming))
173         {
174             deltaSensor = lastSensor - getSensorDistance(6);
175             if(abs(deltaSensor)>ENEMY_LOCATION_SENSATIVITY)
176             {
177                 doTurn(90, getWeaponMotorSpeed());
178                 foundEnemy = true;
179             }
180         }
181         //Allow the PID to output a control value
182         if(timerDone(&decisionTiming))
183         {
184             int _lm;
185             int _rm;
186             if(direction==FORWARD)
187             {
188                 _lm = BACK_AND_FORTH_SPEED;
189                 _rm = BACK_AND_FORTH_SPEED;

```

```

188     }
189     else
190     {
191         _lm = -BACK_AND_FORTH_SPEED;
192         _rm = -BACK_AND_FORTH_SPEED;
193     }
194
195
196     //Using the two 45 degree sensors for wall orientation
197     if(getSensorDistance(1)<getSensorDistance(3)-25)
198     {
199         if(angleOffset<10)
200         {
201             angleOffset++;
202             setGyroRelativeAngleValue(getGyroRelativeAngleValue()+1);
203         }
204     }
205     else if(getSensorDistance(1)>getSensorDistance(3)+25)
206     {
207         if(angleOffset>-10)
208         {
209             setGyroRelativeAngleValue(getGyroRelativeAngleValue()-1);
210             angleOffset--;
211         }
212     }
213     else
214     {
215     //           angleOffset=0;
216     //           setGyroRelativeAngle();
217     }
218
219
220
221     if((((getSensorDistance(0)<30)&&(direction==FORWARD))||((getSensorDistance(4)<30)&&(direction==BACKWARD)))
222     {
223         ToSend(SPEED_SETTING_DATA_ADDRESS,0);
224         sendData(MDB_L_ADDRESS);
225         ToSend(SPEED_SETTING_DATA_ADDRESS,0);
226         sendData(MDB_R_ADDRESS);
227
228         //Reset the offset for getting yourself straight
229         angleOffset=0;
230         //Reverse Direction
231         direction = !direction;
232         //Set the relative angle
233         resetXAngle();
234         setGyroRelativeAngle();
235
236         //Correct the motor values using the IMU
237         motorGyroFeedbackStraight(&_lm, &_rm);
238
239         //Send to the motor system to be sent by the communications library
240         setLeftMotorSpeed(boundMotorCommands(boundMotorCommandsBAF(_lm)));
241         setRightMotorSpeed(boundMotorCommands(boundMotorCommandsBAF(_rm)));
242     }
243
244     #ifdef DEBUG_PID_UART
245         if(timerDone(&debugTHIS))
246         {
247     }
248
249 }
250

```

```

251     completeMacro();
252 }
253
254
255 void runBackAndForthFindEnemyBoth(void)
256 {
257     //Prep data values
258     double angleOffset=0;
259
260     //SENSOR 2
261     double deltaSensor1=0, lastSensor1 = getSensorDistance(6);
262     //SENSOR 6
263     double deltaSensor2=0, lastSensor2 = getSensorDistance(2);
264
265
266     bool direction = FORWARD;
267     bool foundEnemy = false;
268     //Prepare a timer for making control decisions and sampling
269     setTimerInterval(&decisionTiming,50);
270     setTimerInterval(&sampleTiming,5);
271     setTimerInterval(&debugTHIS,100);
272     resetXAngle();      //Relative turning (zero before we turn)
273
274     setGyroRelativeAngle();
275     //resetPIDController(angle, KP_TURN, KI_TURN, KD_TURN);
276     printf("Delta1: %f\r\n", deltaSensor1);
277     printf("Delta2: %f\r\n", deltaSensor2);
278     printf("Last1: %f\r\n", lastSensor1);
279     printf("Last2: %f\r\n", lastSensor2);
280     printf("This1: %f\r\n", getSensorDistance(6));
281     printf("This2: %f\r\n", getSensorDistance(2));
282     //Do method until forwards sensor is at wall, left sensor is at wall or told
283     //to stop
284     while(!foundEnemy && (getMacroCommand() !=0))//&& getSensorDistance(0)>15 &&
285     getSensorDistance(2)>15
286     {
287         //Update the gyroscope values
288         updateIMU();
289
290         //Update the communications systems (Motors/Sensors)
291         updateCommunications(true);
292
293 #ifndef IGNORE_MACRO_KILL
294         //Update the RC input to cancel
295         updateRCInputs();
296 #endif
297
298         //Check to see if the change in sensor value is beyond the threshold,
299         //meaning we saw a non steady slope (not a wall)
300         if(timerDone(&sampleTiming))
301         {
302             deltaSensor1    = lastSensor1 - getSensorDistance(6);
303             deltaSensor2    = lastSensor2 - getSensorDistance(2);
304             lastSensor1    = getSensorDistance(6);
305             lastSensor2    = getSensorDistance(2);
306             if(abs(deltaSensor1)>ENEMY_LOCATION_SENSATIVITY)
307             {
308                 //printf("Delta1: %f\r\n", deltaSensor1);
309                 doTurn(95, getWeaponMotorSpeed());
310                 //                     if(isabout(getSensorDistance(6),lastSensor1,15))
311                 //                     {
312                 //                         return true;
313                 //                     }

```

```

314         foundEnemy = true;
315     }
316
317     if(abs(deltaSensor2)>ENEMY_LOCATION_SENSATIVITY)
318     {
319
320         //printf("Delta2: %f\r\n", deltaSensor2);
321         doTurn(-95, getWeaponMotorSpeed());
322         foundEnemy = true;
323     }
324
325     if((getSensorDistance(1)+getSensorDistance(3))<
326     (getSensorDistance(7)+getSensorDistance(5)))
327     {
328
329         //Using the two 45 degree sensors for wall orientation
330         if(getSensorDistance(1)<getSensorDistance(3)+15)
331         {
332             if(angleOffset<10)
333             {
334                 angleOffset++;
335                 setGyroRelativeAngleValue(getGyroRelativeAngleValue()+1);
336             }
337         }
338         else if(getSensorDistance(1)>getSensorDistance(3)-15)
339         {
340             if(angleOffset>-10)
341             {
342                 setGyroRelativeAngleValue(getGyroRelativeAngleValue()-1);
343                 angleOffset--;
344             }
345         }
346         //angleOffset=0;
347         //setGyroRelativeAngle();
348     }
349
350     else
351     {
352
353         //Using the two 45 degree sensors for wall orientation
354         if(getSensorDistance(5)<getSensorDistance(7)+15)
355         {
356             if(angleOffset>-10)
357             {
358                 angleOffset++;
359                 setGyroRelativeAngleValue(getGyroRelativeAngleValue()-1);
360             }
361         }
362         else if(getSensorDistance(5)>getSensorDistance(7)-15)
363         {
364             if(angleOffset<10)
365             {
366                 setGyroRelativeAngleValue(getGyroRelativeAngleValue()+1);
367                 angleOffset--;
368             }
369         }
370     }
371
372     //angleOffset=0;
373     //setGyroRelativeAngle();
374
375 }
376
377 //Allow the PID to output a control value
if(timerDone(&decisionTiming))

```

```

378 {
379     int _lm;
380     int _rm;
381     if(direction==FORWARD)
382     {
383         _lm = 20;
384         _rm = 20;
385     }
386     else
387     {
388         _lm = -20;
389         _rm = -20;
390     }
391
392 //    //Using ONE sensor facing the wall for orientation (BAD)
393 //    if(getSensorDistance(2)<50)
394 //    {
395 //        if(angleOffset<10)
396 //        {
397 //            angleOffset++;
398 //            setGyroRelativeAngleValue(getGyroRelativeAngleValue()+1);
399 //        }
400 //    }
401 //    else if(getSensorDistance(2)>100)
402 //    {
403 //        if(angleOffset>-10)
404 //        {
405 //            setGyroRelativeAngleValue(getGyroRelativeAngleValue()-1);
406 //            angleOffset--;
407 //        }
408 //    }
409 //    else
410 //    {
411 //        angleOffset=0;
412 //        setGyroRelativeAngle();
413 //    }
414
415
416
417
418
419     if(((getSensorDistance(0)<65)&&(direction==FORWARD))||((getSensorDistance(4)<65)&&(direction==BACKWARD)))
420     {
421         ToSend(SPEED_SETTING_DATA_ADDRESS,0);
422         sendData(MDB_L_ADDRESS);
423         ToSend(SPEED_SETTING_DATA_ADDRESS,0);
424         sendData(MDB_R_ADDRESS);
425
426         //Reset the offset for getting yourself straight
427         angleOffset=0;
428         //Reverse Direction
429         direction = !direction;
430         //Set the relative angle
431         resetXAngle();
432         setGyroRelativeAngle();
433
434         //Correct the motor values using the IMU
435         motorGyroFeedbackStraight(&_lm, &_rm);
436
437         //Send to the motor system to be sent by the communications library
438         setLeftMotorSpeed(boundMotorCommands(boundMotorCommandsBAF(_lm)));
439         setRightMotorSpeed(boundMotorCommands(boundMotorCommandsBAF(_rm)));
440     }
}

```

```

441  #ifdef DEBUG_PID_UART
442      if(timerDone(&debugTHIS))
443      {
444          printf("Delta1: %f\r\n", deltaSensor1);
445          printf("Delta2: %f\r\n", deltaSensor2);
446          printf("Last1: %f\r\n", lastSensor1);
447          printf("Last2: %f\r\n", lastSensor2);
448          printf("This1: %f\r\n", getSensorDistance(6));
449          printf("This2: %f\r\n", getSensorDistance(2));
450          // printf("This2: %f\r\n", getSensorDistance(2));
451      }
452  #endif
453
454
455
456
457
458     completeMacro();
459 }
460
461 int boundMotorCommandsBAF(int val)
462 {
463     if(val>MAX_SPEED_BAF)
464     {
465         val=MAX_SPEED_BAF;
466     }
467     else if(val<MIN_SPEED_BAF)
468     {
469         val=MIN_SPEED_BAF;
470     }
471     return val;
472 }
```

```

1  #include "runWalls.h"
2  #include "helperFunctions.h"
3  #include "IMU.h"
4  #include "communications.h"
5  #include "PID.h"
6  #include "motorControl.h"
7  #include "FastTransfer.h"
8  #include "driveAssistance.h"
9  #include "macros.h"
10 #include "RCInputs.h"
11 #include "LIDARSensors.h"
12 #include "MillisTimer.h"
13 #include "spinWeaponUp.h"
14 #include "doTurn.h"
15
16 #define RUN_WALL_SPEED          40
17 #define MAX_OFFSET_SPEED        9
18 #define FORWARD_WALL_DISTANCE   150
19
20 #define TURN_DEGREES_90          120
21 #define TURN_DEGREES_45          50
22
23 #define ENEMY_LOCATION_SENSATIVITY 45
24 #define MAX_SPEED_RUN_WALLS      35
25 #define MIN_SPEED_RUN_WALLS      -35
26
27 //#define DEBUG
28
29 //##define DEBUG_PID_UART
30 int _lm =0;
31 int _rm =0;
32 timers_t debugTHIS;
33 timers_t decisionTiming, sampleTiming, correctionTimer;
34
35 int boundMotorCommandsRunWalls(int val);
36
37 void runWalls(void)
38 {
39     int offsetSpeed=0;
40     double deltaSensor=0, lastSensor = getSensorDistance(2);
41
42     //Prepare a timer for making control decisions and sampling
43     setTimerInterval(&decisionTiming,50);
44     setTimerInterval(&sampleTiming,5);
45     setTimerInterval(&debugTHIS,100);
46     setTimerInterval(&correctionTimer,100);
47     resetXAngle();           //Relative turning (zero before we turn)
48
49     setGyroRelativeAngle();
50
51     spinWeaponUp(15);
52
53     //Do method until forwards sensor is at wall, left sensor is at wall or told
54     //to stop
55     while(getSensorDistance(0)>40 && (getMacroCommand() !=0))
56     {
57         //Update the gyroscope values
58         updateIMU();
59
59         //Update the communications systems (Motors/Sensors)
60         updateCommunications(true);
61
62     #ifndef IGNORE_MACRO_KILL
63         //Update the RC input to cancel
64         updateRCInputs();

```

```

65  #endif
66  if(timerDone(&correctionTimer))
67  {
68  //      //Using the two 45 degree sensors for wall orientation
69  //      if(getSensorDistance(1)<getSensorDistance(3)-15)
70  //
71  //          if(angleOffset<10)
72  //          {
73  //              angleOffset++;
74  //              setGyroRelativeAngleValue(getGyroRelativeAngleValue()+1);
75  //          }
76  //
77  //      else if(getSensorDistance(1)>getSensorDistance(3)+15)
78  //      {
79  //          if(angleOffset>-10)
80  //          {
81  //              setGyroRelativeAngleValue(getGyroRelativeAngleValue()-1);
82  //              angleOffset--;
83  //          }
84  //      }
85  //      else
86  //
87  //          angleOffset=0;
88  //          //setGyroRelativeAngle();
89  //
90
91  deltaSensor = lastSensor-getSensorDistance(2);
92  lastSensor = getSensorDistance(2);
93  offsetSpeed=deltaSensor*1.25;
94  if(offsetSpeed>=MAX_OFFSET_SPEED)
95  {
96      offsetSpeed=MAX_OFFSET_SPEED;
97
98
99  if(offsetSpeed<=-MAX_OFFSET_SPEED)
100 {
101     offsetSpeed=-MAX_OFFSET_SPEED;
102
103 //     if(deltaSensor>0)
104 //
105 //
106 //         if(angleOffset<5)
107 //         {
108 //             //setGyroRelativeAngleValue(getGyroRelativeAngleValue()+1);
109 //             angleOffset++;
110 //             offsetSpeed++;
111 //         }
112 //     }
113 //     else
114 //
115 //
116 //         if(angleOffset>-5)
117 //         {
118 //             angleOffset--;
119 //             //setGyroRelativeAngleValue(getGyroRelativeAngleValue()-1);
120 //             offsetSpeed--;
121 //         }
122 //     }
123
124 //Allow the PID to output a control value
125 if(timerDone(&decisionTiming))
126 {
127     lm = RUN_WALL_SPEED;
128     rm = RUN_WALL_SPEED;
129

```

```

130      //Using ONE sensor facing the wall for orientation (BAD)
131      //if(getSensorDistance(2)<50)
132      //{
133      //    if(angleOffset<10)
134      //{
135      //        angleOffset++;
136      //        setGyroRelativeAngleValue(getGyroRelativeAngleValue()+1);
137      //}
138      //}
139      //else if(getSensorDistance(2)>100)
140      //{
141      //    if(angleOffset>-10)
142      //{
143      //        setGyroRelativeAngleValue(getGyroRelativeAngleValue()-1);
144      //        angleOffset--;
145      //}
146      //}
147      //else
148      //{
149      //    angleOffset=0;
150      //    setGyroRelativeAngle();
151      //}
152
153
154
155     if(getSensorDistance(0)<FORWARD_WALL_DISTANCE)
156     {
157         ToSend(SPEED_SETTING_DATA_ADDRESS,0);
158         sendData(MDB_L_ADDRESS);
159         ToSend(SPEED_SETTING_DATA_ADDRESS,0);
160         sendData(MDB_R_ADDRESS);
161
162         //Reset the offset for getting yourself straight
163         offsetSpeed=0;
164         //Turn 90 degrees at the corner
165         doTurn(90, getWeaponMotorSpeed());
166         //Set the relative angle
167         resetXAngle();
168         setGyroRelativeAngle();
169     }
170     //Correct the motor values using the IMU
171     //motorGyroFeedbackStraight(&_lm, &_rm);
172     if(isInverted())
173     {
174         if(offsetSpeed<0)
175         {
176             _lm-=offsetSpeed;
177         }
178         else
179         {
180             _rm+=offsetSpeed;
181         }
182     }
183     else
184     {
185         if(offsetSpeed<0)
186         {
187             _rm-=offsetSpeed;
188         }
189         else
190         {
191             _lm+=offsetSpeed;
192         }
193     }

```

```

195         }
196         //Send to the motor system to be sent by the communications library
197         setLeftMotorSpeed(boundMotorCommands(boundMotorCommandsRunWalls(_lm)))
198         ;
199     }
200
201 #ifdef DEBUG_PID_UART
202     if(timerDone(&debugTHIS))
203     {
204         //         printf("lm: %d, rm: %d\r\n", getLeftMotorSpeed(),
205         getRightMotorSpeed());
206         //         printf("deltaSensor: %f\r\n", deltaSensor);
207         //         printf("sensorVal: %f\r\n", getSensorDistance(2));
208         //         printf("offsetSpeed: %d\r\n\r\n", offsetSpeed);
209         printf("forward: %f\r\n", getSensorDistance(0));
210     }
211 #endif
212 }
213
214     completeMacro();
215 }
216
217 timers_t justTurnedDelay;
218 void runWallsFindEnemy(uint8_t weaponSpeed)
219 {
220     int offsetSpeed=0;
221     bool justTurned=true;
222     double deltaSensor=0,           lastSensor = getSensorDistance(2);
223     double deltaEnemySensor = 0,   lastEnemySensor = getSensorDistance(6);
224     //Prep data values
225     bool foundEnemy=false;
226
227     //Prepare a timer for making control decisions and sampling
228     setTimerInterval(&decisionTiming,50);
229     setTimerInterval(&sampleTiming,5);
230     setTimerInterval(&debugTHIS,500);
231     setTimerInterval(&correctionTimer,100);
232     setTimerInterval(&justTurnedDelay,100);
233
234     resetXAngle();      //Relative turning (zero before we turn)
235
236     setGyroRelativeAngle();
237
238     //Do method until forwards sensor is at wall, left sensor is at wall or told
239     //to stop
240     while(!foundEnemy && getSensorDistance(0)>40  && (getMacroCommand() !=0))
241     {
242         //Update the gyroscope values
243         updateIMU();
244
245         //Update the communications systems (Motors/Sensors)
246         updateCommunications(true);
247
248 #ifndef IGNORE_MACRO_KILL
249         //Update the RC input to cancel
250         updateRCInputs();
251 #endif
252         uint8_t speed = (int)((getChannel16()+255)/5.0);
253         if(speed>weaponSpeed)
254         {

```

```

254         if(speed!=getWeaponMotorSpeed())
255             setWeaponMotorSpeed(speed);
256     }
257     else
258     {
259         if(weaponSpeed!=getWeaponMotorSpeed())
260             setWeaponMotorSpeed(weaponSpeed);
261     }
262 #ifdef DEBUG
263     if(timerDone(&debugTHIS))
264     {
265         printf("Weapon Speed: %d\r\n", getWeaponMotorSpeed());
266     }
267 #endif
268
269     if(timerDone(&correctionTimer))
270     {
271         deltaSensor = lastSensor-getSensorDistance(2);
272         lastSensor = getSensorDistance(2);
273         offsetSpeed=deltaSensor*2;
274         if(offsetSpeed>=MAX_OFFSET_SPEED)
275         {
276             offsetSpeed=MAX_OFFSET_SPEED;
277         }
278         if(offsetSpeed<=-MAX_OFFSET_SPEED)
279         {
280             offsetSpeed=-MAX_OFFSET_SPEED;
281         }
282     }
283 //Allow the PID to output a control value
284 if(timerDone(&decisionTiming))
285 {
286     _lm = RUN_WALL_SPEED;
287     _rm = RUN_WALL_SPEED;
288
289     if(justTurned && timerDone(&justTurnedDelay))
290     {
291         justTurned=false;
292         lastEnemySensor = getSensorDistance(6);
293     }
294     else
295     {
296         deltaEnemySensor = lastEnemySensor - getSensorDistance(6);
297         // if(abs(deltaEnemySensor)>ENEMY_LOCATION_SENSATIVITY &&
298         // !justTurned)
299         // {
300         //     doTurn(110);
301         //     foundEnemy = true;
302         // }
303         if(getEnemyLocation()==7)
304         {
305             doTurn(TURN_DEGREES_45, getWeaponMotorSpeed());
306             foundEnemy=true;
307             return;
308         }
309         if(getEnemyLocation()==6)
310         {
311             doTurn(TURN_DEGREES_90, getWeaponMotorSpeed());
312             foundEnemy = true;
313             return;
314         }
315     }
316
317     if(getSensorDistance(0)<FORWARD_WALL_DISTANCE)

```

```

318     {
319         ToSend(SPEED_SETTING_DATA_ADDRESS,0);
320         sendData(MDB_L_ADDRESS);
321         ToSend(SPEED_SETTING_DATA_ADDRESS,0);
322         sendData(MDB_R_ADDRESS);
323
324         //Reset the offset for getting yourself straight
325         offsetSpeed=0;
326         //Turn 90 degrees at the corner
327         doTurn(90, getWeaponMotorSpeed());
328         justTurned=true;
329         //Set the relative angle
330         resetXAngle();
331         setGyroRelativeAngle();
332     }
333     //Correct the motor values using the IMU
334     //motorGyroFeedbackStraight(&_lm, &_rm);
335     if(isInverted())
336     {
337         if(offsetSpeed>0)
338         {
339             _rm+=offsetSpeed;
340         }
341         else
342         {
343             _lm-=offsetSpeed;
344         }
345     }
346     else
347     {
348         if(offsetSpeed>0)
349         {
350             _lm+=offsetSpeed;
351         }
352         else
353         {
354             _rm-=offsetSpeed;
355         }
356     }
357     //Send to the motor system to be sent by the communications library
358
359     setLeftMotorSpeed(boundMotorCommands(boundMotorCommandsRunWalls(_lm)));
360
361     setRightMotorSpeed(boundMotorCommands(boundMotorCommandsRunWalls(_rm)));
362 }
363
364 #ifdef DEBUG_PID_UART
365     if(timerDone(&debugTHIS))
366     {
367         printf("lm: %d, rm: %d\r\n", getLeftMotorSpeed(),
368               getRightMotorSpeed());
369         printf("deltaSensor: %f\r\n", deltaSensor);
370         printf("sensorVal: %f\r\n", getSensorDistance(2));
371         printf("offsetSpeed: %d\r\n\r\n", offsetSpeed);
372     }
373
374     completeMacro();
375 }
376
377 int boundMotorCommandsRunWalls(int val)

```

```
378  {
379      if(val>MAX_SPEED_RUN_WALLS)
380      {
381          val=MAX_SPEED_RUN_WALLS;
382      }
383      else if(val<MIN_SPEED_RUN_WALLS)
384      {
385          val=MIN_SPEED_RUN_WALLS;
386      }
387      return val;
388 }
```

```

1  #include "runWalls.h"
2  #include "helperFunctions.h"
3  #include "IMU.h"
4  #include "communications.h"
5  #include "PID.h"
6  #include "motorControl.h"
7  #include "FastTransfer.h"
8  #include "driveAssistance.h"
9  #include "macros.h"
10 #include "RCInputs.h"
11 #include "LIDARSensors.h"
12 #include "MillisTimer.h"
13 #include "doTurn.h"
14
15 #define RUN_WALL_SPEED 25
16
17 #define MIN_SPEED_FLEE -30
18 #define MAX_SPEED_FLEE 30
19
20 timers_t debugTHIS;
21 timers_t decisionTiming, sampleTiming;
22
23 int boundMotorCommandsFlee(int val);
24
25
26 void runWallsFlee(void)
27 {
28     //Prep data values
29     float angleOffset=0;
30
31     //Prepare a timer for making control decisions and sampling
32     setTimerInterval(&decisionTiming,50);
33     setTimerInterval(&sampleTiming,5);
34     setTimerInterval(&debugTHIS,100);
35     resetXAngle();           //Relative turning (zero before we turn)
36
37     setGyroRelativeAngle();
38
39     //Do method until forwards sensor is at wall, left sensor is at wall or told
40     //to stop
41     while(getSensorDistance(0)>25 && getSensorDistance(2)>25 &&
42         (getMacroCommand() !=0)) //&& !weaponUpToSpeed())
43     {
44         //Update the gyroscope values
45         updateIMU();
46
47         //Update the communications systems (Motors/Sensors)
48         updateCommunications(true);
49
50     #ifndef IGNORE_MACRO_KILL
51         //Update the RC input to cancel
52         updateRCInputs();
53     #endif
54
55         //Allow the PID to output a control value
56         if(timerDone(&decisionTiming))
57         {
58             int _lm =  RUN_WALL_SPEED;
59             int _rm =  RUN_WALL_SPEED;
60
61             //                if(getSensorDistance(2)<50)
62             //                {
63             //                    if(angleOffset<10)
64             //                    {

```

```

64    //                                angleOffset++;
65    //                                setGyroRelativeAngleValue(getGyroRelativeAngleValue() + 1);
66    //
67    }
68    //                                else if(getSensorDistance(2) > 100)
69    {
70    //                                if(angleOffset > -10)
71    //                                {
72    //                                    setGyroRelativeAngleValue(getGyroRelativeAngleValue() - 1);
73    //                                    angleOffset--;
74    //                                }
75    //
76    //                                else
77    //                                {
78    //                                    angleOffset = 0;
79    //                                    setGyroRelativeAngle();
80    //                                }
81
82    //Using the two 45 degree sensors for wall orientation
83    if(getSensorDistance(1) < getSensorDistance(3) - 25)
84    {
85        if(angleOffset < 10)
86        {
87            angleOffset++;
88            setGyroRelativeAngleValue(getGyroRelativeAngleValue() + 1);
89        }
90    }
91    else if(getSensorDistance(1) > getSensorDistance(3) + 25)
92    {
93        if(angleOffset > -10)
94        {
95            setGyroRelativeAngleValue(getGyroRelativeAngleValue() - 1);
96            angleOffset--;
97        }
98    }
99    else
100   {
101    //                                angleOffset = 0;
102    //                                setGyroRelativeAngle();
103   }
104
105   if(getSensorDistance(0) < 75)
106   {
107       ToSend(SPEED_SETTING_DATA_ADDRESS, 0);
108       sendData(MDB_L_ADDRESS);
109       ToSend(SPEED_SETTING_DATA_ADDRESS, 0);
110       sendData(MDB_R_ADDRESS);
111
112       //Reset the offset for getting yourself straight
113       angleOffset = 0;
114       //Turn 90 degrees at the corner
115       doTurn(90, 25);
116       //Set the relative angle
117       setGyroRelativeAngle();
118   }
119   //Correct the motor values using the IMU
120   motorGyroFeedbackStraight(&_lm, &_rm);
121
122   //Send to the motor system to be sent by the communications library
123   setLeftMotorSpeed(boundMotorCommands(boundMotorCommandsFlee(_lm)));
124   setRightMotorSpeed(boundMotorCommands(boundMotorCommandsFlee(_rm)));
125   }
126
127 #ifdef DEBUG_PID_UART
128     if(timerDone(&debugTHIS))

```

```

129         {
130             printf("Doing stuff\r\n");
131     #endif
133     }
135
136     //CLEANUP WHEN WE ARE DONE
137     setLeftMotorSpeed(0);
138     setRightMotorSpeed(0);
139
140     ToSend(SPEED_SETTING_DATA_ADDRESS,0);
141     sendData(MDB_L_ADDRESS);
142     ToSend(SPEED_SETTING_DATA_ADDRESS,0);
143     sendData(MDB_R_ADDRESS);
144 }
145
146
147 int boundMotorCommandsFlee(int val)
148 {
149     if(val>MAX_SPEED_FLEE)
150     {
151         val=MAX_SPEED_FLEE;
152     }
153     else if(val<MIN_SPEED_FLEE)
154     {
155         val=MIN_SPEED_FLEE;
156     }
157     return val;
158 }
```

```

1  /* **** Descriptive File Name **** */
2  /** Descriptive File Name
3
4      @Company
5      Zac Kilburn
6
7      @File Name
8      selectionSwitches.c
9
10     @Summary
11         Brief description of the file.
12
13     @Description
14         Describe the purpose of this file.
15 */
16 /* **** */
17 #include "selectionSwitches.h"
18 #include "I2CFunctions.h"
19 typedef enum{
20     START_BIT_SS = 0,
21     ADDR_BYTE_SS,
22     COMMAND_BYTE_SS,
23     RESTART_BIT_SS,
24     ADDR_BYTE_READ_SS,
25     CLOCK_BYTE_READ_SS,
26     GRAB_BYTE_READ_SS,
27     NACK_BYTE_READ_SS,
28     STOP_BIT_SS
29 }SELECTION_SWITCHES_STATES_t;
30
31 typedef enum{
32     SWITCH1 = 0,
33     SWITCH2,
34     SWITCH3,
35     SWITCH4,
36     SWITCH5,
37     SWITCH6,
38     SWITCH7,
39     SWITCH8,
40     NUMBER_OF_SWITCHES
41 }SELECTION_SWITCHES_t;
42
43 bool
44 switch1State,switch2State,switch3State,switch4State,switch5State,switch6State,switch7State,switch8State;
45
46 uint8_t readSelectionSwitches(void);
47
48 void initSelectionSwitches(void)
49 {
50     uint8_t selectionRead = readSelectionSwitches();
51     switch1State = selectionRead & (1<<SWITCH1);
52     switch2State = selectionRead & (1<<SWITCH2);
53     switch3State = selectionRead & (1<<SWITCH3);
54     switch4State = selectionRead & (1<<SWITCH4);
55     switch5State = selectionRead & (1<<SWITCH5);
56     switch6State = selectionRead & (1<<SWITCH6);
57     switch7State = selectionRead & (1<<SWITCH7);
58     switch8State = selectionRead & (1<<SWITCH8);
59 }
60
61 uint8_t readSelectionSwitches(void)
62 {
63     uint8_t selectionRead=0;

```

```

64     SELECTION_SWITCHES_STATES_t states=START_BIT_SS;
65     bool writeIncomplete=true;
66     //Make sure another thread is not using the I2C
67     if(getMutex()==false)
68     {
69         //Take the mutex before use
70         setMutex(true);
71         while(writeIncomplete)
72         {
73             //Wait for idle bus
74             while (! PLIB_I2C_BusIsIdle ( I2C_ID_1 ) );
75             switch(states)
76             {
77                 case START_BIT_SS:
78                     I2cStart();
79                     states++;
80                     break;
81                 case ADDR_BYTE_SS:
82                     PLIB_I2C_TransmitterByteSend(I2C_ID_1,SELECTION_SWITCHES_ADDR_
83                     WRITE);
84                     while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
85                     states++;
86                     break;
87                 case COMMAND_BYTE_SS:
88                     PLIB_I2C_TransmitterByteSend(I2C_ID_1,SELECTION_SWITCHES_READ_
89                     REQUEST);
90                     while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
91                     states++;
92                     break;
93                 case RESTART_BIT_SS:
94                     PLIB_I2C_MasterStartRepeat(I2C_ID_1);
95                     states++;
96                     break;
97                 case ADDR_BYTE_READ_SS:
98                     PLIB_I2C_TransmitterByteSend(I2C_ID_1,SELECTION_SWITCHES_ADDR_
99                     READ);
100                    while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
101                    states++;
102                    break;
103                    case CLOCK_BYTE_READ_SS:
104                        PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_1);
105                        //while(!PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_1));
106                        if(WaitRxByteAvailable())
107                        {
108                            states++;
109                        }
110                        else
111                        {
112                            resetI2CSysytem();
113                            states=START_BIT_SS;
114                        }
115                        break;
116                    case GRAB_BYTE_READ_SS:
117                        selectionRead=PLIB_I2C_ReceivedByteGet(I2C_ID_1);
118                        states++;
119                        break;
120                    case NACK_BYTE_READ_SS:

```

```

119             PLIB_I2C_ReceivedByteAcknowledge(I2C_ID_1, false);
120
121             //while(!PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_1
122             //));
123             if(WaitReceiverByteAck())
124             {
125                 states++;
126             }
127             else
128             {
129                 resetI2CSYSTEM();
130
131                 states=START_BIT_SS;
132
133             }
134             break;
135         case STOP_BIT_SS:
136             I2cStop();
137             states=START_BIT_SS;
138             writeIncomplete=false;
139             break;
140         }
141     return selectionRead;
142 }
143
144
145 /* **** End of File ****
146 */
147
148

```

```

1
2 #include "sevenSegDigit.h"
3 #include <stdio.h>
4 #include "../../../../../framework/driver/i2c/driv_i2c.h"
5 #include "I2CFunctions.h"
6
7 //===== Method Prototypes =====
8 void writeValueSevenSeg(char num, bool decimalPoint);
9 void I2cStart(void);
10 void I2cStop (void);
11 //===== Variable Declarations =====
12 timers_t blinkyTimer;
13 char lastCharOnScreen;
14 bool decBlinkState=false;
15 timers_t mutexTimer;
16 //===== Send State Machine States =====
17 typedef enum{
18     START_BIT = 0,
19     ADDR_BYTE,
20     COMMAND_BYTE,
21     DATA_BYTE,
22     STOP_BIT
23 }LED_DIGIT_STATES_t;
24 //=====
25 //DRV_HANDLE I2CHandleSevenSeg; //Maybe should have local copy
26
27
28 //=====
29 void initSevenSeg(void)
30 {
31     setTimerInterval(&mutexTimer,1);
32     LED_DIGIT_STATES_t states=START_BIT;
33     bool writeIncomplete=true;
34     //Make sure another thread is not using the I2C
35     if(!getMutex())
36     {
37         //Take the mutex before use
38         setMutex(true);
39         while(writeIncomplete)
40         {
41             //Wait for idle bus
42             while (! PLIB_I2C_BusIsIdle ( I2C_ID_1 ) );
43             switch(states)
44             {
45                 case START_BIT:
46                     I2cStart();
47                     states++;
48                     break;
49                 case ADDR_BYTE:
50                     PLIB_I2C_TransmitterByteSend(I2C_ID_1,LED_DIGIT_ADDR_WRITE);
51                     while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
52                     states++;
53                     break;
54                 case COMMAND_BYTE:
55                     PLIB_I2C_TransmitterByteSend(I2C_ID_1,LED_DIGIT_WRITE_CONFIG);

56                     while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
57                     states++;
58                     break;
59                 case DATA_BYTE:
60                     PLIB_I2C_TransmitterByteSend(I2C_ID_1,LED_DIGIT_ENABLE_OUTPUT)
61                     ;
62                     while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));

```

```

62             states++;
63             break;
64         case STOP_BIT:
65             I2cStop();
66             states=START_BIT;
67             writeIncomplete=false;
68             break;
69         }
70     }
71     //Return the mutex when finished
72     setMutex(false);
73 }
74 //Setup blinky timer for use
75 setTimerInterval(&blinkyTimer,500);
76 }
77 //=====
78 void blinkDecimal(unsigned long ms)
79 {
80     static unsigned long blinkInterval=0;
81     if(ms != blinkInterval)
82     {
83         setTimerInterval(&blinkyTimer,ms);
84         blinkInterval=ms;
85     }
86
87     if(timerDone(&blinkyTimer))
88     {
89         //printf("Blinky\r\n");
90         decimalBlink();
91     }
92 }
93
94 void decimalBlink(void)
95 {
96     if(decBlinkState) //IF DECIMAL POINT WAS ON
97     {
98         //Turn it off
99         writeValueSevenSeg(lastCharOnScreen,false);
100        decBlinkState=false;
101    }
102    else
103    {
104        //Turn it on
105        writeValueSevenSeg(lastCharOnScreen,true);
106        decBlinkState=true;
107    }
108 }
109 //=====
110 void writeValueSevenSeg(char num, bool decimalPoint)
111 {
112     //Make sure another thread is not using the I2C
113     if(!getMutex())
114     {
115         //Take the mutex before use
116         setMutex(true);
117         //Record last char on digit
118         lastCharOnScreen=num;
119         //Setup state machine
120         LED_DIGIT_STATES_t states=START_BIT;
121         //Mark that we need to complete state machine
122         bool writeIncomplete=true;
123
124         //Continue till write is complete
125         while(writeIncomplete)
126         {

```

```

127 //Wait for idle bus
128 while ( ! PLIB_I2C_BusIsIdle ( I2C_ID_1 ) );
129 switch(states)
130 {
131     case START_BIT:
132         I2cStart();
133         states++;
134         break;
135     case ADDR_BYTE:
136         PLIB_I2C_TransmitterByteSend(I2C_ID_1,LED_DIGIT_ADDR_WRITE);
137         while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
138         states++;
139         break;
140     case COMMAND_BYTE:
141         PLIB_I2C_TransmitterByteSend(I2C_ID_1,LED_DIGIT_WRITE_DATA);
142         while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
143         states++;
144         break;
145     case DATA_BYTE:
146         if(decimalPoint)
147             PLIB_I2C_TransmitterByteSend(I2C_ID_1,num & SEG_DP);
148         else
149             PLIB_I2C_TransmitterByteSend(I2C_ID_1,num);

150         while(!PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_1));
151         states++;
152         break;
153     case STOP_BIT:
154         I2cStop();
155         states=START_BIT;
156         writeIncomplete=false;
157         break;
158     }
159 }
160 //Return the mutex when finished
161 setMutex(false);
162 }
163 }
164
165 void writeDigitSevenSegNoDec(char num)
166 {
167     writeDigitSevenSeg(num,decBlinkState);
168 }
169 //=====
170 void writeDigitSevenSeg(char num, bool decimalPoint)
171 {
172     //Determine which number is requested
173     switch(num)
174     {
175         case 0:
176             writeValueSevenSeg(DIGIT0,decimalPoint);
177             break;
178         case 1:
179             writeValueSevenSeg(DIGIT1,decimalPoint);
180             break;
181         case 2:
182             writeValueSevenSeg(DIGIT2,decimalPoint);
183             break;
184         case 3:
185             writeValueSevenSeg(DIGIT3,decimalPoint);
186             break;
187         case 4:
188             writeValueSevenSeg(DIGIT4,decimalPoint);
189             break;

```

```

190     case 5:
191         writeValueSevenSeg(DIGIT5,decimalPoint);
192         break;
193     case 6:
194         writeValueSevenSeg(DIGIT6,decimalPoint);
195         break;
196     case 7:
197         writeValueSevenSeg(DIGIT7,decimalPoint);
198         break;
199     case 8:
200         writeValueSevenSeg(DIGIT8,decimalPoint);
201         break;
202     case 9:
203         writeValueSevenSeg(DIGIT9,decimalPoint);
204         break;
205     case 10: //A
206         writeValueSevenSeg(DIGITA,decimalPoint);
207         break;
208     case 11: //B
209         writeValueSevenSeg(DIGITB,decimalPoint);
210         break;
211     case 12: //C
212         writeValueSevenSeg(DIGITC,decimalPoint);
213         break;
214     case 13: //D
215         writeValueSevenSeg(DIGITD,decimalPoint);
216         break;
217     case 14: //E
218         writeValueSevenSeg(DIGITE,decimalPoint);
219         break;
220     case 15: //F
221         writeValueSevenSeg(DIGITF,decimalPoint);
222         break;
223     }
224 }
225
226 void testSevenSegment(void)
227 {
228
229     static int counter=0;
230     static bool dP = false;
231     //Write the seven segment display
232     writeDigitSevenSeg(counter,dP);
233     if(counter < MAX_DIGITS_SEVEN_SEG)
234     {
235         counter++;
236     }
237     else
238     {
239         counter=0;
240         if(dP)
241         {
242             dP=false;
243         }
244         else
245         {
246             dP=true;
247         }
248     }
249 }
250
251

```

```

1  #include "spinWeaponUp.h"
2  #include "PID.h"
3
4  #include "holdRotationWeaponStopped.h"
5  #include "motorControl.h"
6  #include "MillisTimer.h"
7  #include "driveAssistance.h"
8  #include "macros.h"
9  //##include "IMU.h"
10 #define DEBUG_PID_UART
11 //##define WEAPON_SPEED_SETTING 25
12 #define HOLD_KP 4
13 #define HOLD_KI 0
14 #define HOLD_KD 0.5
15
16 timers_t spinUpTimer;
17 timers_t rampTimer;
18 timers_t debugTHIS;
19 int motorSpeed;
20 bool firstTime=true;
21 void setupSpinUp(void)
22 {
23     setupHoldPosition();
24     setTimerInterval(&rampTimer,100);
25     setTimerInterval(&spinUpTimer,3000);
26     setTimerInterval(&debugTHIS,100);
27 }
28
29 void testMovementSpinning(void)
30 {
31     spinWeaponUp(15);
32
33     setLeftMotorSpeed(20);
34     setRightMotorSpeed(20);
35     resetTimer(&spinUpTimer);
36     while(!timerDone(&spinUpTimer) && (getMacroCommand() !=0))
37     {
38         //Update the gyroscope values
39         updateIMU();
40
41         //Update the communications systems (Motors/Sensors)
42         updateCommunications(true);
43
44         #ifndef IGNORE_MACRO_KILL
45             //Update the RC input to cancel
46             updateRCInputs();
47         #endif
48
49     }
50     completeMacro();
51 }
52
53 void spinWeaponUp(uint8_t weaponSpeed)
54 {
55     //Set angle to 0
56     resetXAngle();
57     setGyroRelativeAngle();
58
59     //Hold angle at 0
60     resetPIDController(0, HOLD_KP, HOLD_KI, HOLD_KD);
61
62     //setWeaponMotorSpeed(WEAPON_SPEED_SETTING);
63     resetTimer(&spinUpTimer);
64     motorSpeed=0;

```

```

66     while(!timerDone(&spinUpTimer) && (getMacroCommand() !=0))
67     {
68         if(timerDone(&rampTimer) && motorSpeed!=weaponSpeed)
69         {
70             motorSpeed++;
71             setWeaponMotorSpeed(motorSpeed);
72             resetTimer(&spinUpTimer);
73         }
74         //Update the gyroscope values
75         updateIMU();
76
77         //Update the communications systems (Motors/Sensors)
78         updateCommunications(true);
79
80         #ifndef IGNORE_MACRO_KILL
81             //Update the RC input to cancel
82             updateRCInputs();
83         #endif
84
85         holdPosition();
86
87         #ifdef DEBUG_PID_UART
88             if(timerDone(&debugTHIS))
89             {
90                 printf("lm: %d, rm: %d\r\n", getLeftMotorSpeed(),
91                     getRightMotorSpeed());
92                 printf("gyro: %f\r\n\r\n", getXAxisAngle());
93             }
94         #endif
95
96     }
97
98     completeMacro();
99
100 }
```

```
1 #include "TimerlessDelay.h"
2 #include <xc.h>
3
4
5 void hardDelay_ms(uint16_t d)
6 {
7     uint32_t i;
8     for(i=0;i<((uint32_t)d*200000);i++)
9     {
10         asm ("nop");
11         WDTCONbits.WDTCLRKEY=0x5743;
12     }
13 }
14
15 void hardDelay_cycles(uint16_t d)
16 {
17     uint32_t i;
18     for(i=0;i<((uint32_t)d);i++)
19     {
20         asm ("nop");
21     }
22 }
23
```

```

1
2 #include "uartHandler.h"
3 #include "app.h"
4
5 volatile uartCircBuff_t recBuf0, sendBuf0, recBuf1,sendBuf1, recBuf2,sendBuf3;
6 DRV_HANDLE
7 internalUART0Handle,internalUART1Handle,internalUART2Handle,internalUART3Handle;
8 volatile bool transmitStall0 = true, transmitStall1 = true, transmitStall3 = true;
9 extern APP_DATA appData;
10
11 void wipeBuffer(volatile unsigned char * buf);
12 void bufPut(volatile uartCircBuff_t * buf, unsigned char c);
13 unsigned char bufGet(volatile uartCircBuff_t * buf);
14 unsigned int buff_modulo_inc(const unsigned int value, const unsigned int
15 modulus);
16
17 void initUARTBufferSystem(DRV_HANDLE uartHandle0,DRV_HANDLE
18 uartHandle1,DRV_HANDLE uartHandle2,DRV_HANDLE uartHandle3)
19 {
20     recBuf0.head      = 0;
21     recBuf0.tail      = 0;
22     recBuf0.count     = 0;
23     wipeBuffer(recBuf0.buf);
24     sendBuf0.head    = 0;
25     sendBuf0.tail    = 0;
26     sendBuf0.count   = 0;
27     wipeBuffer(sendBuf0.buf);
28     internalUART0Handle = uartHandle0;
29
30     recBuf1.head      = 0;
31     recBuf1.tail      = 0;
32     recBuf1.count     = 0;
33     wipeBuffer(recBuf0.buf);
34     sendBuf1.head    = 0;
35     sendBuf1.tail    = 0;
36     sendBuf1.count   = 0;
37     wipeBuffer(sendBuf1.buf);
38     internalUART1Handle = uartHandle1;
39
40     recBuf2.head      = 0;
41     recBuf2.tail      = 0;
42     recBuf2.count     = 0;
43     wipeBuffer(recBuf2.buf);
44     internalUART2Handle = uartHandle2;
45
46     sendBuf3.head    = 0;
47     sendBuf3.tail    = 0;
48     sendBuf3.count   = 0;
49     wipeBuffer(sendBuf3.buf);
50     internalUART3Handle = uartHandle3;
51 }
52
53 void initializeUARTHandles(void)
54 {
55     /*=====USART 0 (485 TX)=====*/
56     SYS_STATUS usart0Status = DRV_USART_Status(sysObj.drvUsart0);
57     if (SYS_STATUS_READY == usart0Status)
58     {
59         //Tie the driver to a tx buffer transmitting function (handle buffer tx))
60         DRV_USART_ByteTransmitCallbackSet(sysObj.drvUsart0, (void
61             *)UART0_TX_CALLBACK);
62
63         //Tie the driver to a rx buffer receiving function (handle buffer rx))
64         DRV_USART_ByteReceiveCallbackSet(sysObj.drvUsart0, (void

```

```

    *)UART0_RX_CALLBACK);

62     //Setup and start the USART Driver
63     appData.USART0Handle =
64         DRV_USART_Open(sysObj.drvUsart0,DRV_IO_INTENT_READWRITE);
65
66     //Baud Rate setting attempt
67     //DRV_USART_BaudSet(appData.USART0Handle,111111);
68
69     //Ensure system start
70     if (DRV_HANDLE_INVALID == appData.USART0Handle)
71     {
72         writeDigitSevenSeg(1,true);
73         while(1);
74     }
75 }

76 /*===== USART 1 (DEBUG 1) =====*/
77 SYS_STATUS usart1Status = DRV_USART_Status(sysObj.drvUsart1);
78 if (SYS_STATUS_READY == usart1Status)
79 {
80     //Tie the driver to a tx buffer transmitting function (handle buffer tx)
81     DRV_USART_ByteTransmitCallbackSet(sysObj.drvUsart1, (void
82     *)UART1_TX_CALLBACK);
83
84     //Tie the driver to a rx buffer receiving function (handle buffer rx)
85     DRV_USART_ByteReceiveCallbackSet(sysObj.drvUsart1, (void
86     *)UART1_RX_CALLBACK);
87
88     //Setup and start the USART Driver
89     appData.USART1Handle =
90         DRV_USART_Open(sysObj.drvUsart1,DRV_IO_INTENT_READWRITE);
91
92     //Baud Rate setting attempt
93     //DRV_USART_BaudSet(appData.USART1Handle,115200);
94
95     //Ensure system start
96     if (DRV_HANDLE_INVALID == appData.USART1Handle)
97     {
98         writeDigitSevenSeg(2,true);
99         while(1);
100    }
101
102 /*===== USART 2 (485 RX)=====*/
103 SYS_STATUS usart2Status = DRV_USART_Status(sysObj.drvUsart2);
104 if (SYS_STATUS_READY == usart2Status)
105 {
106     //NO TX on UART2
107
108     //Tie the driver to a rx buffer receiving function (handle buffer rx)
109     DRV_USART_ByteReceiveCallbackSet(sysObj.drvUsart2, (void
110     *)UART2_RX_CALLBACK);
111
112     //Setup and start the USART Driver
113     appData.USART2Handle =
114         DRV_USART_Open(sysObj.drvUsart2,DRV_IO_INTENT_READWRITE);
115
116     //Baud Rate setting attempt
117     //DRV_USART_BaudSet(appData.USART2Handle,115200);
118
119     //Ensure system start
120     if (DRV_HANDLE_INVALID == appData.USART2Handle)
121     {
122         writeDigitSevenSeg(3,true);

```

```

120         while(1);
121     }
122 }
/*===== USART 3 (DEBUG 2)=====*/
124 SYS_STATUS usart3Status = DRV_USART_Status(sysObj.drvUsart3);
125 if (SYS_STATUS_READY == usart3Status)
126 {
127     //Tie the driver to a tx buffer receiving function (handle buffer tx)
128     DRV_USART_ByteCallbackSet(sysObj.drvUsart3,(void
129     *)UART3_TX_CALLBACK);
130
131     //NO RX on UART3
132
133     //Setup and start the USART Driver
134     appData.USART3Handle =
135     DRV_USART_Open(sysObj.drvUsart3,DRV_IO_INTENT_READWRITE);
136
137     //Baud Rate setting attempt
138     //DRV_USART_BaudSet(appData.USART3Handle,115200);
139
140     //Ensure system start
141     if (DRV_HANDLE_INVALID == appData.USART3Handle)
142     {
143         writeDigitSevenSeg(4,true);
144         while(1);
145     }
146 }
/*===== USART 0 =====*/
147 void writeByteUART0(unsigned char ch)
148 {
149     //DUPLICATE THE OUTPUT TO THE DEBUG 2 CHANNEL
150     //writeByteUART3(ch);
151
152     SYS_INT_SourceDisable( INT_SOURCE_USART_1_TRANSMIT );
153     bufPut(&sendBuf0,ch);
154     //If the ISR is not delivering bytes to the TX, force it to
155     //if(transmitStall0 && !DRV_USART_TransmitBufferIsFull(internalUART0Handle))
156     {
157         DRV_USART_WriteByte(internalUART0Handle,bufGet(&sendBuf0));
158     }
159     else if(DRV_USART_TransmitBufferIsFull(internalUART0Handle))
160     {
161         transmitStall0=false;
162     }
163     SYS_INT_SourceEnable( INT_SOURCE_USART_1_TRANSMIT );
164 }
165
166 void UART0_TX_CALLBACK(void)
167 {
168     //If there is data stored in the uart buffer waiting to be sent
169     if(sendBuf0.count>0)
170     {
171         DRV_USART_WriteByte(internalUART0Handle,bufGet(&sendBuf0));
172     }
173     else
174     {
175         transmitStall0 = true;
176     }
177 }
178
179 /*UNUSED*/
180 uint16_t rxBytesAvailable0(void)
181 {

```

```

183     return recBuf0.count;
184 }
185 /*UNUSED*/
186 unsigned char peekByteUART0(void)
187 {
188     if(recBuf0.head != recBuf0.tail)
189         return recBuf0.buff[recBuf0.tail];
190     else
191         return 0;
192 }
193 /*UNUSED*/
194 unsigned char readByteUART0(void)
195 {
196     if(recBuf0.head != recBuf0.tail)
197     {
198         //grab the byte we want out
199         uint8_t byteBack = bufGet(&recBuf0);
200         return byteBack;
201     }
202     return 0;
203 }
204 /*UNUSED*/
205 void UART0_RX_CALLBACK(void)
206 {
207     unsigned char readByte= DRV_USART_ReadByte(internalUART0Handle);
208     //writeByteUART1(readByte);
209     bufPut(&recBuf0,readByte);
210 }
211 /*===== USART 1 DEBUG =====*/
212 void writeByteUART1(unsigned char ch)
213 {
214     SYS_INT_SourceDisable( INT_SOURCE_USART_2_TRANSMIT );
215     bufPut(&sendBuf1,ch);
216     //If the ISR is not delivering bytes to the TX, force it to
217     if(transmitStall1 && !DRV_USART_TransmitBufferIsFull(internalUART1Handle))
218     {
219         DRV_USART_WriteByte(internalUART1Handle,bufGet(&sendBuf1));
220     }
221     else if(DRV_USART_TransmitBufferIsFull(internalUART1Handle))
222     {
223         transmitStall1=false;
224     }
225     SYS_INT_SourceEnable( INT_SOURCE_USART_2_TRANSMIT );
226 }
227 void UART1_TX_CALLBACK(void)
228 {
229     //If there is data stored in the uart buffer waiting to be sent
230     if(sendBuf1.count>0)
231         DRV_USART_WriteByte(internalUART1Handle,bufGet(&sendBuf1));
232     else
233         transmitStall1 = true;
234 }
235 uint16_t rxBytesAvailable1(void)
236 {
237     return recBuf1.count;
238 }
239 unsigned char peekByteUART1(void)

```

```

247 {
248     if(recBuf1.head != recBuf1.tail)
249         return recBuf1.buff[recBuf1.tail];
250     else
251         return 0;
252 }
253
254 unsigned char readByteUART1(void)
255 {
256     if(recBuf1.head != recBuf1.tail)
257     {
258         //grab the byte we want out
259         uint8_t byteBack = bufGet(&recBuf1);
260         return byteBack;
261     }
262     return 0;
263 }
264
265 void UART1_RX_CALLBACK(void)
266 {
267     //unsigned char readByte=DRV_USART_ReadByte(internalUART1Handle);
268     bufPut(&recBuf1,DRV_USART_ReadByte(internalUART1Handle));
269     //writeByteUART1(readByte);
270 }
271
272
273 uint16_t rxBytesAvailable2(void)
274 {
275     return recBuf2.count;
276 }
277
278 unsigned char peekByteUART2(void)
279 {
280     if(recBuf2.head != recBuf2.tail)
281         return recBuf2.buff[recBuf2.tail];
282     else
283         return 0;
284 }
285
286 unsigned char readByteUART2(void)
287 {
288     if(recBuf2.head != recBuf2.tail)
289     {
290         //grab the byte we want out
291         uint8_t byteBack = bufGet(&recBuf2);
292         return byteBack;
293     }
294     return 0;
295 }
296
297 void UART2_RX_CALLBACK(void)
298 {
299     unsigned char readByte=DRV_USART_ReadByte(internalUART2Handle);
300     bufPut(&recBuf2,readByte); //DRV_USART_ReadByte(internalUART2Handle);
301
302     //DUPLICATE THE RX 485 TO THE DEBUG 2 CHANNEL
303     //writeByteUART3(readByte);
304     //DUPLICATE THE RX 485 TO THE DEBUG 1 CHANNEL
305     //writeByteUART1(readByte);
306 }
307
308 void wipeBuffer2(void)
309 {
310     wipeBuffer(recBuf2.buff);
311 }

```

```

312
313 //-----USART 3 DEBUG NUM 2-----
314 void writeByteUART3(unsigned char ch)
315 {
316     SYS_INT_SourceDisable( INT_SOURCE_USART_3_TRANSMIT );
317     bufPut(&sendBuf3,ch);
318     //If the ISR is not delivering bytes to the TX, force it to
319     if(transmitStall3 && !DRV_USART_TransmitBufferIsFull(internalUART3Handle))
320     {
321         DRV_USART_WriteByte(internalUART3Handle,bufGet(&sendBuf3));
322     }
323     else if(DRV_USART_TransmitBufferIsFull(internalUART3Handle))
324     {
325         transmitStall3=false;
326     }
327     SYS_INT_SourceEnable( INT_SOURCE_USART_3_TRANSMIT );
328 }
329
330 void UART3_TX_CALLBACK(void)
331 {
332     //If there is data stored in the uart buffer waiting to be sent
333     if(sendBuf3.count>0)
334     {
335         DRV_USART_WriteByte(internalUART3Handle,bufGet(&sendBuf3));
336     }
337     else
338     {
339         transmitStall3 = true;
340     }
341 }
342
343 /*=====Generic UART Functions=====*/
344 void wipeBuffer(volatile unsigned char * buf)
345 {
346     int i=0;
347     for(i=0;i<UART_BUFFER_LENGTH;i++)
348     {
349         buf[i]=0;
350     }
351 }
352
353 void bufPut(volatile uartCircBuff_t * buf, unsigned char c)
354 {
355     if (buf->count < UART_BUFFER_LENGTH)
356     {
357         buf->buff[buf->head] = c;
358         buf->head = buff_modulo_inc(buf->head, UART_BUFFER_LENGTH);
359         ++buf->count;
360     }
361     else
362     {
363         buf->buff[buf->head] = c;
364         buf->head = buff_modulo_inc(buf->head, UART_BUFFER_LENGTH);
365         buf->tail = buff_modulo_inc(buf->tail, UART_BUFFER_LENGTH);
366     }
367 }
368
369 unsigned char bufGet(volatile uartCircBuff_t * buf)
370 {
371     unsigned char c;
372     if (buf->count > 0)
373     {
374         c = buf->buff[buf->tail];
375         buf->buff[buf->tail]=0;
376     }

```

```
377         buf->tail = buff_modulo_inc(buf->tail, UART_BUFFER_LENGTH);
378         --buf->count;
379     }
380     else
381     {
382         c = 0;
383     }
384     return (c);
385 }
386
387 unsigned int buff_modulo_inc(const unsigned int value, const unsigned int modulus)
388 {
389     unsigned int my_value = value + 1;
390     if (my_value >= modulus) // WAS >= SHOULD BE JUST >
391     {
392         my_value = 0;
393     }
394     return (my_value);
395 }
```

3.2.6 Main Control Neural Network Generation

The neural networks that the MCB uses to localize itself within the arena and to identify the enemy robot were generated within Matlab and exported as C code that the PIC processor on the MCB executes. The training data for the neural network was generated by simulating the 8 measurements the LIDAR sensors would acquire as a function of position and angle of the robot within the arena. The geometry of the arena and relative positioning of the robot can be seen in Figure 20. The length of the entire arena wall is “L” and half of the area wall length is “l”. The position of the robot within the area is characterized by the position on the x-axis, X_p , and the position on the Y-axis, Y_p . The center of the arena is the origin for the Cordiant system. The last important parameter is the angles from corner to corner of the arena formed from the position of the robot. These angles are denoted as alpha 1 though alpha 4. (MJH)

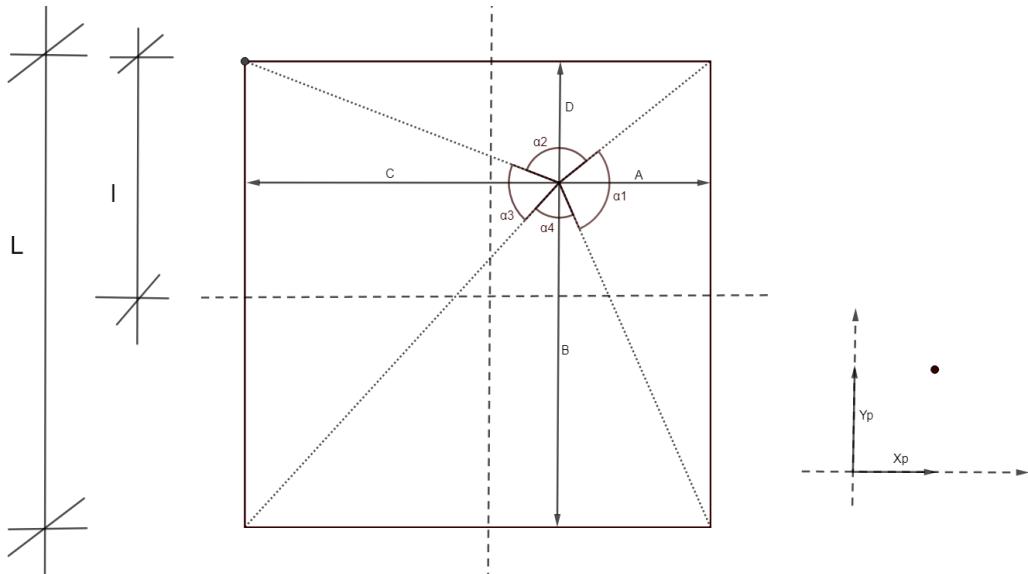


Figure 20: Neural Network Training Data Simulation Positioning

From the geometry of the arena and the position of the robot within it, the equation for distance to a wall as a function of position and angle can be derived as seen in Figure 21. Because of the four walls of the arena the equation is a four-element piecewise wise function

where each element corresponds to a single wall. The interval, over which each subdomain function is valid, is bounded by the angle that forms between the position of the robot and two corners of the arena. The magnitude of the bound of the first subdomain function is equivalent to alpha 1, the magnitude of the bound of the second subdomain function is equivalent to alpha 2 and so on. (MJH)

$$W_d = \begin{cases} |(l - X_p) * \sec(\alpha)| & \tan^{-1}\left(\frac{l + X_p}{l - Y_p}\right) \leq \alpha \leq \tan^{-1}\left(\frac{l - X_p}{l - Y_p}\right) \\ |(l - Y_p) * \sec(\alpha + 90)| & \tan^{-1}\left(\frac{l - Y_p}{l - X_p}\right) \leq \alpha \leq \tan^{-1}\left(\frac{l + Y_p}{l - X_p}\right) \\ |(l + X_p) * \sec(\alpha + 180)| & \tan^{-1}\left(\frac{l - X_p}{l + Y_p}\right) \leq \alpha \leq \tan^{-1}\left(\frac{l + X_p}{l + Y_p}\right) \\ |(l + Y_p) * \sec(\alpha + 270)| & \tan^{-1}\left(\frac{l + Y_p}{l + X_p}\right) \leq \alpha \leq \tan^{-1}\left(\frac{l - Y_p}{l + X_p}\right) \end{cases}$$

Figure 21: Neural Network Training Data Simulation Wall Distance Equation

From the derived equation in Figure 21 a training set of 20,000 samples was generated. The sample data was generated at a random position in the arena with the robot facing a random angle. The position on angle were used to create the training set of the 8 measurements the LIDAR sensors would acquire. The LIDAR sensors on the robot are separated 45 degrees apart so by adding 45 to alpha the distance each following sensor would acquire could be calculated. The 8 calculated LIDAR measurements are used for the training set. The solution set for the neural network is derived from the position and angle of the robot. The position of the robot must first be adjusted to the symmetric equivalent position within the first octet of the arena. Because the arena is a square, the symmetry results in 8 positions in which the set of lidar measurements would be equivalent. To allow for this the neural network is trained on the symmetric equivalent position in the first octet of the arena. (MJH)

The enemy detection was achieved by training a neural network to identify which of the 8 LIDAR sensors is facing the enemy robot. The sample set used to train the previous neural network was used with a single modification. A random sensor would have its measurement reduced to and random percent of its original value. The solution set for the enemy detection neural networks outputs a pseudo percent of certainty of which sensor is facing the enemy robot. The solution set also has a placeholder that identifies when none of the sensors are facing the enemy robot. (MJH)

3.2.7 Main Control Neural Network Generation Code

```

clc

clear all
close all

% The position (0,0) is the center of the arena
% L is the length of the arena wall
L = 40;
% 1220 cm is approximately 40 ft
L = 1220;
l = L/2;
Samples = 20000;
Aquired_Measurements = 8;

% Create a matrix that will store the distances measured
Measured_Distance = zeros(Samples,Aquired_Measurements);

% Create a matrix to store training solutions
%Soulution_Matrix = zeros(Samples,3);
Soulution_Matrix = zeros(Samples,3);
Loc_Soulution_Matrix = zeros(Samples,8);

for k = 1:Samples
% Generate a random position and Angle(k) (X, Y, alpha)
Xp(k) = randi(L)-l-1+(randi(100)/101);
Yp(k) = randi(L)-l-1+(randi(100)/101);

Starting_Xp(k) = Xp(k);
Starting_Yp(k) = Yp(k);

if((Yp(k)>=0) && (Xp(k)>=0))% First Quad
    if(Yp(k)>Xp(k)) % In octet 2

```

```

Xp(k) = Starting_Yp(k);
Yp(k) = Starting_Xp(k);
else % In octet 1
    Xp(k) = Starting_Xp(k);
    Yp(k) = Starting_Yp(k);
end
elseif((Yp(k)>=0) && (Xp(k)<0)) % Second Quad
    if(abs(Yp(k))>abs(Xp(k))) % In octect 3
        Xp(k) = Starting_Yp(k);
        Yp(k) = (-1)*Starting_Xp(k);
    else % In octect 4
        Xp(k) = (-1)*Starting_Xp(k);
        Yp(k) = Starting_Yp(k);
    end
elseif((Yp(k)<0) && (Xp(k)<0)) % Third Quad
    if(abs(Yp(k))>abs(Xp(k))) % In octect 6
        Xp(k) = (-1)*Starting_Yp(k);
        Yp(k) = (-1)*Starting_Xp(k);
    else % In octect 5
        Xp(k) = (-1)*Starting_Xp(k);
        Yp(k) = (-1)*Starting_Yp(k);
    end
elseif((Yp(k)<0) && (Xp(k)>=0)) % Fourth Quad
    if(abs(Yp(k))>abs(Xp(k))) % In octect 7
        Xp(k) = (-1)*Starting_Yp(k);
        Yp(k) = Starting_Xp(k);
    else % In octect 8
        Xp(k) = Starting_Xp(k);
        Yp(k) = (-1)*Starting_Yp(k);
    end
end

Angle(k) = randi(360)+(randi(100)/101);
Starting_Angle(k) = Angle(k);

% Store solutions
%%%%%%%%%%%%%
% Orientation Anlge Stored in 1
if(abs(rad2deg(atan(abs(Yp(k))/abs(Xp(k)))))) <= 45)
    Soulution_Matrix(k,1) = abs(rad2deg(atan(abs(Yp(k))/abs(Xp(k))))));
elseif((45 < abs(rad2deg(atan(abs(Yp(k))/abs(Xp(k)))))) &
(abs(rad2deg(atan(abs(Yp(k))/abs(Xp(k)))))) <= 90))
    Soulution_Matrix(k,1) = 90 - abs(rad2deg(atan(abs(Yp(k))/abs(Xp(k))))));
end
% Distance from center stored in 2
Soulution_Matrix(k,2) = sqrt(Xp(k)^2 + Yp(k)^2);

[Psuedo_Xp(k) Psuedo_Yp(k)] = pol2cart(deg2rad(Starting_Angle(k)),1);

```

```

if((Starting_Yp(k)>=0) && (Starting_Xp(k)>=0))% First Quad
    if(Starting_Yp(k)>Starting_Xp(k)) % In octet 2
        Sym_Psu_Xp = Psuedo_Yp(k);
        Sym_Psu_Yp = Psuedo_Xp(k);
        [Soulution_Matrix(k,3) Psu_Mag] = cart2pol(Sym_Psu_Xp,Sym_Psu_Yp);

    else % In octet 1
        Sym_Psu_Xp = Psuedo_Xp(k);
        Sym_Psu_Yp = Psuedo_Yp(k);
        [Soulution_Matrix(k,3) Psu_Mag] = cart2pol(Sym_Psu_Xp,Sym_Psu_Yp);
    end

elseif((Starting_Yp(k)>=0) && (Starting_Xp(k)<0)) % Second Quad
    if(abs(Starting_Yp(k))>abs(Starting_Xp(k))) % In octect 3
        Sym_Psu_Xp = Psuedo_Yp(k);
        Sym_Psu_Yp = (-1)*Psuedo_Xp(k);
        [Soulution_Matrix(k,3) Psu_Mag] = cart2pol(Sym_Psu_Xp,Sym_Psu_Yp);
    else % In octect 4
        Sym_Psu_Xp = (-1)*Psuedo_Xp(k);
        Sym_Psu_Yp = Psuedo_Yp(k);
        [Soulution_Matrix(k,3) Psu_Mag] = cart2pol(Sym_Psu_Xp,Sym_Psu_Yp);
    end

elseif((Starting_Yp(k)<0) && (Starting_Xp(k)<0)) % Third Quad
    if(abs(Starting_Yp(k))>abs(Starting_Xp(k))) % In octect 6
        Sym_Psu_Xp = (-1)*Psuedo_Yp(k);
        Sym_Psu_Yp = (-1)*Psuedo_Xp(k);
        [Soulution_Matrix(k,3) Psu_Mag] = cart2pol(Sym_Psu_Xp,Sym_Psu_Yp);
    else % In octect 5
        Sym_Psu_Xp = (-1)*Psuedo_Xp(k);
        Sym_Psu_Yp = (-1)*Psuedo_Yp(k);
        [Soulution_Matrix(k,3) Psu_Mag] = cart2pol(Sym_Psu_Xp,Sym_Psu_Yp);
    end

elseif((Starting_Yp(k)<0) && (Starting_Xp(k)>=0)) % Fourth Quad
    if(abs(Starting_Yp(k))>abs(Starting_Xp(k))) % In octect 7
        Sym_Psu_Xp = (-1)*Psuedo_Yp(k);
        Sym_Psu_Yp = Psuedo_Xp(k);
        [Soulution_Matrix(k,3) Psu_Mag] = cart2pol(Sym_Psu_Xp,Sym_Psu_Yp);
    else % In octect 8
        Sym_Psu_Xp = Psuedo_Xp(k);
        Sym_Psu_Yp = (-1)*Psuedo_Yp(k);
        [Soulution_Matrix(k,3) Psu_Mag] = cart2pol(Sym_Psu_Xp,Sym_Psu_Yp);
    end
end

Soulution_Matrix(k,3) = rad2deg(Soulution_Matrix(k,3));

```

```

if(Soulution_Matrix(k,3) > 360)
    Soulution_Matrix(k,3) = Soulution_Matrix(k,3)-360;
end
if(Soulution_Matrix(k,3) < 0)
    Soulution_Matrix(k,3) = Soulution_Matrix(k,3)+360;
end

Pos_Alpha_W1(k) = radtodeg(atan(((1-Yp(k))/(1-Xp(k)))));

Neg_Alpha_W2(k) = Pos_Alpha_W1(k)+ radtodeg(atan(((1-Xp(k))/(1-Yp(k)))));
Pos_Alpha_W2(k) = Neg_Alpha_W2(k) + radtodeg(atan(((1+Xp(k))/(1-Yp(k))));

Neg_Alpha_W3(k) = Pos_Alpha_W2(k)+ radtodeg(atan(((1-Yp(k))/(1+Xp(k)))));
Pos_Alpha_W3(k) = Neg_Alpha_W3(k) + radtodeg(atan(((1+Yp(k))/(1+Xp(k))));

Neg_Alpha_W4(k) = Pos_Alpha_W3(k)+ radtodeg(atan(((1+Xp(k))/(1+Yp(k))));
Pos_Alpha_W4(k) = Neg_Alpha_W4(k) + radtodeg(atan(((1-Xp(k))/(1+Yp(k))));

Neg_Alpha_W1 = Pos_Alpha_W4(k) + radtodeg(atan(((1+Yp(k))/(1-Xp(k))));

% Calculate the measured distances
Angle(k) = Soulution_Matrix(k,3);

for i = 1:Aquired_Measurements
    % Distance if facing wall 1
    if (Angle(k) < Pos_Alpha_W1(k));
        Measured_Distance(k,i) = ((1-Xp(k))*abs(sec(degtorad((Angle(k)))));

    % Distance if facing wall 2
    elseif (Pos_Alpha_W1(k) < Angle(k) && (Angle(k) <= Pos_Alpha_W2(k));
        Measured_Distance(k,i) = ((1-Yp(k))*abs(sec(degtorad((Angle(k) + 90)))));

    % Distance if facing wall 3
    elseif (Pos_Alpha_W2(k) < Angle(k) && (Angle(k) <= Pos_Alpha_W3(k));
        Measured_Distance(k,i) = ((1+Xp(k))*abs(sec(degtorad((Angle(k) + 180)))));

    % Distance if facing wall 4
    elseif (Pos_Alpha_W3(k) < Angle(k) && (Angle(k) <= Pos_Alpha_W4(k));
        Measured_Distance(k,i) = ((1+Yp(k))*abs(sec(degtorad((Angle(k) + 270)))));

    % Distance if facing wall 1 (negative Angle(k)s)
    else Angle(k) <= 360;
        Measured_Distance(k,i) = ((1-Xp(k))*abs(sec(degtorad((Angle(k)))));
    end
    Measured_Angles(k,i) = Angle(k);
    Angle(k) = Angle(k) + 45;
    if (Angle(k) > 360)
        Angle(k) = Angle(k) - 360;
    end

```

```

    end

end

% Store the origional Distances
for k=1:Samples
    for i=1:Aquired_Measurements
        Measured_Distance_Origional(k,i) = Measured_Distance(k,i);
    end
end
X = Measured_Distance';

T = Soulution_Matrix';

setdemorandstream(491218382);

% https://www.mathworks.com/help/nnet/ref/fitnet.html
%net = feedforwardnet([20 20], 'trainbr');
net = feedforwardnet([100 100 100], 'trainscg');
net.trainParam.goal = 0.01;
net.trainParam.epochs = Samples;
net.trainParam.max_fail = 200;
%net = fitnet([8 8 8],'trainlm');
%view(net)
[net,tr] = train(net,X,T);
%[net,tr] = train(net,X,T,'useGPU','yes');
nntraintool
plotperform(tr)
% To test the generated net use the following two lines
% net(X(:,6))
% T(:,6)
% https://www.mathworks.com/matlabcentral/answers/264160-how-to-save-and-reuse-a-trained-neural-networkfor
k=1:Samples
if(~mod(mod(randi(100),5),2))
    Enemy_Local = randi(8);
    Measured_Distance(k,Enemy_Local) = rand() * Measured_Distance(k,Enemy_Local);
    %Measured_Distance(Enemy_Local,i) = Measured_Distance(Enemy_Local,i) - rand()
    * Measured_Distance(Enemy_Local,i);
    %Loc_Soulution_Matrix(k,Enemy_Local+1) = 100;
    Loc_Soulution_Matrix(k,Enemy_Local+1) = 100;
else
    Loc_Soulution_Matrix(k,1) = 100;
end
Measured_Distance(k,9) = Xp(k);
Measured_Distance(k,10) = Yp(k);
end
Loc_X = Measured_Distance';

Loc_T = Loc_Soulution_Matrix';

```

```

Loc_net = feedforwardnet([100 100 100], 'trainscg');
Loc_net.trainParam.goal = 0.0001;
Loc_net.trainParam.epochs = Samples;
Loc_net.trainParam.max_fail = 200;
[Loc_net,tr] = train(Loc_net,Loc_X,Loc_T);
nntraintool
plotperform(tr)

```

(MJH)

3.3.0 Motor Driver Board

The motor driver board is used to control the motors that propel the robot. This board receives RS485 data that contains speed and direction information. The received data is processed and sent to a closed loop controller to match the desired speed of rotation to the actual speed of rotation. (MJH)

3.3.1 Motor Driver Board Hardware

Figure 22 shows the level 2 block diagram of the main subsystems the motor driver board will use to function. The received communication data is first passed through a level shifter to convert the incoming data to an acceptable level for the micro controller. The received data is then processed and transmitted to the PWM generator circuitry. A hardware PWM generator is used so that the microcontroller can carry out other operations while the PWM signal is constantly being outputted. The PWM signal is received by the bridge driver circuitry. The bridge driver processes the received PWM signal and drives the gates of the MOSFETs in the H-bridge. The bridge driver circuitry also generates a dead-time between switching of MOSFETs to prevent shoot-through. Protection logic is also included in the bridge driver to prevent a shoot-through conditions that could damage the H-bridge MOSFETs or the batteries. Once the desired output speed it calculated and transmitted to the H-bridge, the desired speed is compared to the actual speed of the motor through the decoded data from the rotary encoder decoder. The signal

to the rotary encoder decoder is a quadrature signal. The direction of the motor is determined by sending the quadrature data to a D-type flip-flop. The flip-flop will output a 1 or a 0 depending on the rotation direction. The speed of rotation is determined by using a single quadrature line and a counter. When the counter is sampled after a known time the speed of rotation can be calculated by the ratio of pulse counts to time between samples. If the desired speed of rotation is less than the actual speed of rotation the duty ratio must decrease. If the desired speed is greater than the actual speed, then the duty ratio must increase. The board monitor sensors provide temperature and voltage data to the microcontroller. If the temperature of the board rises to a point where the board or components on the board can be damaged, then the microprocessor will proportionally lower the desired speed to prevent damage. A micro-SD card is included on the board to log sensor data and operation data of the motor driver board. The logged data will allow operation data to be analyzed when the board is not in use. (MJH)

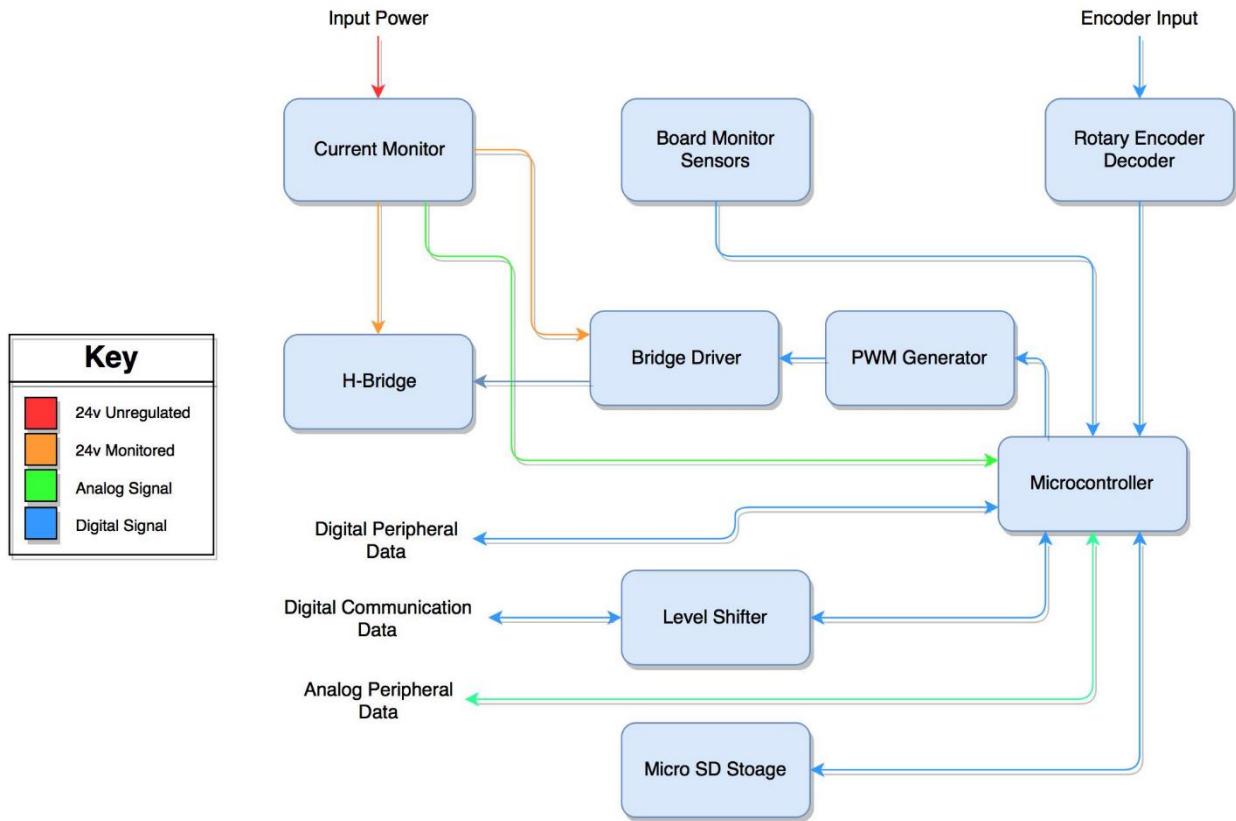


Figure 22: Level 2 Motor Driver Board Hardware Block Diagram

Module	Current Monitor
Designer	Mack J. Hawkins
Inputs	Power: Unregulated 24 Vdc Power: Regulated 5 Vdc
Outputs	Power: Monitored 24 Vdc Signal: 0-5 VDC Analog Voltage Proportional to Input Current
Description	The current monitor will provide the microcontroller with information about the current being drawn. This will allow the microcontroller to assess if there is an issue with the motors and regulate the power going to the motors.

Table 32: Level 2 Motor Driver Board Hardware Current Monitor Module

Module	H-Bridge
Designer	Mack J. Hawkins
Inputs	Power: Monitored 24 Vdc Signal: PWM Control Signal
Outputs	Power: High power PWM signal to drive the locomotion motors
Description	The H-Bridge allows control of the locomotion motor speed and direction without the need of a negative power rail.

Table 33: Level 2 Motor Driver Board Hardware H-Bridge Module

Module	PWM Generator
Designer	Mack J. Hawkins
Inputs	Power: Regulated 5 Vdc Signal: I2C Speed and Direction Data
Outputs	Signal: PWM Control Data
Description	This section of the board generates the PWM signal to control the speed at which the motor will rotate.

Table 34: Level 2 Motor Driver Board Hardware PWM Generator Module

Module	Bridge Driver
Designer	Mack J. Hawkins
Inputs	Power: Regulated 5 Vdc Power: Monitored 24 Vdc Signal: PWM Control Data
Outputs	Signal: PWM Control Data
Description	This chip will drive the gates of the FETs in the H-Bridge. The chip will include a charge pump circuit so that NPN FETs can be used on the high and low side of the H-Bridge. The chip will also include an internal deadtime generator to prevent shoot though while the motor controller is operating.

Table 35: Level 2 Motor Driver Board Hardware Bridge Driver Module

Module	Microcontroller
Designer	Mack J. Hawkins
Inputs	Power: Regulated 5 Vdc Signal: TTL UART Speed and Direction Data Signal: 0-5 VDC Analog Input Current Data Signal: I2C Board Temperature Data Signal: I2C Input Voltage Data Signal: Digital Peripherals Signal: Analog Peripherals
Outputs	Signal: LED Status Signal: I2C Speed and Direction Data
Description	The microcontroller will monitor the state of the board and process the control signal. The processed signal will be sent to the pre-driver to operate the H-Bridge and drive the motors.

Table 36: Level 2 Motor Driver Board Hardware Microcontroller Module

Module	Micro SD Storage
Designer	Mack J. Hawkins
Inputs	Power: Regulated 3.3 Vdc Signal: SPI
Outputs	Data: Stored Data on the Micro SD Card
Description	The micro SD storage is used so the operating data of the motor controller can be examined when the board is not in use.

Table 37: Level 2 Motor Driver Board Hardware Micro SD Storage Module

Module	Level Shifter
Designer	Mack J. Hawkins
Inputs	Signal: RS485 Digital Communication Signal: TTL UART Digital Communication
Outputs	Signal: RS485 Digital Communication Signal: TTL UART Digital Communication
Description	This block changes the level of digital communication data down to TTL for the microcontroller.

Table 38: Level 2 Motor Driver Board Hardware Level Shifter Module

Module	Board Monitor Sensors
Designer	Mack J. Hawkins
Inputs	Signal: Temperature Signal: Voltage
Outputs	Signal: I2C Data containing voltage and temperature
Description	These sensors will be used to monitor the physical status of the board.

Table 39: Level 2 Motor Driver Board Hardware Board Monitor Sensor Module

Module	Rotary Encoder Decoder
Designer	Mack J. Hawkins
Inputs	Signal: Quadrature
Outputs	Signal: 8 Bit Counter Signal: 1 Bit Direction
Description	This block decodes the quadrature data from the encoder to a more usable digital communication signal. This frees up processing time on the microcontroller so that encoder data can be sampled instead of continuously monitored.

Table 40: Level 2 Motor Driver Board Hardware Rotary Encoder Decoder Module

3.3.2 Motor Driver Board Software

The high-level overview of the software implemented on the motor driver board can be seen in Figure 23. The received data containing speed and direction information is first adjusted by the thermal throttling before being sent to the closed loop controller. The protection throttle monitors the temperature of the board and adjust the desired speed if the temperature of the board rises to a threshold temperature that could damage the board. The closed loop controller receives the adjusted speed and direction data and outputs a control signal. If the actual speed of the motor does not match the desired speed, then the closed loop controller will adjust the control signal accordingly. (MJH)

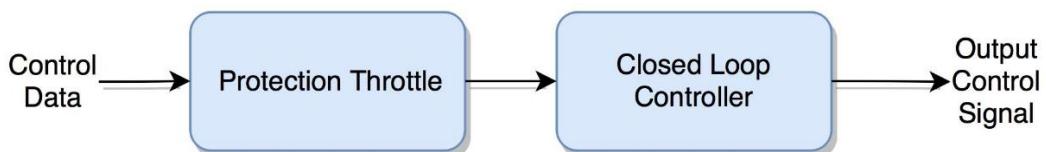


Figure 23: Level 1 Motor Driver Board Software Block Diagram

Module	Protection Throttle
Designer	Mackenzie Hawkins
Inputs	Signal: Speed and Direction Data
Outputs	Signal: Adjusted Speed and Direction Data
Description	If the microcontroller detects that the board is operating at higher temperatures than preferred the speed will be scaled down to prevent damage to the board.

Table 41: Level 1 Motor Driver Board Software Protection Throttle Module

Module	Closed Loop Controller
Designer	Mackenzie Hawkins
Inputs	Signal: Proportional Speed and Direction Data
Outputs	Signal: PID Controlled Speed and Direction Data
Description	The closed loop controller will control the output signal to match the adjusted speed request regardless of load on the output.

Table 42: Level 1 Motor Driver Board Software Closed Loop Controller Module

Figure 24 shows the level 2 block diagram of the software that will be used to control the motor driver board. The received RS485 data containing speed and direction information must first be decoded so that the speed is a separate value than the direction. The decoded data is then sent to a thermal throttle function that adjusts the desired speed if needed. The desired speed is decreased if the temperature of the board exceeds a threshold that could cause damage to the board. The temperature data is read from the temperature sensors and then mapped to a value corresponding to the value of the received speed data. The PID controller receives the mapped thermally adjusted speed data and the mapped encoder count. The encoder counter is mapped according to the sample time of the counter and then again to an appropriate value to match the format of the thermally adjusted desired speed data. The PID controller monitors the encoder data and desired speed data. The outputted signal from the PID controller is adjusted so that the actual speed of rotation matches the desired speed of rotation. The outputted data from the PID controller is used to write the PWM duty ratio that controls the H-Bridge of the motor controller.

(MJH)

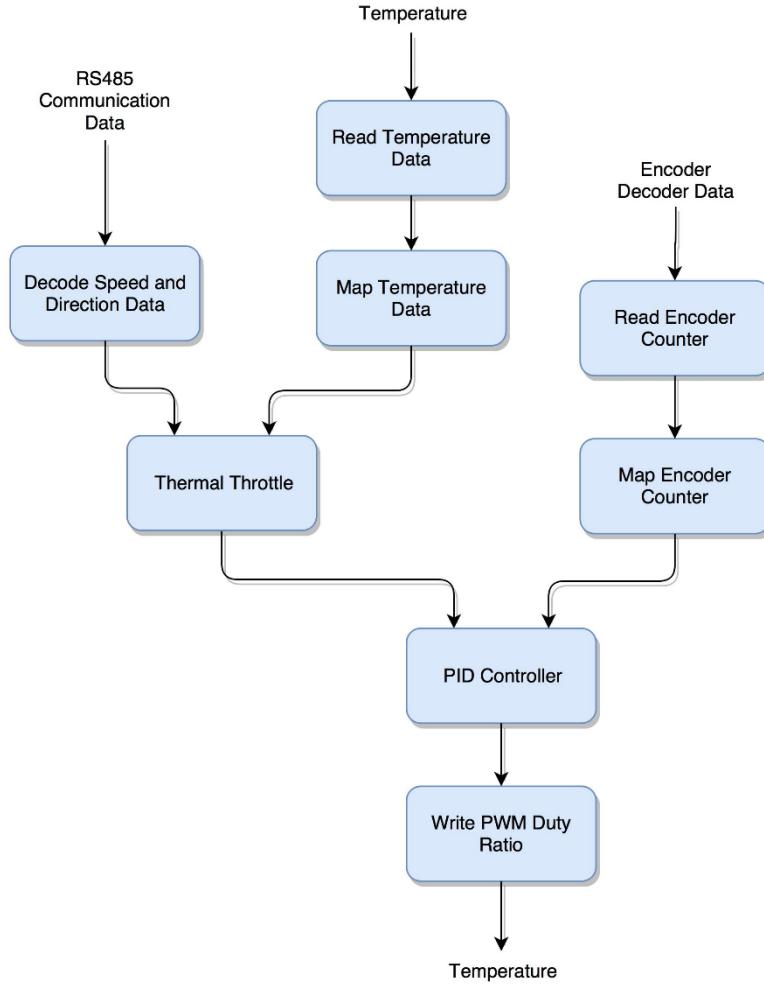


Figure 24: Level 2 Motor Driver Board Software Block Diagram

Module	Decode Speed and Direction Data
Designer	Mackenzie Hawkins
Inputs	Signal: RS485
Outputs	Signal: Speed and Direction Variables
Description	Received RS485 is decoded into variables for speed and direction data.

Table 43: Level 2 Motor Driver Board Software Decode Speed and Direction Data Module

Module	Read Temperature Data
Designer	Mackenzie Hawkins
Inputs	Signal: I2C
Outputs	Signal: Temperature Variable
Description	Received temperature data is wrote to temperature variables.

Table 44: Level 2 Motor Driver Board Software Read Temperature Data Module

Module	Map Temperature Data
Designer	Mackenzie Hawkins
Inputs	Signal: Temperature Variable
Outputs	Signal: Mapped Temperature Variable
Description	The stored temperature variables are mapped to a value range that has the same format of the speed data

Table 45: Level 2 Motor Driver Board Software Map Temperature Data Module

Module	Thermal Throttle
Designer	Mackenzie Hawkins
Inputs	Signal: Speed Variable Signal: Mapped Temperature Variable
Outputs	Signal: Scaled Speed Variable
Description	The thermal throttle adjusts the received set speed if the temperature sensors indicate that the temperature of the board exceeds a threshold at which temperature that could damage the board.

Table 46: Level 2 Motor Driver Board Software Thermal Throttle Module

Module	Read Encoder Counter
Designer	Mackenzie Hawkins
Inputs	Signal: Quadrature
Outputs	Signal: 8-bit variable
Description	The encoder counter increments by 1 every single pulse of a quadrature phase. The count is stored on an 8-bit counter. And read by the microcontroller after a known period of time.

Table 47: Level 2 Motor Driver Board Software Read Encoder Counter Module

Module	Map Encoder Counter
Designer	Mackenzie Hawkins
Inputs	Signal: 8-bit variable
Outputs	Signal: Mapped Encoder Count Variable
Description	The count of the encoder is mapped proportionally to the time between samples.

Table 48: Level 2 Motor Driver Board Software Map Encoder Counter Module

Module	PID Controller
Designer	Mackenzie Hawkins
Inputs	Signal: Duty Ration Variable 0-100% Signal: Mapped Encoder Count Variable
Outputs	Signal: Duty Ration Variable 0-100%
Description	The PID controller compares the adjusted desired speed to the actual speed of rotation and adjust the outputted duty ratio so that the desired speed matches the actual speed of rotation.

Table 49: Level 2 Motor Driver Board Software PID Controller Module

Module	Write PWM Duty Ratio
Designer	Mackenzie Hawkins
Inputs	Signal: Duty Ration Variable 0-100%
Outputs	Signal: I2C Data containing Speed and Direction Data
Description	The speed and direction data is wrote to the devices that generate the PWM signal.

Table 50: Level 2 Motor Driver Board Software Write PWM Duty Ratio Module

3.3.3 Motor Driver Board Schematic

Figure 25 though Figure 33 shows the schematic for the motor driver board. The schematics are derived from the block diagrams of the motor driver board in the previous sections. (MJH)

The input sheet of the schematic seen in Figure 25 contains the power input connectors that the power wires will connect to, a capacitor bank for minimizing voltage ripple on the 24 volt rail, a switching regulator that supplies power to the 5 volt rail, and a 5 volt rail monitor that protects the 5 volt regulator and the devices on the 5 volt rail from an overvoltage scenario. (MJH)

The monitors seen on Figure 26 are the voltage sensors and temperature sensors used to monitor the state of the board. The voltage sensors are used to monitor the input voltage from the battery pack and the voltage from the current sensor that is proportional to the input current to the motor driver board. (MJH)

The microcontroller sheet seen in Figure 27 contains the microprocessor that is used to control the board as well as the connectors for peripheral devices. The connectors for communication to and from the motor driver board are also included on this sheet. (MJH)

The components that support the use of the micro-SD card can be seen in Figure 28. The components include the housing the micro-SD is inserted into, a 3.3v linear regulator, and a level shifter. The micro-SD card requires 3.3 volts to power the devices thus the 5 volt rail needs

regulated to 3.3 volt. Communication to the micro-SD card also needs to be regulated to 3.3 volt therefore a level shifter is needed that changes the 5 volt communication to 3.3 volt communication. (MJH)

The communication sheet seen in Figure 29 contains the RS232 and RS485 transceivers that allow communication to the micro controller via RS232 or RS485 protocol signals. The communication sheet also contains a seven-segment display that will be used to provide board status information to an observer and six DIP switches that will be used to change software settings of the motor driver board. (MJH)

The encoder decoder sheet seen in Figure 30 shows the devices that are used to acquire the state of a quadrature rotary encoder. The received quadrature data from an encoder is first sent to a Schmitt trigger to make the edges of the received signal sharper. The signal is then sent to a d-type flip-flop and inverter. The flip-flop is used to determine the direction of rotation by comparing the phase of the two quadrature signals. The inverter is used as a buffer to guarantee the signal edges are sharp before being sent to a counter. The counter counts the number of rising edges which is proportional to the number of revolutions the encoder has made. The eight bits from the counter are connected to a I2C I/O expander. The micro controller will sample the I/O expander after a known period to calculate the rate of rotation of the encoder. (MJH)

The PWM generator seen in Figure 31 shows the devices that are used to set the PWM signal that will be sent to the H-bridge of the motor driver board. The PWM duty ratio is set by the Linear Technology LTC6992CS6. This chip outputs a duty ratio at a fixed frequency. The duty ratio is proportional to a 0 to 1 volt signal on the MOD pin. The frequency is set by the resistors connected to SET and DIV pins. The voltage that sets the duty ratio is outputted by an

I2C DAC. The voltage from the DAC is 0 to 5 volts therefor a voltage divider is needed to lower the voltage to the acceptable 0 to 1 volt of the LTC6992CS6 MOD pin. (MJH)

The H-bridge driver seen in Figure 32 contains the devices that are used to control the H-bridge of the motor driver board. This is accomplished with the Allegro MicroSystems A3921. This chip has a built-in gate driver, charge pump for the high side MOSFETs, dead-time generator, shoot-through protection, and protection logic. The A3921 receives a PWM signal from the Linear Technology LTC6992CS6 and a direction bit from the micro controller. The received signal is processed by the A3921 and outputted to the H-bridge. The A3921 has two flag bits that are outputted to an I2C I/O expander. The microcontroller will check the status of the flag bits and make adjustments or corrections as needed. (MJH)

The final sheet of the motor driver board schematics is the H-bridge seen in Figure 33. Each quadrant of the H-bridge consists of three Infineon IRFS3006-7PPBF in parallel and a single Vishay VB30100S-E3/8W that will be used as a freewheeling diode. The gate from each quadrant of MOSFETs is driven by the Allegro A3921 seen in the H-bridge driver sheet. (MJH)

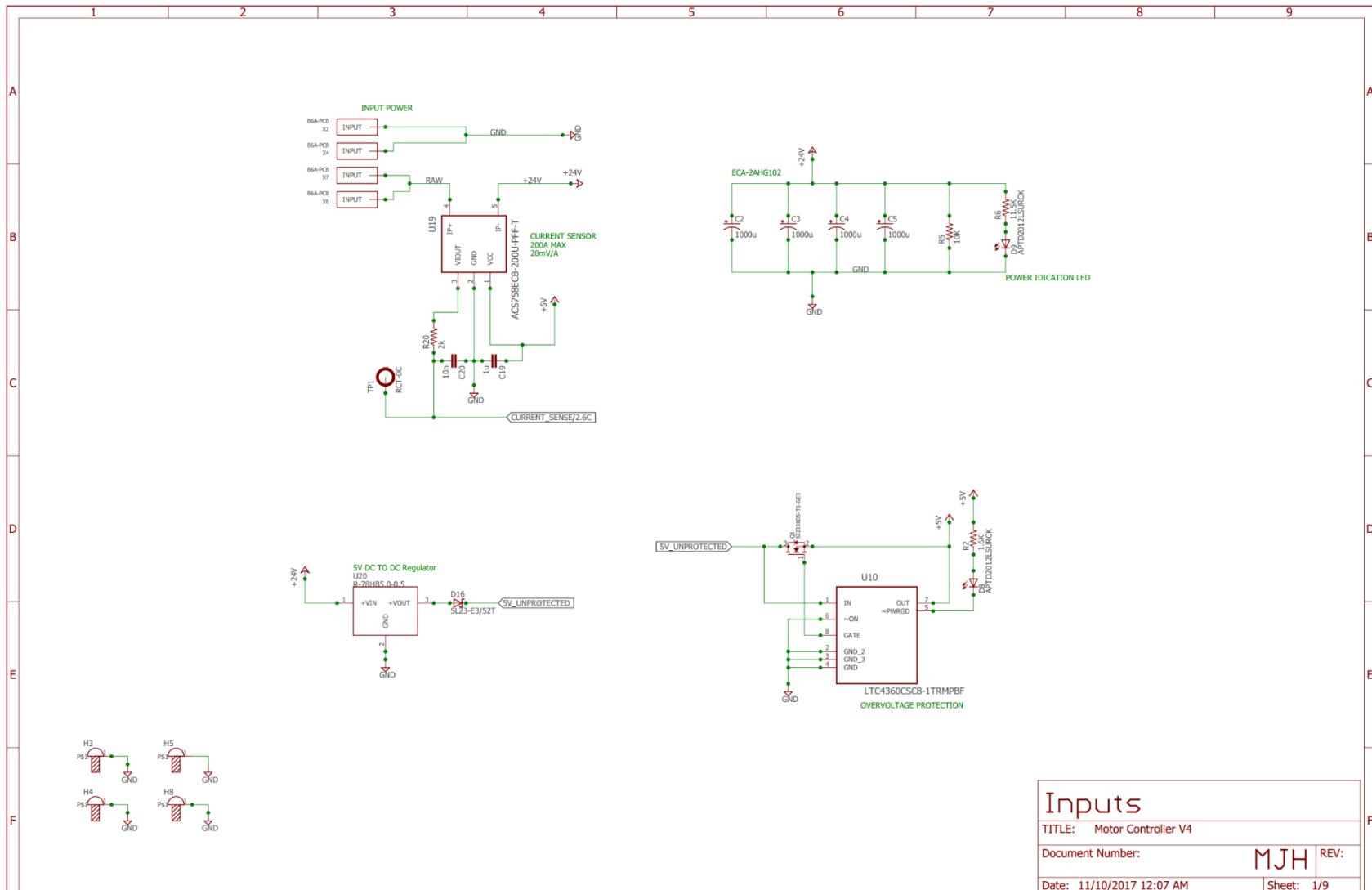


Figure 25: Motor Driver Board Inputs Schematic

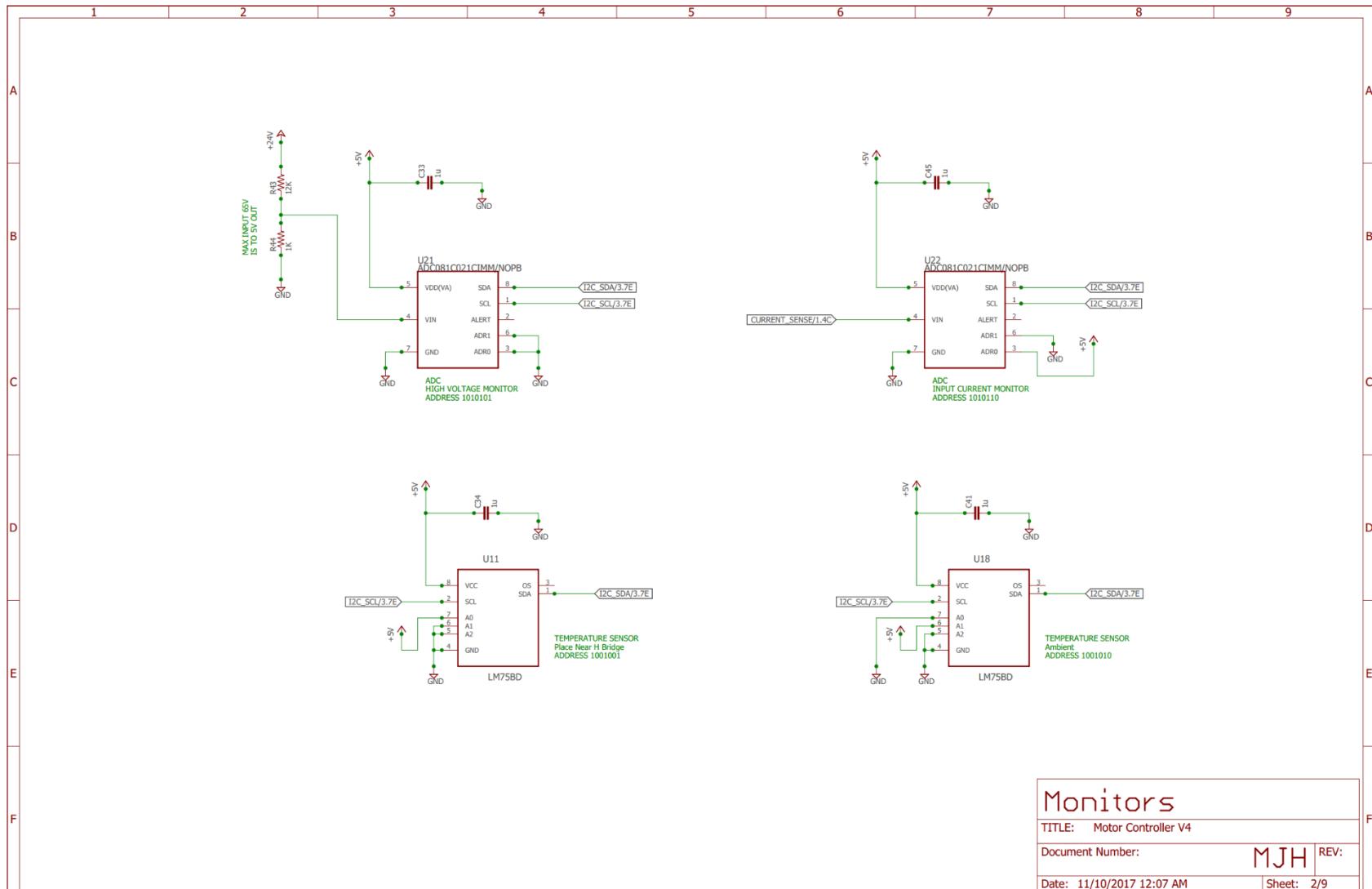


Figure 26: Motor Driver Board Monitors Schematic

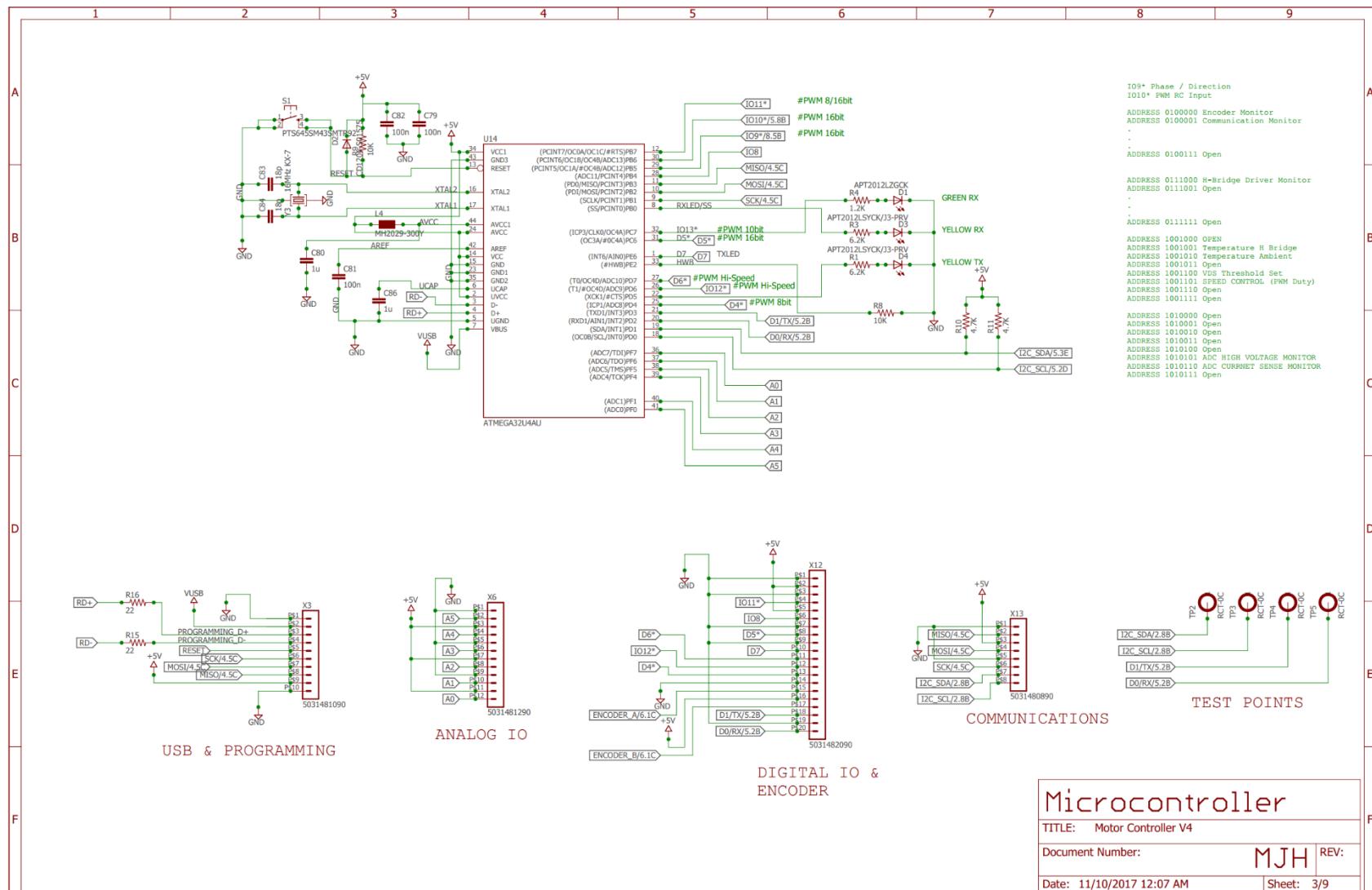


Figure 27: Motor Driver Board Microcontroller Schematic

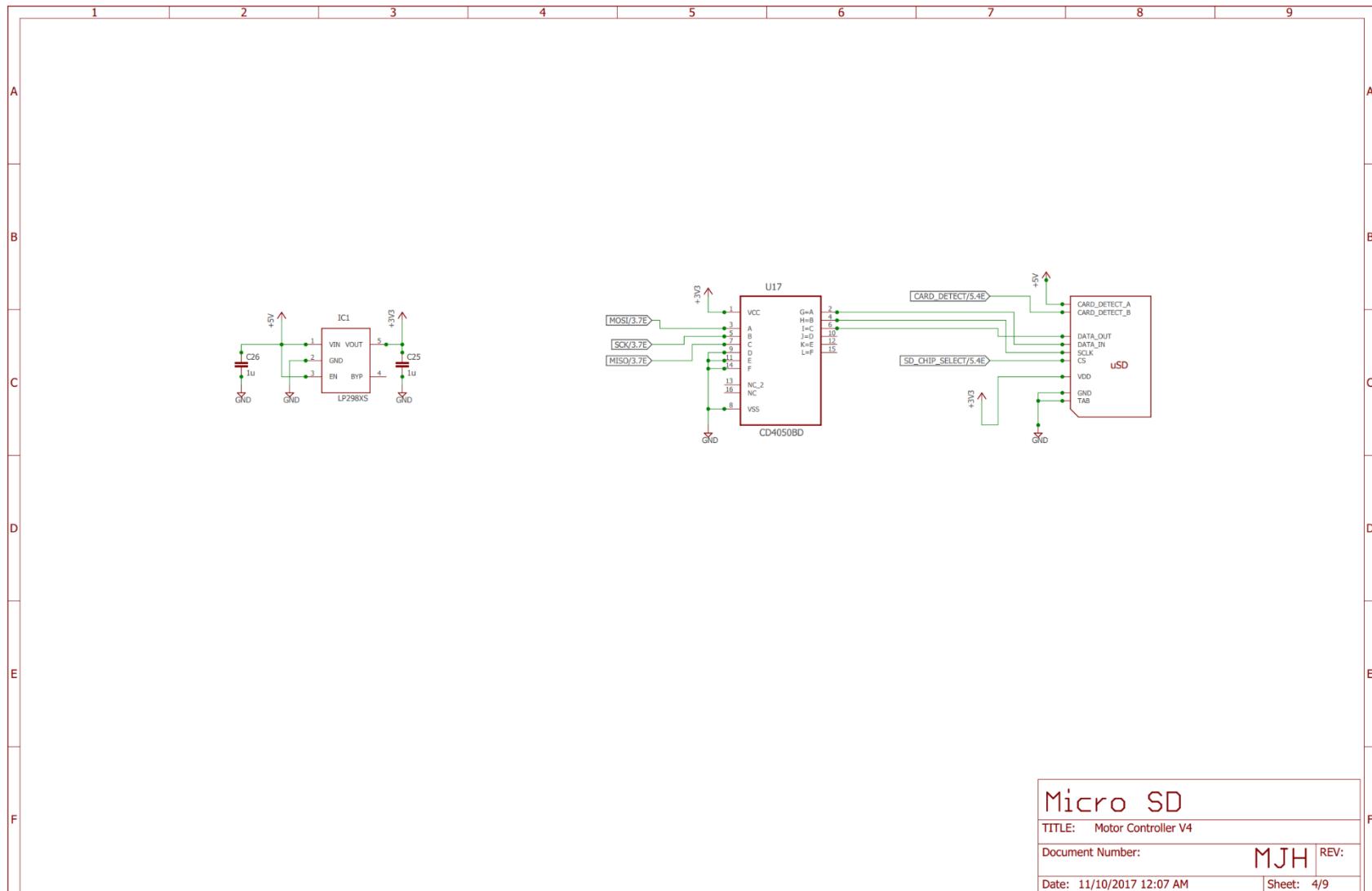


Figure 28: Motor Driver Board Micro SD Schematic

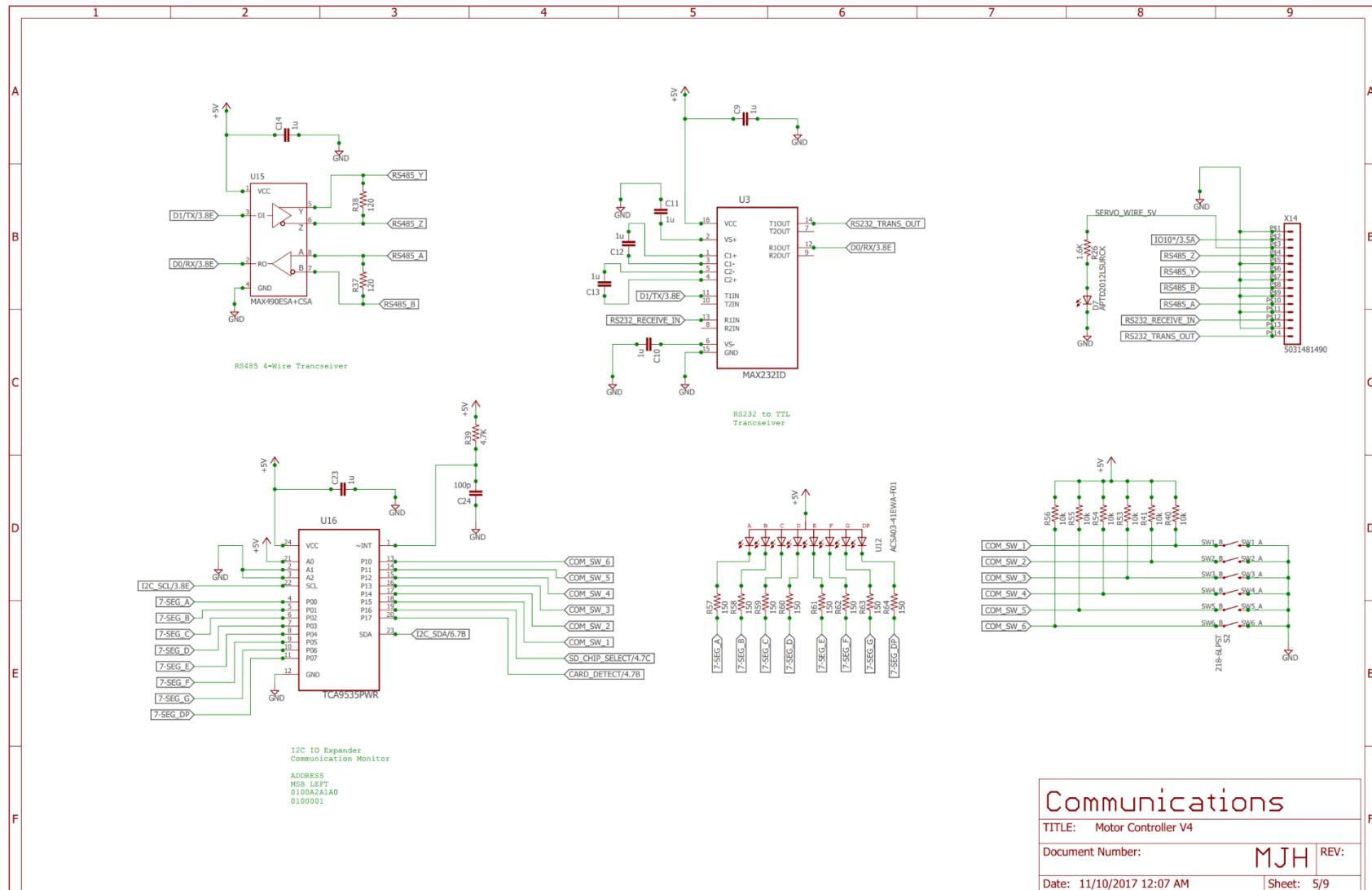


Figure 29: Motor Driver Board Communications Schematic

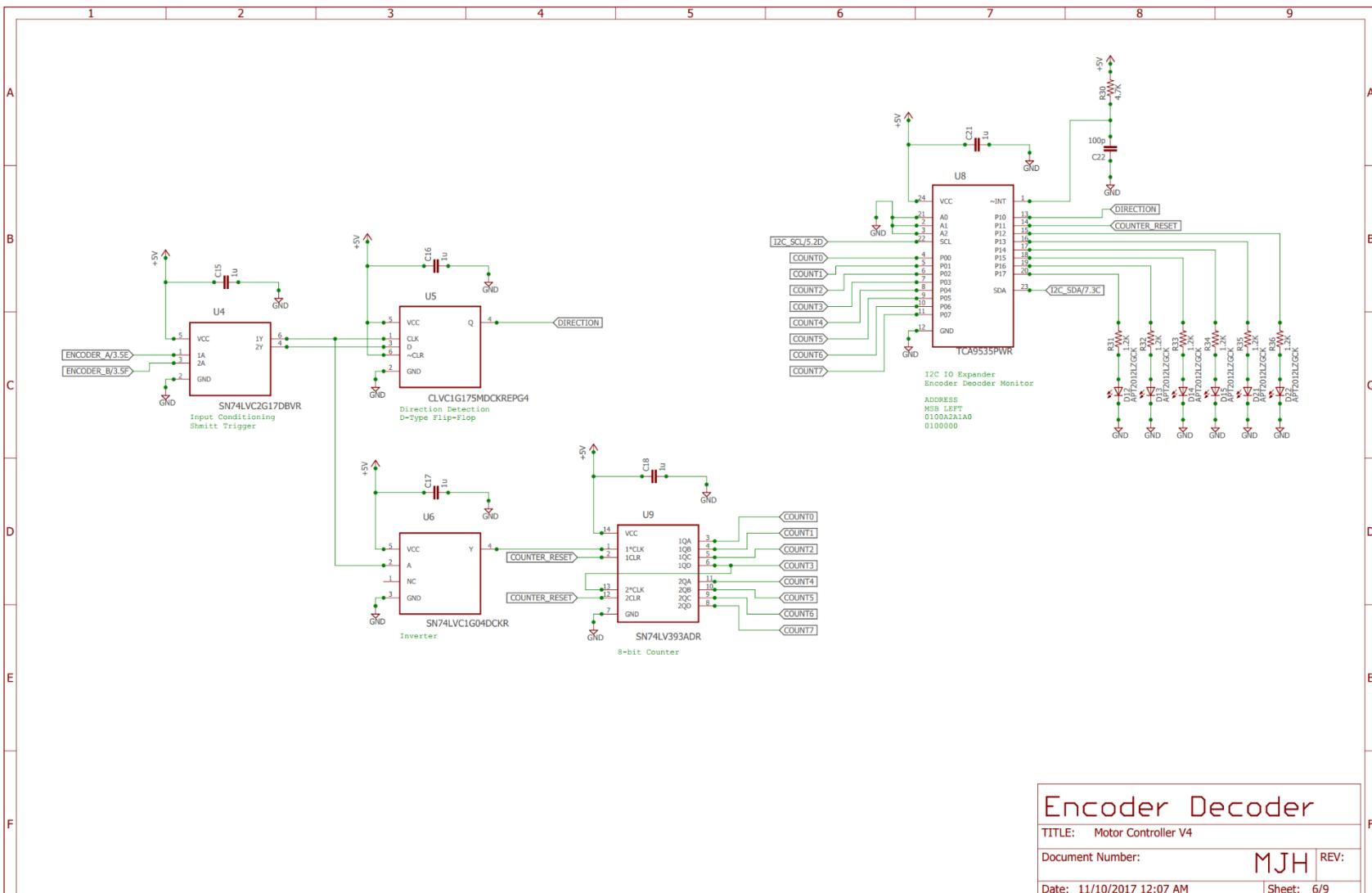


Figure 30: Motor Driver Board Encoder Decoder Schematic

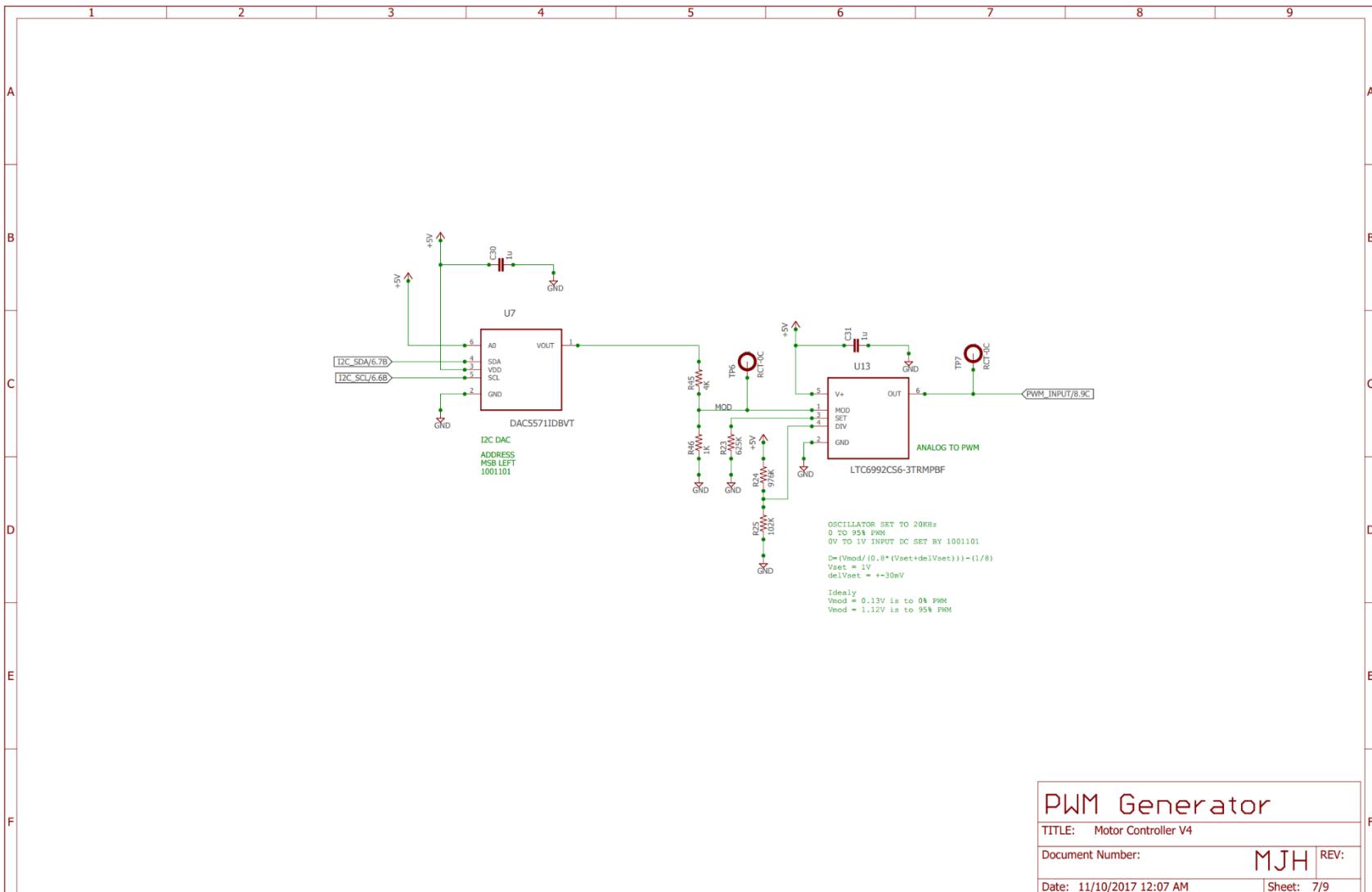


Figure 31: Motor Driver Board PWM Generator Schematic

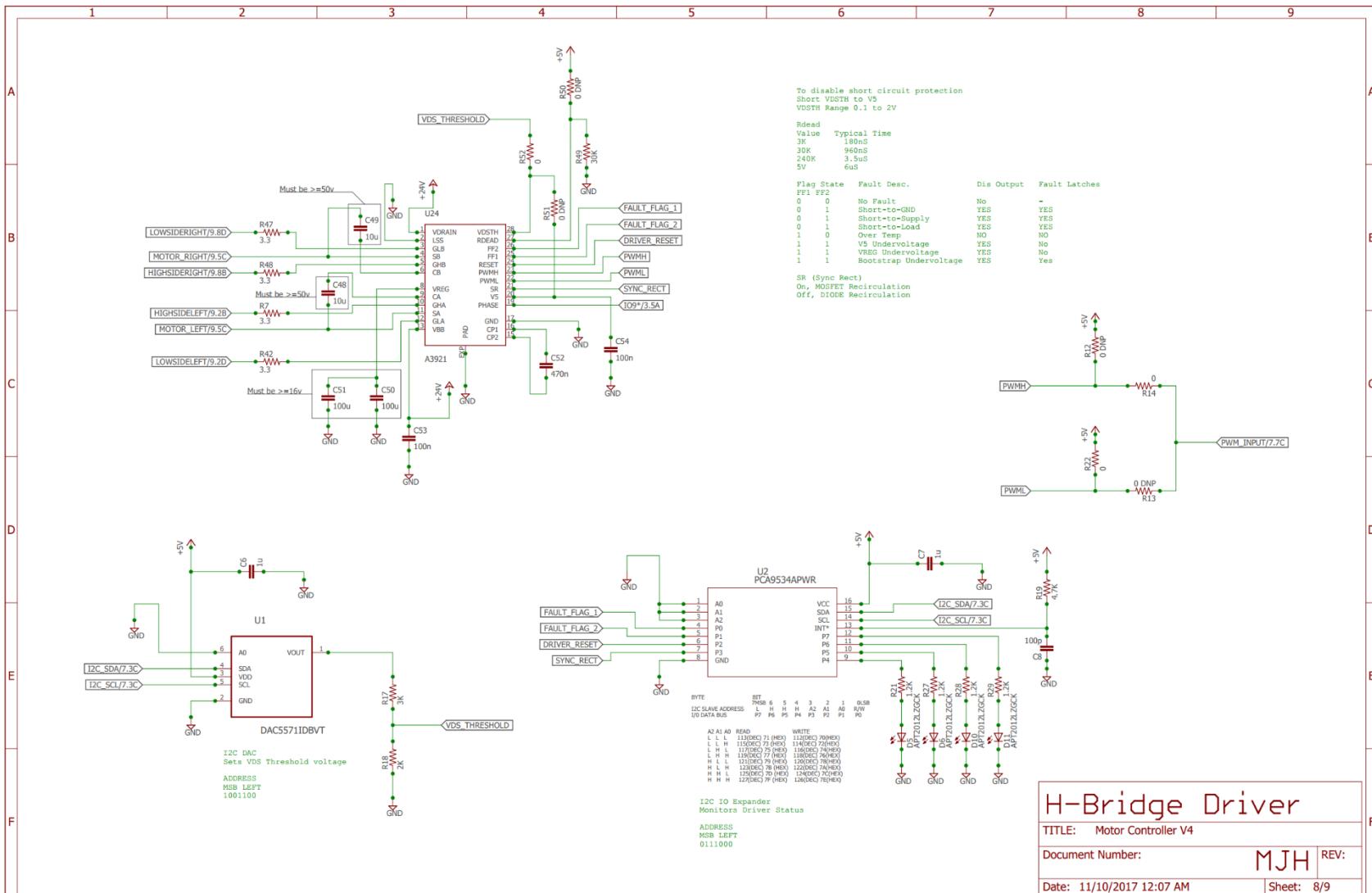


Figure 32: Motor Driver Board H-Bridge Driver Schematic

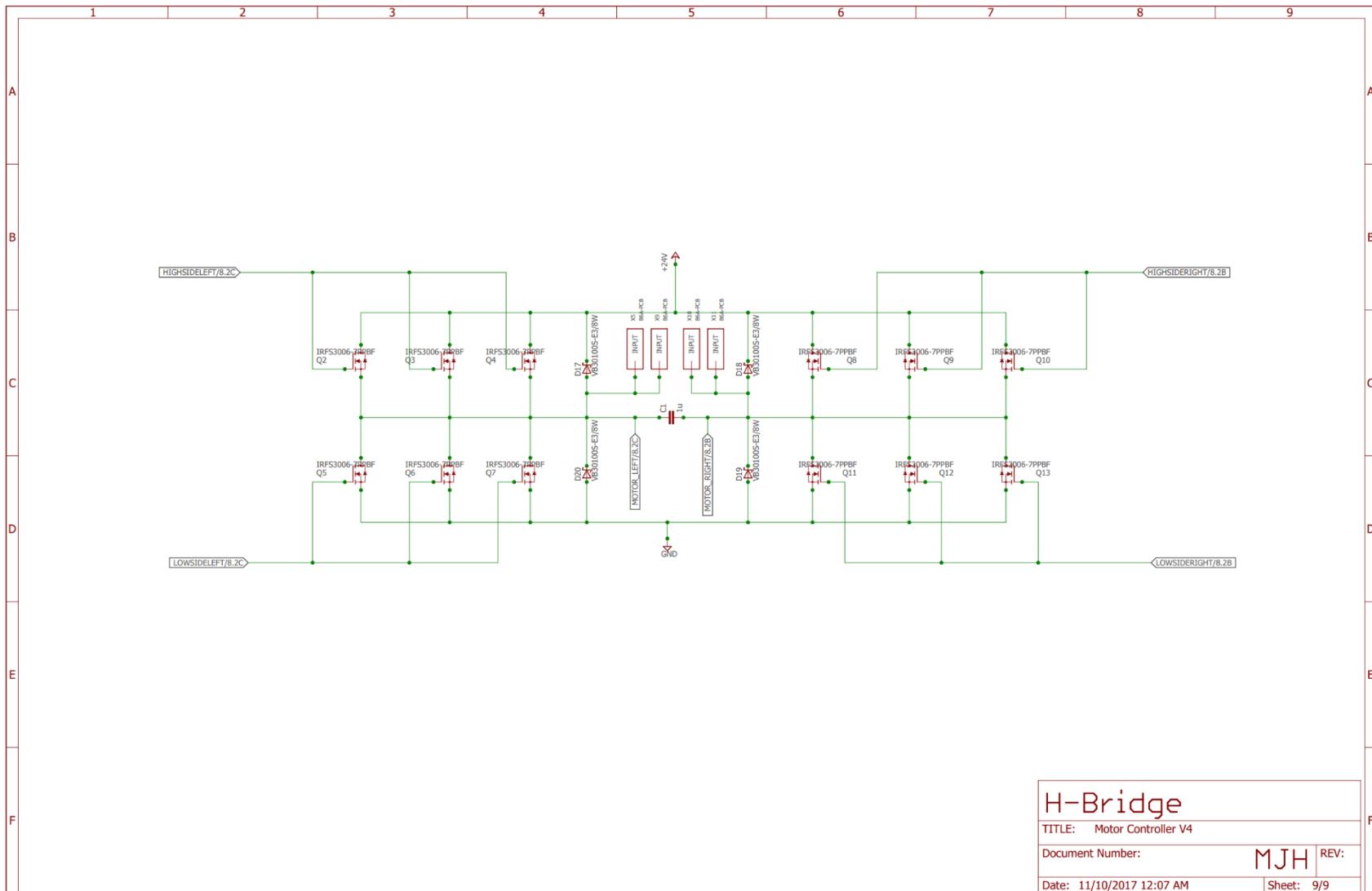


Figure 33: Motor Driver Board H-Bridge Schematic

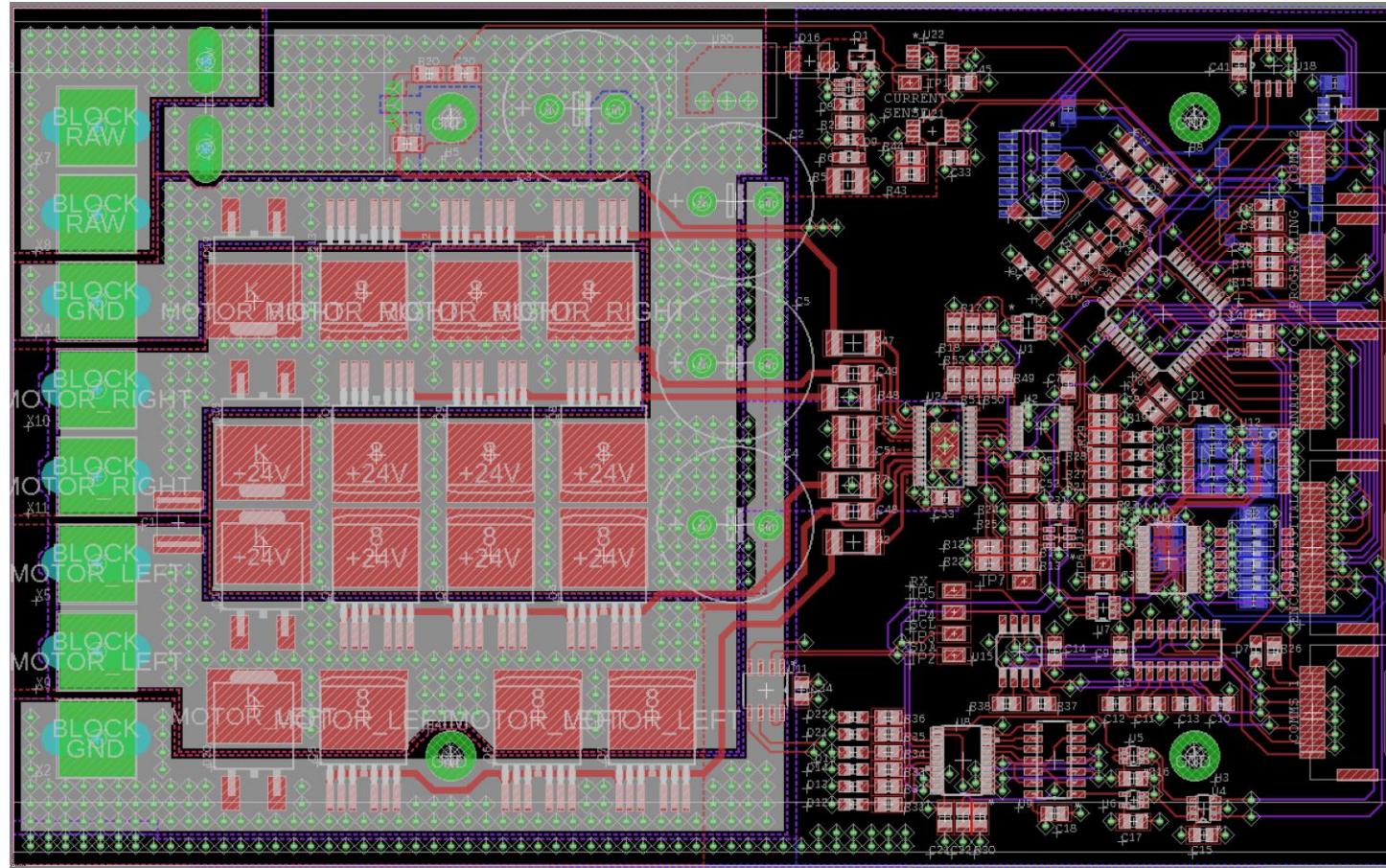


Figure 34: Motor Driver Board PCB Layout

3.3.4 Motor Driver Board Bill of Material

The bill of material in Table 51 lists the components needed to construct two of the motor driver boards. The reference designators in Table 51 correspond to the reference designators in the schematics of the motor driver board. (MJH)

Qty.	Refdes	Part Num.	Description
2	C1	CC2220KKX7R9BB105	CAP CER 1UF 50V X7R 2220
8	C2, C3, C4, C5	ECA-2AHG102	CAP ALUM 1000UF 20% 100V RADIAL
2	C20	CC0805KRX7R9BB103	CAP CER 10000PF 50V X7R 0805
4	C48, C49	GRM319R61E106KA12D	CAP CER 10UF 25V X5R 1206
4	C50, C51	CC1210MKX5R7BB107	CAP CER 100UF 16V X5R 1210
2	C52	C0805C474K5RACTU	CAP CER 0.47UF 50V X7R 0805
10	C53, C54, C79, C81, C82	08055C104KAT2A	CAP CER 0.1UF 50V X7R 0805
50	C6, C7, C9, C10, C11, C12, C13, C14, C15, C16, C17, C18, C19, C21, C23, C25, C26, C30, C31, C33, C34, C41, C45, C80, C86	CL21B105KAFNNNE	CAP CER 1UF 25V X7R 0805
6	C8, C22, C24	C0805C101J5GACTU	CAP CER 100PF 50V C0G/NP0 0805
4	C83, C84	CL21C180JBANNNC	CAP CER 18PF 50V C0G/NP0 0805
100	Clik-Mate Pins	502579-1000	1.5 WB CONN. PLUG TERMINAL 24-28
22	D1, D5, D6, D10, D11, D12, D13, D14, D15, D21, D22	APT2012LZGCK	LED GREEN CLEAR 0805 SMD
2	D16	SL23-E3/52T	DIODE SCHOTTKY 30V 2A DO214AA
8	D17, D18, D19, D20	VB30100S-E3/8W	DIODE SCHOTTKY 100V 30A TO263AB
2	D2	CD1206-S01575	DIODE GEN PURP 100V 150MA 1206
4	D3, D4	APT2012LSYCK/J3-PRV	LED YELLOW CLEAR 0805 SMD
6	D7, D8, D9	APTD2012LSURCK	LED RED CLEAR SMD
2	Enclosure	1455N1601BK	BOX ALUM BLACK 6.3" L X 4.06" W

2	Encoder Timming Pulley	A 6Z25M028DF0906	5 mm (HTD) Pitch,28 Teeth, 6mm Bore, Aluminum Insert, Polycarbonate Timing Pulley for 9mm Wide Belt
2	Heatsink	I010	3.500" WIDE EXTRUDED ALUMINUM HEATSINK 10" Length
2	IC1	LP2985-33DBVR	IC REG LINEAR 3.3V 150MA SOT23-5
2	L4	MH2029-300Y	SMD EMI Suppression Ferrite Beads
2	Motor Timming Pulley	A 6Z25-040DF0916	5 mm (HTD) Pitch,40 Teeth, 0.5" Bore, Aluminum Insert, Polycarbonate Timing Pulley for .354(9mm)" Wide Belt
2	PCB	Motor Controller V4	PCB of the Motor Controller Board 5x
2	Q1	SI2338DS-T1-GE3	MOSFET N-CH 30V 6A SOT23
24	Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12, Q13	IRFS3006-7PPBF	MOSFET N-CH 60V 240A D2PAK-7
4	R1, R3	RC0805JR-076K2L	RES SMD 6.2K OHM 5% 1/8W 0805
10	R10, R11, R19, R30, R39	RC0805FR-074K7L	RES SMD 4.7K OHM 1% 1/8W 0805
8	R12, R13, R50, R51	RC0805JR-070RL	RES SMD 0 OHM JUMPER 1/8W 0805
6	R14, R22, R52	RC0805JR-070RL	RES SMD 0 OHM JUMPER 1/8W 0805
4	R15, R16	RMCF0805JT22R0	RES SMD 22 OHM 5% 1/8W 0805
2	R17	RMCF0805JT3K00	RES SMD 3K OHM 5% 1/8W 0805
4	R18, R20	RMCF0805JT2K00	RES SMD 2K OHM 5% 1/8W 0805
4	R2, R26	RC0805JR-071K6L	RES SMD 1.6K OHM 5% 1/8W 0805
2	R23	RC0805FR-07620KL	RES SMD 620K OHM 1% 1/8W 0805
2	R24	RC0805FR-07976KL	RES SMD 976K OHM 1% 1/8W 0805
2	R25	RC0805FR-07102KL	RES SMD 102K OHM 1% 1/8W 0805
4	R37, R38	RMCF0805JT120R	RES SMD 120 OHM 5% 1/8W 0805
22	R4, R21, R27, R28, R29, R31, R32, R33, R34, R35, R36	RMCF0805JT1K20	RES SMD 1.2K OHM 5% 1/8W 0805
2	R43	RC0805JR-0712KL	RES SMD 12K OHM 5% 1/8W 0805
4	R44, R46	RC0805JR-071KL	RES SMD 1K OHM 5% 1/8W 0805
2	R45	RMCF0805JT3K90	RES SMD 3.9K OHM 5% 1/8W 0805
2	R49	RC0805JR-0730KL	RES SMD 30K OHM 5% 1/8W 0805
2	R5	ERJ-14YJ103U	RES SMD 10K OHM 5% 1/2W 1210
12	R57, R58, R59, R60, R61, R62, R63, R64	RMCF0805JT150R	RES SMD 150 OHM 5% 1/8W 0805
2	R6	RC0805FR-0711K5L	RES SMD 11.5K OHM 1% 1/8W 0805

8	R7, R42, R47, R48	CRCW20103R30FKEFHP	RES SMD 3.3 OHM 1% 1W 2010
18	R8, R9, R41, R40, R53, R54, R55, R56,	RC0805JR-0710KL	RES SMD 10K OHM 5% 1/8W 0805
2	Rotary Encoder	N.A.	Signswise 600p/r Incremental Rotary Encoder Dc5-24v Wide Voltage Power Supply 6mm Shaft
2	S1	PTS645SM43SMTR92LFS	SWITCH TACTILE SPST-NO 0.05A 12V
2	S2	218-6LPST	SWITCH SLIDE DIP SPST 25MA 24V
24	Thermal Pad	TG6050-10-10-2	THERMAL PAD 10X10X2MM
2	Timming Belt	A 6R25M065090	5 mm (HTD) Pitch, 65 Teeth, 9mm wide Single Sided Neoprene Belt with Fiberglass Cords
14	TP1, TP2, TP3, TP4, TP5, TP6, TP7	1625854-2	0805 PROBE PAD
4	U1, U7	DAC5571IDBVT	DAC
2	U10	LTC4360CSC8-1#TRMPBF	Ovvoltage Protection Controller
4	U11, U18	LM75BD	Digital temperature sensor and thermal watchdog
2	U12	ACSA03-41EWA-F01	DISPLAY 0.3" SGL 627NM RED SMD
2	U13	LTC6992CS6-3#TRMPBF	Voltage-Controlled Pulse Width Modulator (PWM)
2	U14	ATMEGA32U4-AU	IC MCU 8BIT 32KB FLASH 44TQFP
2	U15	MAX490ESA+	RS485 TRANSEIVER
2	U17	CD4050BDR	IC BUFF/CONVERTER HEX 16SOIC
2	U19	ACS758ECB-200U-PFF-T	SENSOR CURRENT HALL 200A DC
2	U2	PCA9534APWR	IC I/O EXPANDER I2C 8B 16TSSOP
2	U20	R-78HB5.0-0.5	CONV DC/DC 0.5A 5V OUT SIP VERT
4	U21, U22	ADC081C021CIMM/NOPB	ADC081C021CIMM/NOPB
2	U24	A3921KLPTR-T	IC FULL BRIDGE CTLR 28TSSOP
2	U3	MAX232ID	DUAL EIA-232 DRIVERS/ RECEIVERS
2	U4	SN74LVC2G17DBVR	DUAL SCHMITT-TRIGGER BUFFER
2	U5	CLVC1G175MDCKREP	SINGLE D-TYPE FLIP-FLOP WITH ASYNCHRONOUS CLEAR
2	U6	SN74LVC1G04DCKR	SINGLE INVERTER GATE
4	U8, U16	TCA9535PWR	I/O EXPANDER
2	U9	SN74LV393ADR	IC DUAL 4-BIT BIN CNTRS 14-SOIC

2	X1	1040310811	CONN MICRO SD CARD PUSH-PULL R/A
2	X12	503148-2090	Headers & Wire Housings 1.5 W/B Dual RA Rec 20Ckt EmbsTpPkgBeige
2	X12_Mate	503149-2000	CONN PLUG DUAL 20CKT BEIGE
2	X13	503148-0890	Headers & Wire Housings 1.5WB DUAL R/A EMBS TAPE PKG 8P
2	X13_Mate	503149-0800	CONN PLUG DUAL 8POS BEIGE
2	X14	503148-1490	Headers & Wire Housings 1.5WB DUAL R/A EMBS TAPE PKG 14P
2	X14_Mate	503149-1400	CONN PLUG DUAL 14POS BEIGE
16	X2, X4, X5, X7, X8, X9, X10, X11	B6A-PCB-HEX	6 AWG High AMP PCB Wire Lug 6-16 AWG
2	X3	503148-1090	Headers & Wire Housings 1.5 W/B Dual RA Rec 10Ckt EmbsTpPkgBeige
2	X6	503148-1290	Headers & Wire Housings 1.5 W/B Dual RA Rec 12Ckt EmbsTpPkgBeige
2	Y3	FA-238 16.0000MB-C3	CRYSTAL 16.0000MHZ 18PF SMD

Table 51: Motor Driver Board Bill of Material

3.3.5 Motor Driver Board Implementation Revisions

During the building process of the robot there were physical limitations due to the lack of space within the chassis. As a result of this space limitation the rotary encoders were not able to fit inside the robot with adequate room. Without the encoders the motor driver board was changed from a closed loop to an open loop controller. (MJH)

Other changes to the design include the RS232 transceiver chip not being populated. The RS232 transceiver was not needed because all communication to and from the MDB was handled by the RS485 transceiver. The intent of the RS232 transceiver in the original design was to allow for more implementation options while constructing the robot. The RS485 transceiver

worked successfully and thus the need for testing and implementing the RS232 transceiver was not needed. (MJH)

From the power calculation section, it was expected that cooling may be needed to manage the temperature of the MOSFETs during operation. This was the case and cooling was accomplished with a large heatsink and a fan that forced air through the fins of the heatsink. With the addition of the heatsink and fan the board never experienced thermal throttling issues and operated within a safe temperature range during the entire use of the board during competition. The board even exceeded operation expectations by destroying the windings of an Ampflow A25-150 motor. The MDB was still fully operational after the accidental destruction of the motor. The MDB with heatsink and fan can be seen in Figure 35 and Figure 36. (MJH)

The last change from the original design to the implanted design was the micro-SD card holder not being populated. This component was not populated because the population of the micro-SD card onto the PCB was not able to be done successfully with the tools and equipment provided. When attempting to populate the micro-SD card holder, the SD detect pin would short to the chassis of the card holder. This resulted in the 3.3v power railed that powers the SD card to be shorted to ground. To install the SD card holder successfully a solder-paste stencil and SMD reflow oven would be needed. With these tools the proper amount of solder paste could be applied and a reflow process without excess airflow could be used. (MJH)



Figure 35: MDB Completed Board Front

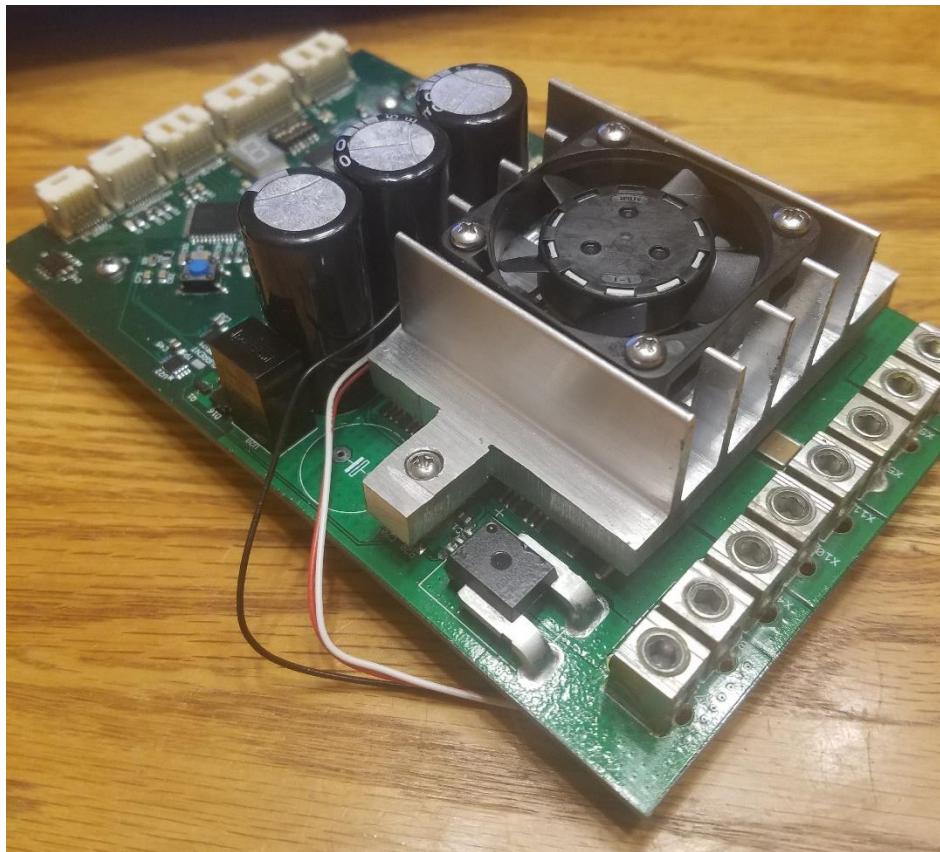


Figure 36: MDB Completed Board Back

3.3.6 Motor Driver Board Code

```
#include <FastTransfer.h>
FastTransfer Receive;
//FastTransfer Send;
#include "Wire.h"

#define IOEx_Encoder 0x20
#define IOEx_Switches 0x21
#define Temp_Sensor_HBridge 0x49
#define Temp_Sensor_Ambient 0x4A
#define HBridge_Monitor 0x38
#define VDS_Threshold 0x4C
#define Set_Speed_ADC 0x4D
#define High_Voltage_ADC 0x55
#define Current_Sense 0x56
#define Direction_Pin 9
#define Not_RS485_Switch_En 10

// 16 Port IO Expander Command Byte Map
byte input_port_0 = 0x00;
byte input_port_1 = 0x01;
byte output_port_0 = 0x02;
byte output_port_1 = 0x03;
```

```

byte config_port_0 = 0x06;
byte config_port_1 = 0x07;

//8 Port IO Expander Command Byte Map
byte inout_port = 0x00;
byte output_port = 0x01;
byte configuration = 0x03;

//Global Variables
int dt = 10;
int samples = 30;
int increment = 0;
int CIV = 1;
bool Rotation_Direction = true;
byte Set_Speed = 0;
byte Old_Set_Speed = 0;
int Serial_Read_Value = 0;
double Baud_Rate = 115200;
byte Number_Data_Points = 12;
int Parameter_Data[12];
// Max_Set_Speed re-written in assign_switch_settings() if operating in Safe-Speed Mode
byte Max_Set_Speed = 200;
byte Speed_Log[4];
byte Last_Speed = 0;

// PID Vararables
float kp = 0.3;
float ki = 0.4;
float kd = 0.3;
float I_Term = 0;
float PID_Speed = 0;
float Accumulator;
long PID_Error[10];

// Switch setting parameters
bool Fast_Transfer_Control = false;
bool Closed_Loop_Control = false;
bool Serial_Param_Dump = false;
bool Sync_Rect = false;
bool Local_FT_ID = false;
bool Babby_Cakes_Mode = false;

// Fast Transfer Variables
// FT_Local_Address re-written in assign_switch_settings()
byte FT_Local_Address = 69;
signed int FT_Receive_Array[3];
int FT_Align_Error;
int FT_Address_Error;
int FT_CRC_Error;
int FT_Data_Error;
unsigned long pastTime = 0;

//Seven-Segment Display Value Array
byte Seven_Seg[] = {0b10000010, 0b10111011, 0b10000101, 0b10010001,
0b10111000, 0b11010000, 0b11000000, 0b10011010, 0b10000000, 0b10011000};

```

```

void setup() {
    Wire.begin();
    Serial.begin(Baud_Rate); // USB Serial
    Serial1.begin(Baud_Rate); // Fast Tranfer Serial

    pinMode(Direction_Pin, OUTPUT);
    digitalWrite(Direction_Pin, HIGH);

    pinMode(LED_BUILTIN, OUTPUT);
    digitalWrite(LED_BUILTIN, LOW);

    pinMode(Not_RS485_Switch_En, OUTPUT);
    digitalWrite(Not_RS485_Switch_En, HIGH);

    // Configure the Encoder IO Expander
    Wire.beginTransmission(IOEx_Encoder);
    Wire.write(config_port_0);
    Wire.write(0xFF);
    Wire.endTransmission();

    Wire.beginTransmission(IOEx_Encoder);
    Wire.write(config_port_1);
    Wire.write(0x01);
    Wire.endTransmission();

    for (byte LED_Count = 0; LED_Count <= 63; LED_Count++) {
        write_LEDs(LED_Count);
        write_7_seg(map(LED_Count, 0, 63, 0, 9));
        delay(20);
    }
    write_LEDs(0);
    write_7_seg(0);

    // Configure the Switch IO Expander
    Wire.beginTransmission(IOEx_Switches);
    Wire.write(config_port_0);
    Wire.write(0x00);
    Wire.endTransmission();

    Wire.beginTransmission(IOEx_Switches);
    Wire.write(config_port_1);
    Wire.write(0xBF);
    Wire.endTransmission();

    //Configure H-Bridge Monitor IO Expander
    Wire.beginTransmission(HBridge_Monitor);
    Wire.write(configuration);
    Wire.write(0x03);
    Wire.endTransmission();

    //Set H-Bridge Monitor Dafault Parameters
    Wire.beginTransmission(HBridge_Monitor);
    Wire.write(output_port);
    Wire.write(0b11111011);
    Wire.endTransmission();
}

```

```

delay(250);

// The following lines should be taken care of in the function call
write_HBridge_monitor()
// verify operation before deleting
// if (Sync_Rect == true) {
//   Wire.beginTransmission(HBridge_Monitor);
//   Wire.write(output_port);
//   Wire.write(0b00000100);
//   Wire.endTransmission();
// }
// if (Sync_Rect == false) {
//   Wire.beginTransmission(HBridge_Monitor);
//   Wire.write(output_port);
//   Wire.write(0b00000100);
//   Wire.endTransmission();
// }

// Read the switch settings and apply them
assign_switch_settings();
//write_HBridge_monitor((read_Switch_Settings() << 4) | 0b00000100); // Force Reset Low
write_HBridge_monitor((read_Switch_Settings() << 4) & 0b11111011); // Force Reset Low

//Set VDS Threshold Voltage
byte VDS_Threshold_Value = 175;
byte VDS_Threshold_Value_MSB = VDS_Threshold_Value >> 4;
VDS_Threshold_Value_MSB = VDS_Threshold_Value_MSB & 0b00001111;
byte VDS_Threshold_Value_LSB = VDS_Threshold_Value << 4;
VDS_Threshold_Value_LSB = VDS_Threshold_Value_LSB & 0b11110000;
Wire.beginTransmission(VDS_Threshold);
Wire.write(VDS_Threshold_Value_MSB);
Wire.write(VDS_Threshold_Value_LSB);
Wire.endTransmission();

write_7_seg(0);
write_LEDs(0);

for (byte i = 0; i < 4; i++) {
  Speed_Log[i] = 0;
}

// Set up the fast transfer communication standards
Receive.begin(Details(FT_Receive_Array), FT_Local_Address, false,
&Serial1);
// Send.begin(Details(FT_Receive_Array), 3, false, &Serial1);
// Display the FT Local Adress
write_7_seg(floor(FT_Local_Address/10));
delay(250);
write_7_seg(FT_Local_Address % 10);
delay(250);
write_7_seg(0);
}

```

```

void loop() {
    ///////////////////////////////////////////////////////////////////
    ///////////////////////////////////////////////////////////////////
    /////////////////////////////////////////////////////////////////// Where is control data comming from
    ///////////////////////////////////////////////////////////////////
    ///////////////////////////////////////////////////////////////////
    ///////////////////////////////////////////////////////////////////
    // If operating in Fast Transfer Control Mode

    //unsigned long Top_Of_Loop_Time = micros();
    if (Fast_Transfer_Control) {
        //while((micros() < Top_Of_Loop_Time
        if (Receive.receiveData()) { // when ever a packet with the correct
address is received the verification is sent automatically
            digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
            Set_Speed = (byte) abs(FT_Receive_Array[0]);
            if (((signed int) FT_Receive_Array[0]) > 0) {
                Rotation_Direction = true;
            }
            if (((signed int) FT_Receive_Array[0]) < 0) {
                Rotation_Direction = false;
            }

            digitalWrite(Not_RS485_Switch_En, LOW);
            //while(!TXC1){
            Receive.ToSend(0, FT_Local_Address);
            Receive.sendData(1);
            //}
            delayMicroseconds(750);
            digitalWrite(Not_RS485_Switch_En, HIGH);

            write_7_seg(map(Set_Speed, 0, 255, 0, 9));
            Old_Set_Speed = Set_Speed;
        }
    }

    ///////////////////////////////////////////////////////////////////
    ///////////////////////////////////////////////////////////////////
    // If operating in Serial Control Mode
    if (!Fast_Transfer_Control) {
        if (Serial.available()) {
            int Received_Serial_Speed = Serial.read();
            if (Received_Serial_Speed > 0) {
                Rotation_Direction = true;
            }
            if (Received_Serial_Speed < 0) {
                Rotation_Direction = false;
            }
            Set_Speed = (byte) abs(Received_Serial_Speed);
            write_7_seg(map(Set_Speed, 0, 255, 0, 9));
            Old_Set_Speed = Set_Speed;
        }
    }

    ///////////////////////////////////////////////////////////////////
    ///////////////////////////////////////////////////////////////////
}

```

```

/////////// How should the speed of the motor be
controlled //////////
///////////
//if (Set_Speed != Old_Set_Speed) { // Only write a new speed if it dosent
match the current speed
    //////////
// If operating in closed loop mode
if (Closed_Loop_Control) {
    Get_PID_Error();
    Calculate_PID();
    write_speed((byte) PID_Speed, Rotation_Direction);
}
///////////
// If operating in open loop mode
if (!Closed_Loop_Control) {
    write_speed(Set_Speed, Rotation_Direction);
}
//}
/////////
///////// Debugging operations
/////////
/////////
// If it is desired to dump all the serial parameter data
if (Serial_Param_Dump) {
    Increment_Counter();
    if ((increment % 3) == 0) {
        //if(Serial.available()){
        prep_parameter_data();
        for (int i = 0; i < Number_Data_Points; i++) {
            Serial.write(Parameter_Data[i]);
        }
    //}
}
}

// Send.receiveData();
// Send.ToSend(map(increment,0,9,0,3), increment);
// Send.ToSend(0, increment);
// Send.sendData(4);
//Serial.println(Serial_Param_Dump);
// digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
// Increment_Counter();
// Serial.print(read_Switch_Settings(), BIN);
// Serial.print(" ");
// Serial.print(Fast_Transfer_Control);
// Serial.print(" ");
// Serial.print(Closed_Loop_Control);
// Serial.print(" ");
// Serial.println(Serial_Param_Dump);
// Serial.println(read_HV_Sensor());
// delay(dt);

```



```

        CIV = 1;
    }
}

///////////////////////////////
/////////////////////////////
void write_speed(byte ADC_Voltage, bool Rotation_Direction) {
    // Function: write_speed()
    // Set the desired speed by writing to the PWM Generators ADC
    // Max range 0% to 95% PWM
    // Ideally the following, needs verified
    // 0% => 0.13V
    // 95% => 1.12V

    byte Min_Write_Speed = 25;

    // Box car filter to smooth the speed
    // for(byte i=9; i>0; i--){
    //     Speed_Log[i] = Speed_Log[i-1];
    // }
    // Speed_Log[0] = ADC_Voltage;
    // unsigned int Speed_Sum = 0;
    // for(byte i=0; i<10; i--){
    //     Speed_Sum = Speed_Sum + Speed_Log[i];
    // }
    // ADC_Voltage = Speed_Sum / 10;

    // for(byte i = (2-1); i > 0; i--){
    //     Speed_Log[i] = Speed_Log[i-1];
    // }
    // Speed_Log[0] = ADC_Voltage;
    // unsigned int Speed_Sum = 0;
    // for(byte i = 0; i<2; i++){
    //     Speed_Sum += Speed_Log[i];
    // }
    // ADC_Voltage = Speed_Sum >> 1;

    // Set the writing speed such that a 1 will begin movement
    if ((ADC_Voltage > 0) & (ADC_Voltage < (Max_Set_Speed - Min_Write_Speed))) {
        ADC_Voltage = ADC_Voltage + Min_Write_Speed;
    }
    // Limit the speed to the Maximum allowable speed
    if (ADC_Voltage >= Max_Set_Speed) {
        ADC_Voltage = Max_Set_Speed;
    }

    // Prep the byte to be written to the DAC
    byte ADC_Voltage_MSB = ADC_Voltage >> 4;
    ADC_Voltage_MSB = ADC_Voltage_MSB & 0b00000111;
    byte ADC_Voltage_LSB = ADC_Voltage << 4;
    ADC_Voltage_LSB = ADC_Voltage_LSB & 0b11110000;

    // Set the rotation direction
}

```

```

    if (Rotation_Direction) {
        digitalWrite(Direction_Pin, HIGH);
    }
    if (!Rotation_Direction) {
        digitalWrite(Direction_Pin, LOW);
    }

    // Write the speed
    Wire.beginTransmission(Set_Speed_ADC);
    Wire.write(ADC_Voltage_MSB);
    Wire.write(ADC_Voltage_LSB);
    Wire.endTransmission();
}

///////////////////////////////
///////////////////////////////

byte Check_HBridge_Flags() {
    // Function: Check_HBridge_Flags()
    // Returns flags from the H-Bridge Pre-Driver IC
    // 00 No Faults
    // 01 Short to GND
    // 01 Short to Supply
    // 01 Short to Load
    // 10 Over Temp
    // 11 V5 Undervoltage
    // 11 VREG Undervoltage
    // 11 Bootstrap Undervoltage
    Wire.requestFrom(HBridge_Monitor, 1);
    byte flags = Wire.read();
    Wire.endTransmission();
    flags = flags & 0b00000011;
    return flags;
}

///////////////////////////////
///////////////////////////////

void write_HBridge_monitor(byte write_values) {
    // Function: write_HBridge_monitor()
    // Write to the H-Bridge Monitor IO Expander
    // b7 - b4 => LEDs
    // b3 Synchronous Rectification Enable
    // b2 Driver Reset
    write_values = write_values & 0b11110000;
    write_values = write_values | 0b00000100; // Reset is active LOW
    //write_values = write_values & 0b11111011; // Reset is active LOW
    if (Sync_Rect == true) {
        write_values = write_values | 0b00001000;
    }
    if (Sync_Rect == false) {
        write_values = write_values & 0b11110111;
    }
    Wire.beginTransmission(HBridge_Monitor);
    Wire.write(output_port);
    Wire.write(write_values);
    Wire.endTransmission();
}

```

```

}

///////////
///////////
void write_LEDs(byte display_value) {
    // Function: write_LEDs()
    // Write to the LEDs of the IO Expander used for with the Endocoder Decoder
    // The IO Expander has 2 ports, port-1 has LEDS on b7-b2
    // Port-2 b1 => Counter Reset
    // Port-2 b0 => Direction of motor rotation
    display_value = display_value << 2;
    display_value = display_value & 0b11111100;
    Wire.beginTransmission(IOEx_Encoder);
    Wire.write(output_port_1);
    Wire.write(display_value);
    Wire.endTransmission();
}

///////////
///////////

bool check_motor_direction() {
    // Function: check_motor_direction()
    // Check the direction of rotation of the motor
    // Return a 1 or 0 indicating direction of rotation
    Wire.beginTransmission(IOEx_Encoder);
    Wire.write(input_port_1);
    Wire.endTransmission(false); // Repeated Start Condition
    Wire.requestFrom(IOEx_Encoder, 1);
    byte Measured_Rotation_Direction = Wire.read();
    Wire.endTransmission();
    return Measured_Rotation_Direction;
}

///////////
///////////

byte sample_encoder_count() {
    // Function: sample_encoder_count()
    // Check the couunter to check how many times the motor have revolved
    // return the count from the 8-bit counter
    // The IO Expander has 2 ports, port0 has the 8-bit counter
    // Sample the Encoder Count
    Wire.beginTransmission(IOEx_Encoder);
    Wire.write(input_port_0);
    Wire.endTransmission(false); // Repeated Start Condition
    Wire.requestFrom(IOEx_Encoder, 1);
    byte Encoder_Count = Wire.read();
    Wire.endTransmission();
    //reset_encoder_coutner();
    return Encoder_Count;
}

///////////
///////////

```

```

void reset_encoder_coutner() {
    // Function: reset_encoder_coutner()
    // Clear the count of the encoder counter
    // First check the state of the LEDs
    Wire.beginTransmission(IOEx_Encoder);
    Wire.write(output_port_1);
    Wire.endTransmission(false); // Repeated Start Condition
    Wire.requestFrom(IOEx_Encoder, 1);
    byte Current_LED_State = Wire.read();
    Wire.endTransmission();
    // Reset the IO expander and write the LEDs
    Wire.beginTransmission(IOEx_Encoder);
    Wire.write(output_port_1);
    Wire.write(Current_LED_State | 0b00000010); // Set Reset HIGH clearing the
counter
    Wire.endTransmission();
    // Set reset LOW to begin counting again
    Wire.beginTransmission(IOEx_Encoder);
    Wire.write(output_port_1);
    Wire.write(Current_LED_State & 0b11111101); // Set Reset Low to begin
counting again
    Wire.endTransmission();
}

///////////////////////////////
/////////////////////////////
////////////////////////////

byte check_rotation_speed() {
    // Function: check_rotation_speed();
    // Check the speed of rotation of the motor by sampling the encoder
counter
    // Returned byte is proportional to the speed of rotation
    unsigned int max_motor_rpm = 6000;
    unsigned int pulses_per_revolution = 600;
    unsigned long micro_sec_per_min = 1000000;
    byte pulse_count_min = 5;
    unsigned long max_aquisition_time = 25000;
    byte pulse_count = 0;

    reset_encoder_coutner();
    unsigned long start_time = micros();
    // Set a minimum required pulse count so that at low RPM the measured RPM
is still accurate
    while ((pulse_count < pulse_count_min) and ((micros() - start_time) <
max_aquisition_time)) {
        pulse_count = sample_encoder_count();
    }
    unsigned long stop_time = micros();

    unsigned long measured_RPM = pulse_count * (60000000 /
pulses_per_revolution);
    measured_RPM = measured_RPM / (stop_time - start_time);

    if (measured_RPM > max_motor_rpm) {
        measured_RPM = max_motor_rpm;
    }
    return map(measured_RPM, 0, max_motor_rpm, 0, 255);
}

```

```

}

///////////////////////////////
/////////////////////////////
void write_7_seg(byte display_value) {
    // Function: write_7_seg()
    // Write the numeric value to display on the 7-Segment Display
    // The IO Expander has 2 ports, port-1 is used for the display
    // See the array Seven_Seg[] for the binary breakdown
    // Each element in the array corresponds to the actual value
    // Seven_Seg[0] => 0, Seven_Seg[1] => 1, ...
    Wire.beginTransmission(IOEx_Switches);
    Wire.write(output_port_0);
    Wire.write(Seven_Seg[display_value]);
    Wire.endTransmission();
}

/////////////////////////////
/////////////////////////////

int readTempSensor(int Sensor) {
    // Function: readTempSensor()
    // Read the temperature of either temperature sensor
    // Must be passed the value Temp_Sensor_HBridge or Temp_Sensor_Ambient
    // Return the temperature in degrees C
    Wire.requestFrom(Sensor, 2);
    int temp_MSB = Wire.read();
    int temp_LSB = Wire.read();
    Wire.endTransmission();

    temp_MSB = temp_MSB << 3;
    temp_LSB = temp_LSB >> 5;
    int temp_store = temp_MSB | temp_LSB;
    temp_store = temp_store & 0x3FF; // Remove the Sign Bit
    temp_store = temp_store * 0.125; // Datasheet required scalar
    return temp_store;
}

/////////////////////////////
/////////////////////////////

byte read_HV_Sensor() {
    // Function: read_HV_Sensor()
    // Read the voltage on the 24V Rail
    // Returns the actual voltage read
    Wire.requestFrom(High_Voltage_ADC, 2);
    byte HVRead_MSB = Wire.read();
    byte HVRead_LSB = Wire.read();
    Wire.endTransmission();
    HVRead_MSB = HVRead_MSB << 4;
    HVRead_LSB = HVRead_LSB >> 4;
    byte HVRead = HVRead_MSB | HVRead_LSB;
    HVRead = map(HVRead, 0, 255, 0, 65);
    return HVRead;
}

```

```

///////////
///////////

byte read_Current_Sensor() {
    // Function: read_Current_Sensor()
    // Read the voltage from the HV input current sensor
    // Return the actual current measured
    Wire.requestFrom(Current_Sense, 2);
    byte CSRead_MSB = Wire.read();
    byte CSRead_LSB = Wire.read();
    Wire.endTransmission();
    CSRead_MSB = CSRead_MSB << 4;
    CSRead_LSB = CSRead_LSB >> 4;
    byte CSRead = CSRead_MSB | CSRead_LSB;
    // There is a 0.5V to 0.6V bias on the output
    // Subtracting 31 corrects this
    if (CSRead > 31) {
        CSRead = CSRead - 31;
    }
    else {
        CSRead = 0;
    }
    // Note: 8-bit ADC => 5V/256=0.0195V/Div~0.02V/Div
    //The Current Sensor outputs 0.02V/A therefore no mapping needed
    CSRead = map(CSRead, 0, 255, 0, 250);
    return CSRead;
}

///////////
///////////

byte read_Switch_Settings() {
    // Function: read_Switch_Settings()
    // Read the Switch Settings of the DIP Switch
    // Return the position of the 6 switches
    // b5=>SW1 b4=>SW2 b3=>SW3 b2=>SW4 b1=>SW5 b0=>SW6
    Wire.beginTransmission(IOEx_Switches);
    Wire.write(input_port_1);
    Wire.endTransmission(false); // Repeated Start Condition
    Wire.requestFrom(IOEx_Switches, 1);
    byte SwitchSettings = Wire.read();
    Wire.endTransmission();
    SwitchSettings = SwitchSettings & 0b00111111;
    return SwitchSettings;
}

void prep_parameter_data() {
    // Function: prep_paramter_data()
    // Gathers all the parapeter data for analysis
    // Sotres values into an array
    Parameter_Data[0] = (readTempSensor(Temp_Sensor_Ambient) * 1.8) + 32;
    Parameter_Data[1] = (readTempSensor(Temp_Sensor_HBridge) * 1.8) + 32;
    Parameter_Data[2] = read_HV_Sensor();
    Parameter_Data[3] = read_Current_Sensor();
    Parameter_Data[4] = increment;
    Parameter_Data[5] = check_rotation_speed();
}

```

```

Parameter_Data[6] = Set_Speed;
Parameter_Data[7] = (byte) PID_Speed;
Parameter_Data[8] = Fast_Transfer_Control;
Parameter_Data[9] = Closed_Loop_Control;
Parameter_Data[10] = Serial_Param_Dump;
Parameter_Data[11] = Accumulator;
}
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////// PID Functions
/////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////// http://blog.solutions-cubed.com/pid-motor-control-with-an-arduino/
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////

void Calculate_PID(void) {
    Accumulator += PID_Error[0];

    PID_Speed = PID_Error[0] * kp;
    PID_Speed += ki * Accumulator;
    PID_Speed += kd * (PID_Error[0] - PID_Error[9]);

    if (PID_Speed >= 255) {
        PID_Speed = 255;
    }
    if (PID_Speed <= 0) {
        PID_Speed = 0;
    }

    if (PID_Speed >= Max_Set_Speed) {
        PID_Speed = Max_Set_Speed;
    }
    // if(PID_Speed >= 127){
    //     PID_Speed = 127;
    // }
    // if(PID_Speed <= -126){
    //     PID_Speed = -126;
    // }
    // PID_Speed = PID_Speed +127;
}

void Get_PID_Error() {
    byte i = 0;
    byte Actual_Speed = check_rotation_speed();
    byte Desired_Speed = Set_Speed;

    for (i = 9; i > 0; i--) {
        PID_Error[i] = PID_Error[i - 1];
    }

    PID_Error[0] = Desired_Speed - Actual_Speed;
}

```

(MJH)

3.4.0 Proximity Sensor Board

The proximity sensor board utilizes LIDAR sensors to read the environment and store the environment data in the board's microcontroller. The LIDAR sensors will be triggered to read at the appropriate time by looking at the rising edge of the hall effect sensor output. A total of 8 PSBs will be installed in the combat robot to maximize enemy detection rate. Each board will communicate to the MCB and exchange all environment data upon request. (MPH)

3.4.1 Proximity Sensor Board Hardware

Figure 37 shows a level 1 block diagram of how the PSB will function. There will be 7 total linear regulators (1 for each PSB) to step the 9 VDC down to 5 VDC and effectively smooth out noise from the switching regulator. The microcontroller will take data inputs from the LIDAR sensor and hall effect sensor. The microcontroller will then send data to the MCB upon request. (MPH)

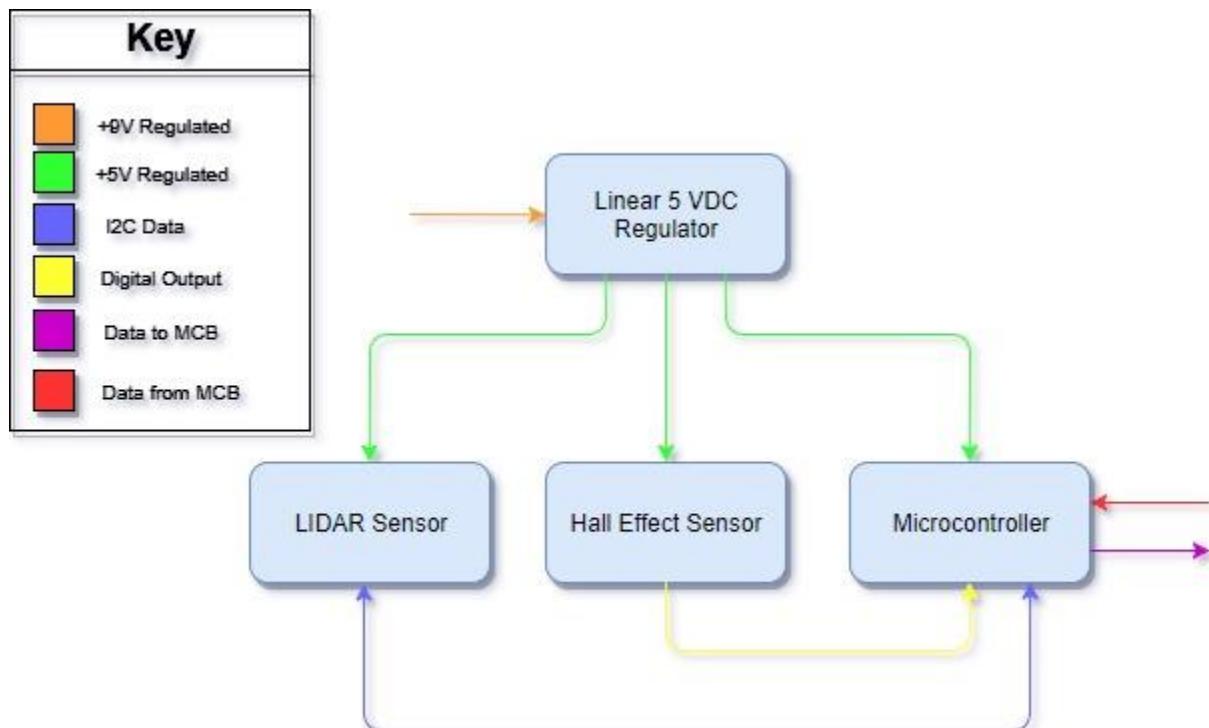


Figure 37: Level 1 Proximity Sensor Board Hardware Block Diagram

Module	Linear 5 VDC Regulator
Designer	Michael P. Hritz
Inputs	Power: Regulated 9 VDC
Outputs	Power: Stepped Down Regulated 5 VDC (noise reduced)
Description	The linear voltage regulator will step down the 9 VDC to 5 VDC and supply power to the LIDAR sensors, hall effect sensors, and microcontrollers at each satellite location.

Table 52: Level 2 Proximity Sensor Board Hardware Linear 5 VDC Regulator Module

Module	LIDAR Sensor
Designer	Off the Shelf Solution (MPH)
Inputs	Power: Regulated 5 VDC Signal: I2C Measurement Trigger
Outputs	Signal: I2C Object Detection Distance Measurements
Description	The LIDAR sensor will be powered by the linear 5 VDC regulators (1 at each satellite location) and will output I2C distance measurements when triggered by the microcontroller.

Table 53: Level 2 Proximity Sensor Board Hardware LIDAR Sensor Module

Module	Hall Effect Sensor
Designer	Off the Shelf Solution (MPH)
Inputs	Power: Regulated 5 VDC
Outputs	Signal: Analog Voltage
Description	The hall effect sensors (1 at each satellite location) will detect the presence of a magnetic field when the weapon tool is in position for the LIDAR sensor to take an accurate reading.

Table 54: Level 2 Proximity Sensor Board Hardware Hall Effect Sensor Module

Module	Microcontroller
Designer	Michael P. Hritz
Inputs	Power: Regulated 5 VDC Signal: I2C LIDAR Data Signal: Analog Voltage from Hall Effect Sensors Signal: I2C Data Request from MCB
Outputs	Signal: I2C Distance Measurement Data Signal: I2C Measurement Trigger
Description	The microcontrollers will determine when to trigger each LIDAR sensor to take a distance measurement. The trigger will be the input signal from the hall effect sensor detecting a magnetic field when the weapon tool is in position. The microcontroller will store all distance data and transmit the data upon request to the MCB.

Table 55: Level 2 Proximity Sensor Board Hardware Microcontroller Module

3.4.2 Proximity Sensor Board Schematic

Figure 38 shows all inputs and outputs for each PSB. Each PSB will have a linear 5 VDC regulator (IC1) to step down the 9 VDC input from the MCB down to 5 VDC. There will be two sensor input ports (X1 and X3) that receive data from the hall effect sensors and LIDAR sensors and also transmit data to the sensors. The ports were designed to be identical in terms of their pins to avoid connecting a sensor to the wrong port. The communication port (X4) will send data to the MCB and also receive data from the MCB through RS485 communication protocol. The communication port also delivers 9 VDC to the linear regulator on each PSB.

Figure 39 shows the transceiver (U2) that will be used to communicate data to the MCB. The RS485 transceiver (U2) will be the primary transceiver used for communication because it has noise immunity due to the differential inputs.

Figure 40 shows the microcontroller circuit that will be used for each PSB. A programming port (X5) will be used to program the microcontroller through ICSP (in-circuit serial programming). The microcontroller will have the ability to be reset with a push button (S2) for debugging purposes. Five LEDs will be used (D1, D2, D3, D4, and D5) for debugging purposes. Each microcontroller will have a dip switch (S1) to set the slave address. The I2C bus (pins RB5 and RB7) will be used to communicate with each LIDAR sensor. (MPH)

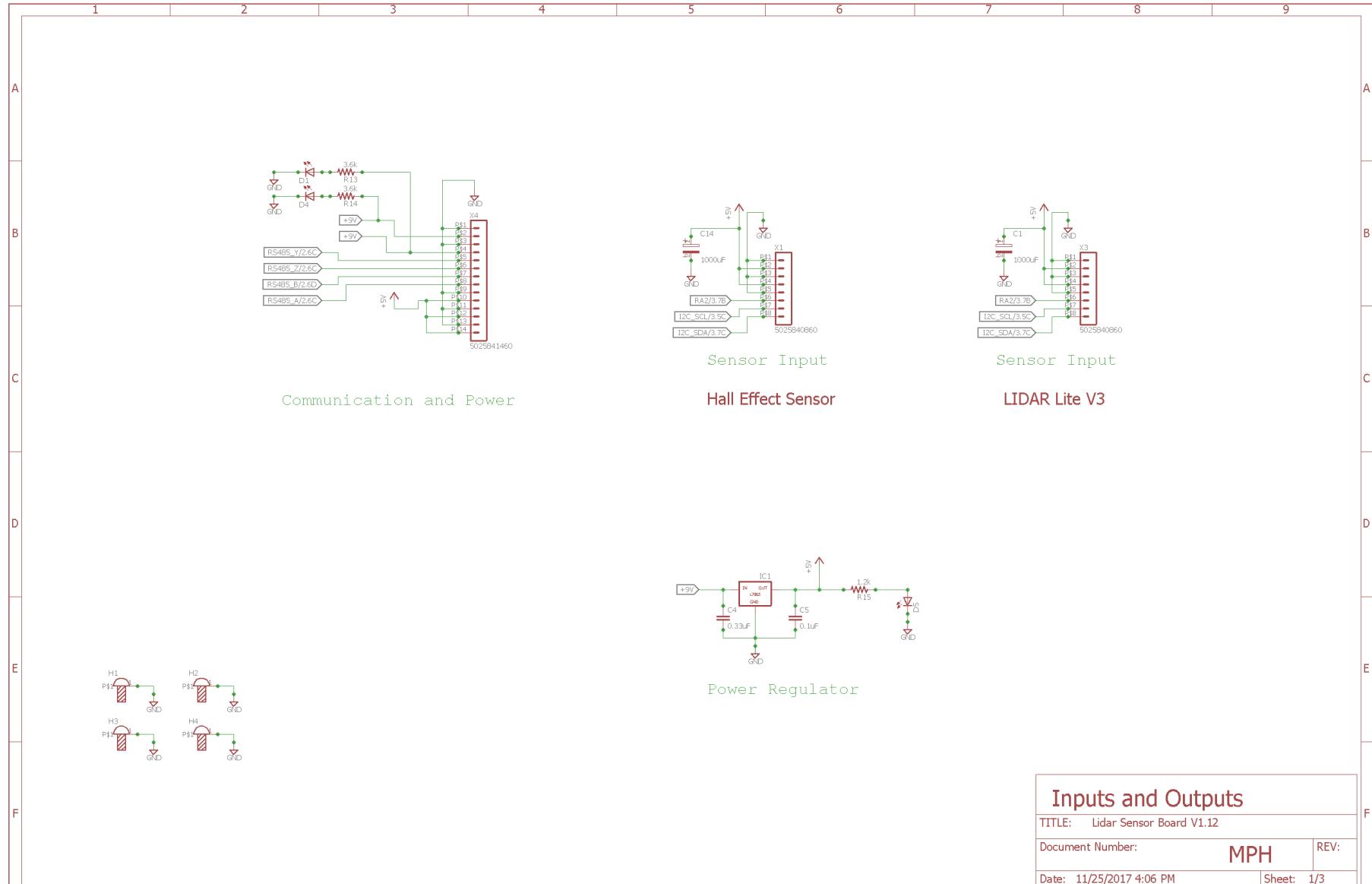


Figure 38: Proximity Sensor Board Inputs and Outputs Schematic

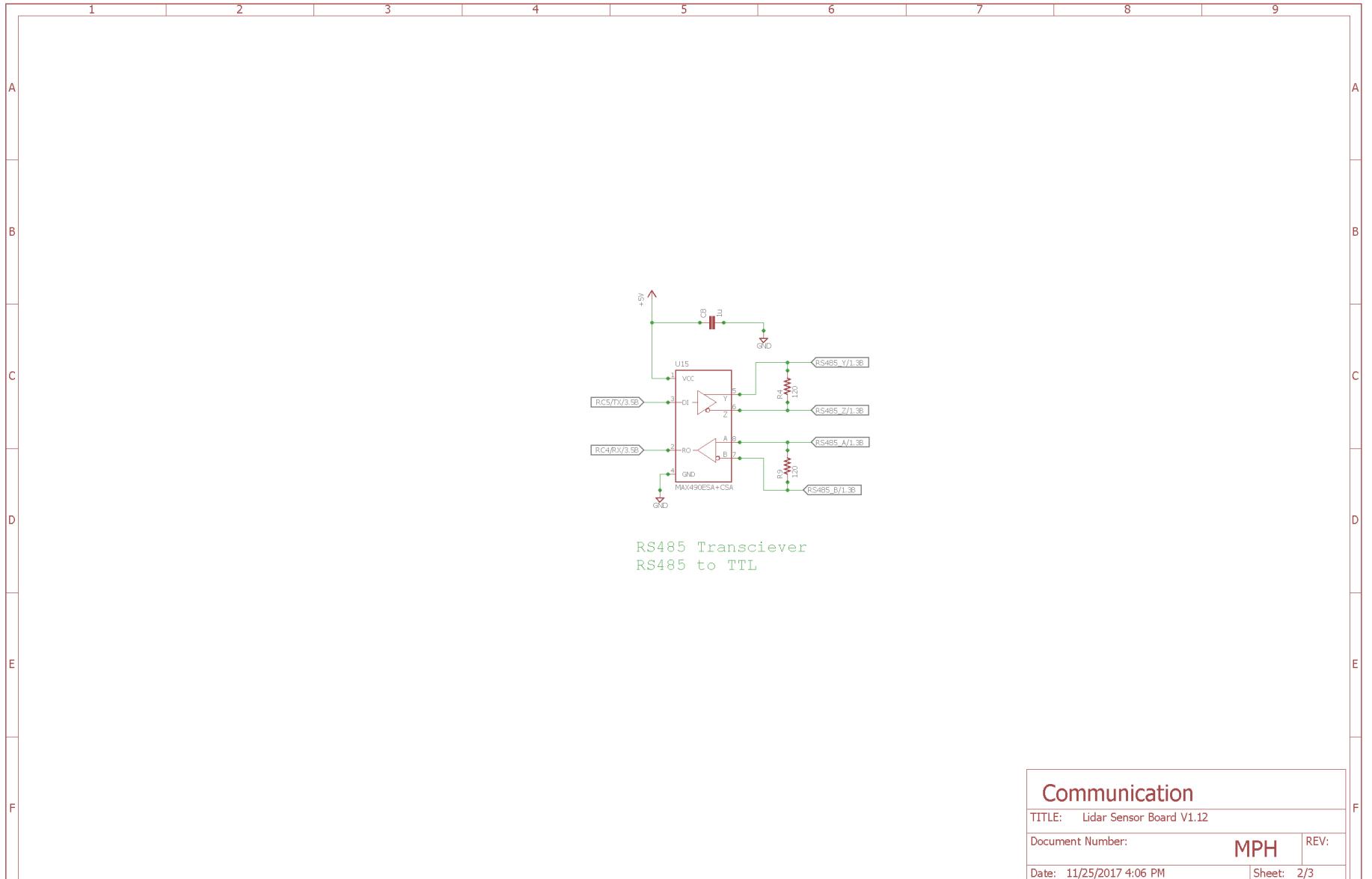


Figure 39: Proximity Sensor Board Communication Schematic

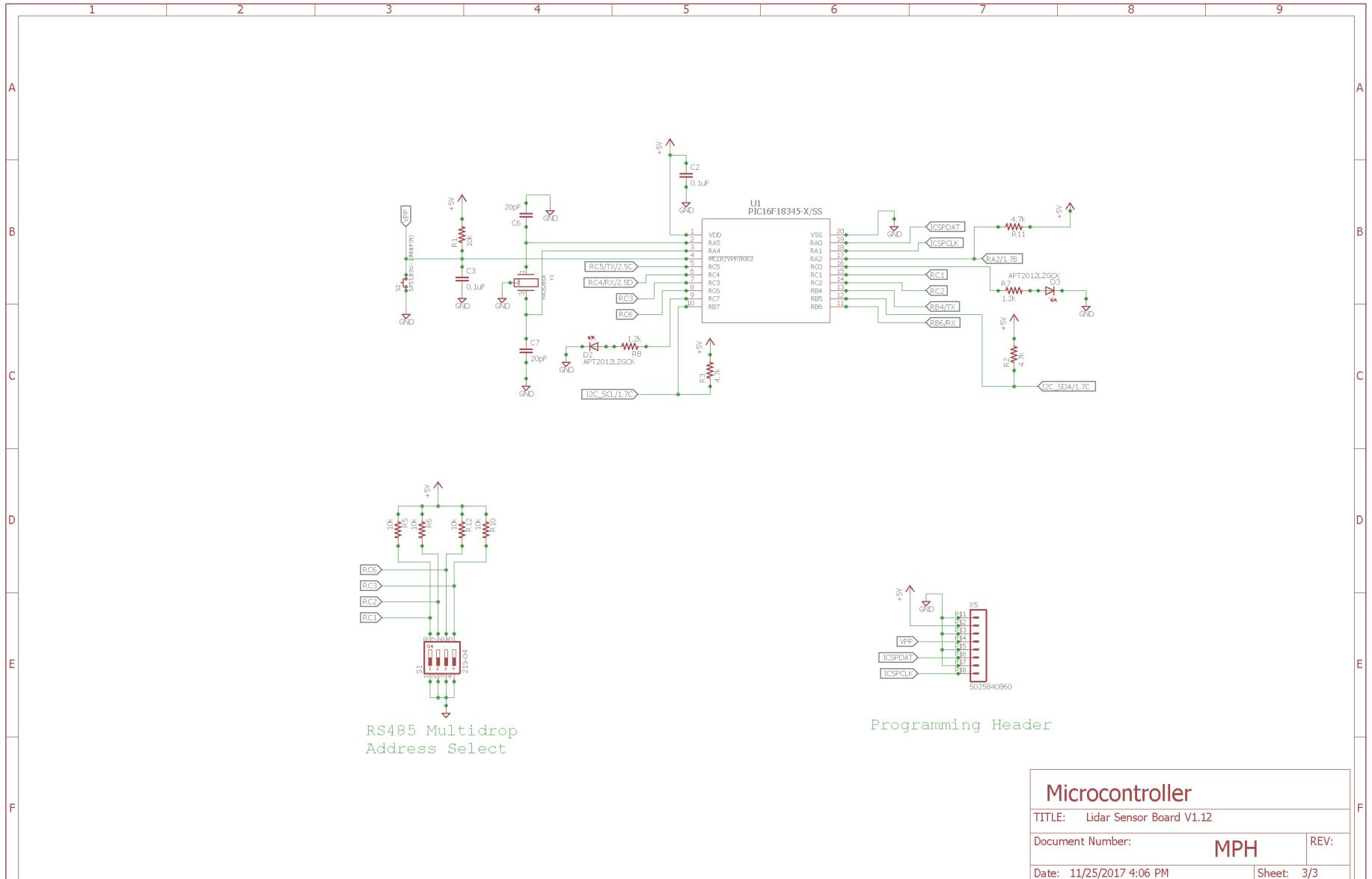


Figure 40: Proximity Sensor Board Microcontroller Schematic

3.4.3 Proximity Sensor Board Software

Figure 41 shows a level 0 software block diagram of how the software for the sensor board will function. The weapon position evaluator module will determine when it is appropriate to scan the environment for enemies according to sensor input data. Once the environment is scanned, the data will be processed and then stored in the microcontroller. The stored environment data will be transmitted to the MCB upon request. (MPH)

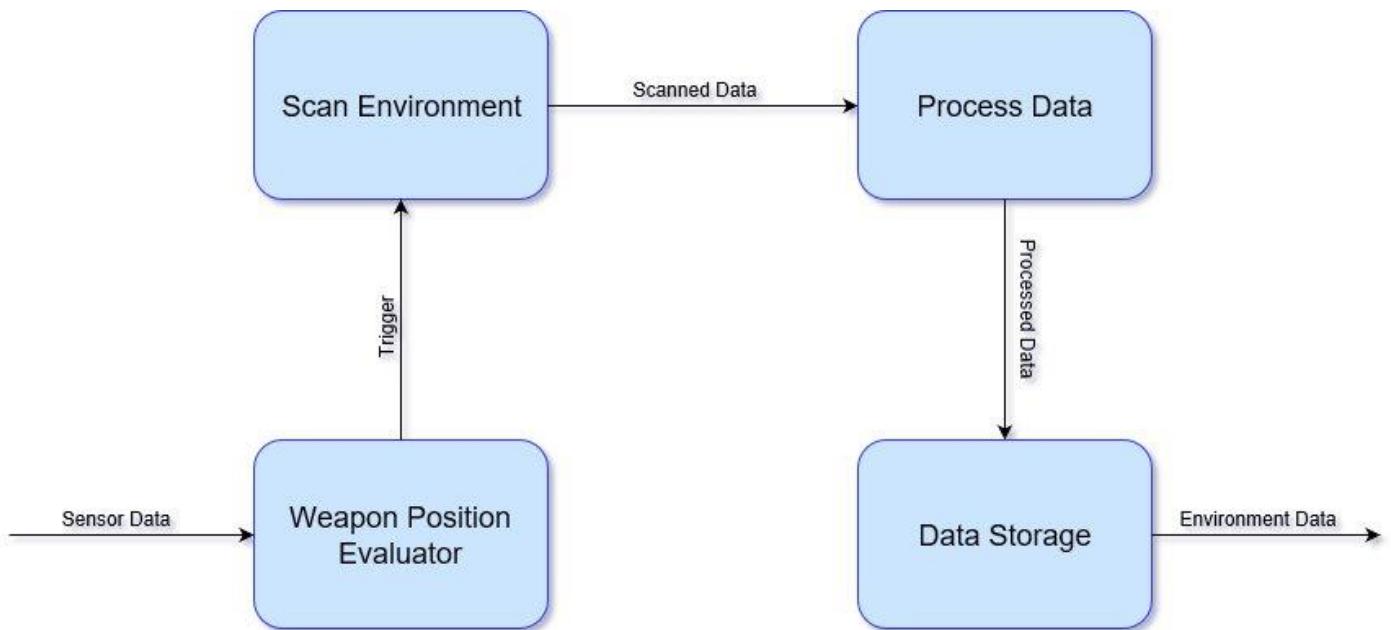


Figure 41: Level 0 Proximity Sensor Board Software Block Diagram

Module	Weapon Position Evaluator
Designer	Michael P. Hritz
Inputs	Signal: Sensor Data
Outputs	Signal: Trigger
Description	The weapon position evaluator will use sensor input data and determine when it's time to send the trigger signal to the vision sensor.

Table 56: Level 0 Proximity Sensor Board Software Weapon Position Evaluator Module

Module	Scan Environment
Designer	Michael P. Hritz
Inputs	Signal: Trigger
Outputs	Signal: Scanned Data
Description	The scan environment module will trigger the vision sensor to scan the environment and look for enemies. The output data will be the scanned data from the vision sensor.

Table 57: Level 0 Proximity Sensor Board Software Scan Environment Module

Module	Process Data
Designer	Michael P. Hritz
Inputs	Signal: Scanned Data
Outputs	Signal: Processed Data
Description	The process data module will take incoming scanned data and process it to determine a distance measurement as and output signal.

Table 58: Level 0 Proximity Sensor Board Software Process Data Module

Module	Data Storage
Designer	Michael P. Hritz
Inputs	Signal: Processed Data
Outputs	Signal: Environment Data
Description	The data storage module will store all processed data and transmit it upon request to the MCB.

Table 59: Level 0 Proximity Sensor Board Software Data Storage Module

Figure 42 shows a level 1 software block diagram of how the software for the sensor board will function. The hall effect sensor will be sampled within the program and the sampled data will be processed and sent to the weapon position evaluator. The weapon position evaluator will then determine if the weapon tool is in position to enable the LIDAR sensor to read. Once the LIDAR sensor reads, the data will be processed and stored. The stored data will then be transmitted to the MCB upon request. (MPH)

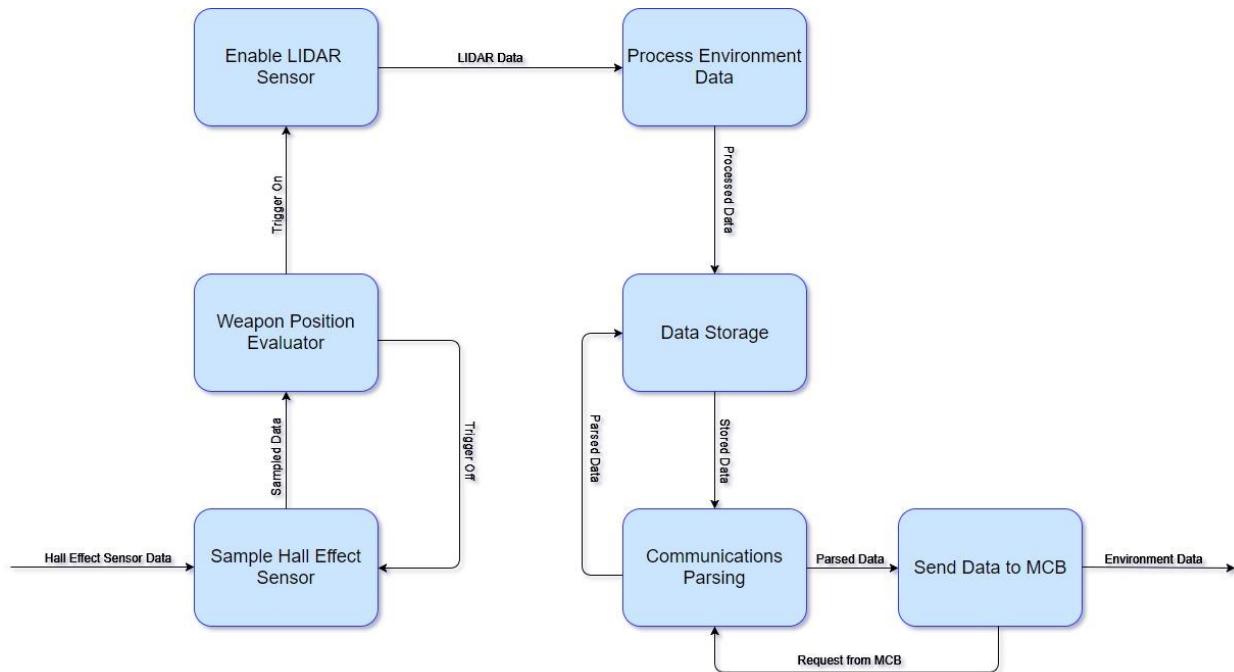


Figure 42: Level 1 Proximity Sensor Board Software Block Diagram

Module	Sample Hall Effect Sensor
Designer	Michael P. Hritz
Inputs	Signal: Hall Effect Sensor Data
Outputs	Signal: Sampled Data
Description	The Sample Hall Effect Sensor module will sample the hall effect sensor and output the data to the Weapon Position Evaluator module.

Table 60: Level 1 Proximity Sensor Board Software Sample Hall Effect Sensor Module

Module	Weapon Position Evaluator
Designer	Michael P. Hritz
Inputs	Signal: Sampled Data
Outputs	Signal: Trigger On, Trigger Off
Description	The weapon position evaluator will use the hall effect sensor input data and determine when it's time to send the trigger signal to enable a read from the LIDAR sensor. The hall effect sensor will keep sampling if the trigger is off.

Table 61: Level 1 Proximity Sensor Board Software Weapon Position Evaluator Module

Module	Enable LIDAR Sensor
Designer	Michael P. Hritz
Inputs	Signal: Trigger On
Outputs	Signal: LIDAR Data
Description	The Enable LIDAR Sensor module will cause the LIDAR sensor to read when the trigger is on. The LIDAR sensor will analyze the environment and output its measurement data.

Table 62: Level 1 Proximity Sensor Board Software Enable LIDAR Sensor Module

Module	Process Environment Data
Designer	Michael P. Hritz
Inputs	Signal: LIDAR Data
Outputs	Signal: Processed Data
Description	The Process Environment Data will process all incoming LIDAR sensor data and convert it to a distance measurement. The output data will be stored.

Table 63: Level 1 Proximity Sensor Board Software Process Environment Data Module

Module	Data Storage
Designer	Michael P. Hritz
Inputs	Signal: Processed Data Signal: Request from MCB
Outputs	Signal: Stored Data
Description	The Data Storage Module will store all incoming data and output the data to the MCB upon request.

Table 64: Level 1 Proximity Sensor Board Software Data Storage Module

Module	Communications Parsing
Designer	Michael P. Hritz
Inputs	Signal: Stored Data Signal: Request from MCB
Outputs	Signal: Parsed Data
Description	Communications parsing will occur within the microcontroller to identify messages on the bus that are designated for the PSB. When a message is identified as addressed for the PSB, the MCB request data will be extracted from the message and given to the data storage module to initiate data transmission to the MCB.

Table 65: Level 1 Proximity Sensor Board Software Communications Parsing Module

Module	Send Data to MCB
Designer	Michael P. Hritz
Inputs	Signal: Parsed Data
Outputs	Signal: Environment Data Signal: Request from MCB
Description	The Send Data to MCB module will take requests from the MCB. When the request is received, all stored data is gathered and sent to the MCB as environment data.

Table 66: Level 1 Proximity Sensor Board Software Send Data to MCB Module

3.4.4 Proximity Sensor Board Pseudo Code

Figure 43, Figure 44, Figure 45, and Figure 46 show the pseudo code for the PSB. User defined functions will be used to handle various tasks of operation. The LIDAR sensor will first be configured by calling its configuration function in the main function. The hall effect sensor will be constantly evaluated for change of state through an external interrupt service routine (window position evaluator) to determine if the window on the weapon attachment is open for the LIDAR sensor to read the environment. Once the LIDAR sensor reads the environment, the data will be processed and converted to centimeters then stored into an integer array. The integer array will then be moved into a storage container array that will be parsed and transmitted to the MCB upon request. (MPH)

```

1  /*  

2   * File: PSB_routine.c  

3   * Author: Michael Hritz  

4   * Date: 11/26/2017  

5   * This program demonstrates the framework  

6   * of the source code for the proximity  

7   * sensor board.  

8   */  

9  

10 #include "config.h"  

11  

12 int current_LIDAR_value //current LIDAR value  

13 int raw_env_data[100]; //raw environment data array  

14 int conversion_factor; //multiply to convert LIDAR data to cm  

15 int processed_data[100]; //store all processed data in this array  

16 int target_to_MCB[100]; //container to transmit target data to MCB  

17 int hallEffectValue; //digital value of hall effect sensor  

18 int trigger; //indicates if LIDAR can be enabled  

19 int MCB_request; //request from MCB  

20 int x_index; //pointer for LIDAR storage array  

21 char LIDARlite_Addr; //slave address of LIDAR  

22  

23 void weapon_pos_evaluate(void){  

24     if(hallEffectValue == 1){  

25         trigger = 1;  

26     }  

27     else trigger = 0;  

28 }  

29  

30 void enableLIDAR(int control_reg, int value, int address){  

31     I2CStart();  

32     us_delay(30);  

33     I2Csendbyte(); //write to LIDAR  

34     us_delay(30);  

35     I2CStop();  

36     us_delay(30);  

37     I2CStart();  

38     us_delay(30);  

39     I2Cgetbyte(); //read LIDAR  

40     us_delay(30);  

41     I2CStop();  

42  

43     //enable LIDAR to read environment  

44 }  

45  

46 void data_storage(void){ //stores LIDAR data in array  

47     if(trigger == 1){  

48         for(x_index = 0; x_index < 100; x_index++){  

49             raw_env_data[x_index] = current_LIDAR_value;

```

Figure 43: PSB Pseudo Code Part 1

```

50 } }
51 }
52 }
53
54 void processEnv_data(void){
55     if(trigger == 0){
56         for(x_index = 0; x_index < 100; x_index++){
57             processed_data[x_index] = raw_env_data[x_index]*conversion_factor;
58         }
59     }
60 }
61
62 void send_to_MCB(void){ //transit LIDAR data to MCB
63     if(MCB_request == 1){
64         for(x_index = 0; x_index < 100; x_index++){
65             target_to_MCB[x_index] = processed_data[x_index];
66         }
67     }
68 }
69
70 void _ISR _SampleHallInterrupt(void){ //interrupt service routine
71     //sample hall effect sensor
72     if(PORTAbits.2 == 1){
73         hallEffectValue = 1;
74     }
75     else
76         hallEffectValue = 0;
77     PIR0bits.0 = 0; //clear flag at end of interrupt
78 }
79
80 void parse_Incoming(void){
81     //parses incoming data
82 }
83
84 void parse_Outgoing(void){
85     //parses data to be transmitted to MCB
86 }
87
88 void us_delay(int n){ //micro second delay function
89     T1CON = 0x8000;
90     TMRL = 0;
91     while(TMRL < 16*n){
92     }
93 }
94
95 void I2Cinit(int BRG) {
96     I2C1BRG = BRG; //set the BRG to 37 for 400kHz fast mode
97     while(I2C1STATbits.P); //bus idle
98     I2C1CONbits.A10M = 0; //7-bit address mode

```

Figure 44: PSB Pseudo Code Part 2

```

99     I2C1CONbits.I2CEN=1; //enable module
100 }
101
102 void I2CStart(void){
103     us_delay(10); //delay
104     I2C1CONbits.SEN = 1; //initiate Start condition
105     while(I2C1CONbits.SEN); //wait for start condition to clear
106     us_delay(10); //delay
107 }
108
109 void I2CStop(void){
110     us_delay(10); //delay
111     I2C1CONbits.PEN = 1; //stop condition enable
112     while(I2C1CONbits.PEN); //wait for stop condition to clear
113     us_delay(10); //delay
114 }
115
116 void I2Csendbyte(char data){
117     while(I2C1STATbits.TBF); //wait if buffer is full
118     I2C1TRN = data; //data to transmission register
119     us_delay(10); //delay 10 microseconds
120 }
121
122 char I2Cgetbyte(void){
123     I2C1CONbits.RCEN = 1;
124     while(!I2C1STATbits.RBF);
125     I2C1CONbits.ACKEN = 1;
126     us_delay(10);
127     return(I2C1RCV);
128 }
129
130 void main(void){ //main function
131     INTPPS = 0x2; //set RA2 as the peripheral interrupt pin
132     INTCON = 0xC1; //setup interrupt control register
133     PIE0 = 0x1; //configure PIE register
134     PIR0bits.0 = 0; //clear the ISR flag
135     TRISA = 0xffff; //set port a as inputs
136     TRISB = 0xff; //set port b as inputs
137     TRISC = 0xdf; //dip switch/RS485/LEDs
138     I2CInit(37); //400KHz fast mode
139     while(1){
140         if(MCB_request != 1){
141             weapon_pos_evaluate();
142             enableLIDAR();
143             data_storage();
144             processEnv_data();
145         }
146         if(MCB_request == 1){
147             parse_Incoming();

```

Figure 45: PSB Pseudo Code Part 3

```

122 122  char I2Cgetbyte(void) {
123 123  |   I2C1CONbits.RCEN = 1;
124 124  |   while(!I2C1STATbits.RBF);
125 125  |   I2C1CONbits.ACKEN = 1;
126 126  |   us_delay(10);
127 127  |   return(I2C1RCV);
128 128  }
129
130 130  void main(void){ //main function
131 131  |   INTPPS = 0x2; //set RA2 as the peripheral interrupt pin
132 132  |   INTCON = 0xC1; //setup interrupt control register
133 133  |   PIE0 = 0x1; //configure PIE register
134 134  |   PIR0bits.0 = 0; //clear the ISR flag
135 135  |   TRISA = 0xffff; //set port a as inputs
136 136  |   TRISB = 0xff; //set port b as inputs
137 137  |   TRISC = 0xdf; //dip switch/RS485/LEDs
138 138  |   I2CInit(37); //400KHz fast mode
139 139  |   while(1){
140 140  |       if(MCB_request != 1){
141 141  |           weapon_pos_evaluate();
142 142  |           enableLIDAR();
143 143  |           data_storage();
144 144  |           processEnv_data();
145 145  |       }
146 146  |       if(MCB_request == 1){
147 147  |           parse_Incoming();
148 148  |           processEnv_data();
149 149  |           parse_Outgoing();
150 150  |           send_to_MCB();
151 151  |       }
152
153 153  |   } //main loop
154 154  }

```

Figure 46: PSB Pseudo Code Part 4

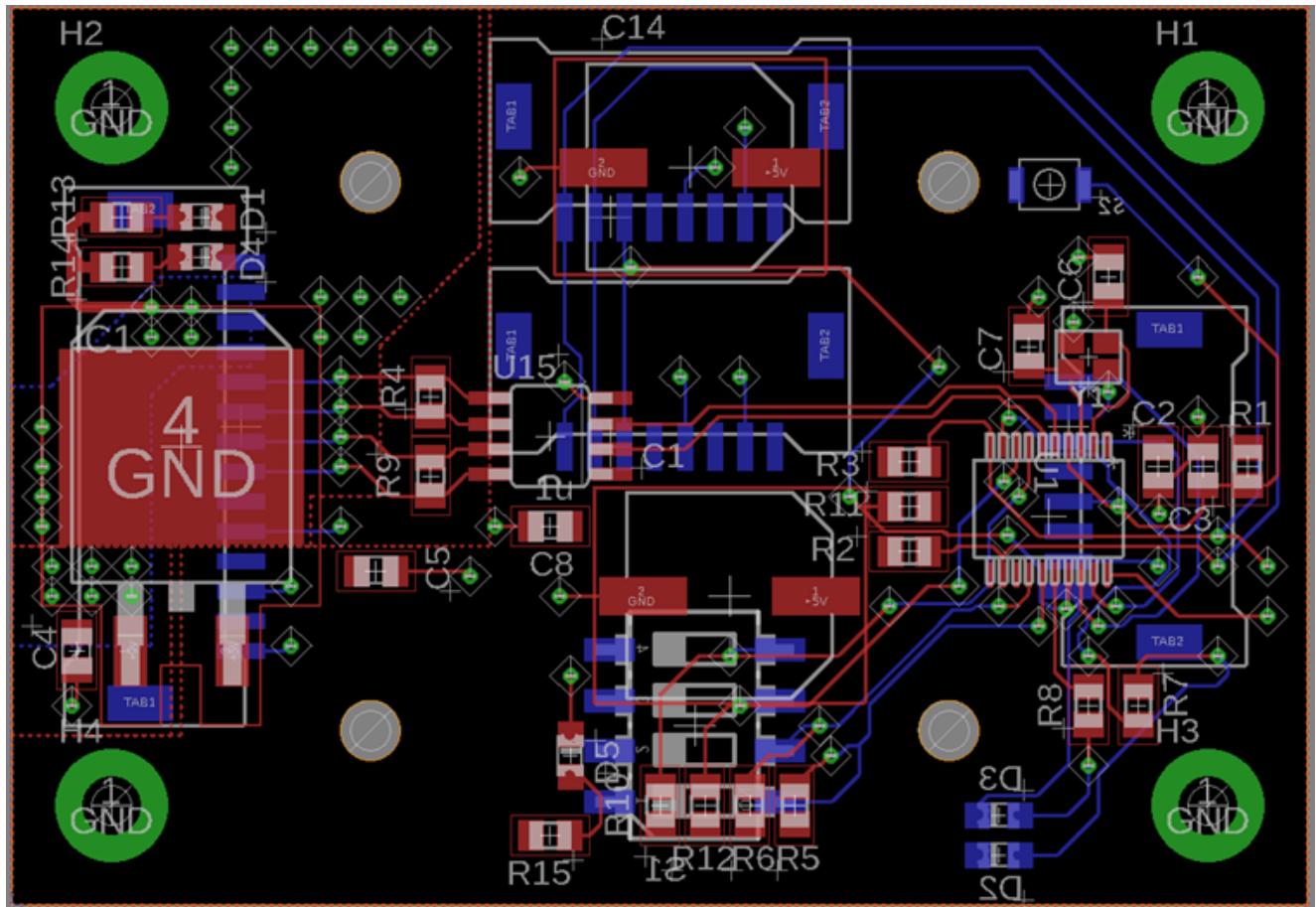


Figure 47: Proximity Sensor Board PCB Layout

3.4.5 Proximity Sensor Board Bill of Material

The bill of material in Table 67 lists the components needed to construct seven of the proximity sensor boards. The reference designators in Table 67 correspond to the reference designators in the schematics of the proximity sensor board. (MPH)

Qty.	Refdes	Part Num.	Description
7	Y1	NX2520SA	CRYSTAL 32.0000MHZ 10PF SMD
7	S2	B3U-1000P(M)	SWITCH TACTILE SPST-NO 0.05A 12V
1	Pickit Programmer	PG164130	PROGRAMMER MCU PICKIT 3
7	IC1	L7805ABD2T-TR	IC REG LINEAR 5V 1.5A D2PAK
7	U15	MAX490ESA+	IC TXRX RS485/RS422 8-SOIC
35	D1, D2, D3, D4, D5	APT2012LZGCK	LED GREEN CLEAR 0805 SMD
7	C1, C14	865080257014	CAP 1000 UF 20% 10 V
7	LIDAR Receptacle	GHR-06V-S	CONN GH HOUSING 6POS 1.25MM
7	S1	219-4MSTR	SWITCH SLIDE DIP SPST 100MA 20V
21	X1, X3, X5	5025780800	CONN PLUG HSG 8CKT BEIGE
21	X1, X3, X5	5025840860	CONN RCPT 8CKT VERT BEIGE
28	Hex Standoff	1894	HEX STANDOFF 4-40 ALUMINUM 5/8"
28	Hex Standoff	2207	HEX STANDOFF 4-40 ALUMINUM 2"
42	C8, C9, C10, C11, C12, C13	C0805C105K3RACTU	Multilayer Ceramic Capacitors MLCC - SMD/SMT 25volts 1uF X7R 10%
14	C6, C7	C0805C200G5GACTU	Multilayer Ceramic Capacitors MLCC - SMD/SMT 50volts 20pf C0G 2%
21	C2, C3, C5	C0805C104J5RACAU TO	Multilayer Ceramic Capacitors MLCC - SMD/SMT 50volts 0.1uF 5% X7R AUTO
7	C4	C0805C334K3RACAU TO	Multilayer Ceramic Capacitors MLCC - SMD/SMT 25volts 0.33uF 10% X7R
231	Clik-Mate Pins	502579-1000	Headers & Wire Housings CLIKMate Plg Term Gld 24-28AWG
1	Pickit Header	22-23-2061	Headers & Wire Housings VERT PCB HDR 6P TIN FRICTION LOCK
112	Nylon Washers	3349	Screws & Fasteners #6 NYLON FLAT WASHER
7	X4	502584-1460	Headers & Wire Housings CLIKMATE PCB RECPT 14P VT TIN PNP TAPE
7	X4	502578-1400	Headers & Wire Housings CLIKMATE PLUG HSG 14P SR BEIGE
7	Hall Effect Sensor	55100-3M-02-A	Industrial Hall Effect / Magnetic Sensors 55100 3M02A HALL SENSOR
35	R1, R5, R6, R10, R12	RC0805FR-0710KL	Thick Film Resistors - SMD 10K OHM 1%
21	R2, R3, R11	RC0805FR-074K7L	Thick Film Resistors - SMD 4.7K OHM 1%
14	R4, R9	RC0805FR-07120RL	Thick Film Resistors - SMD 120 OHM 1%
21	R7, R8, R15	RC0805FR-071K2L	Thick Film Resistors - SMD 1.2K OHM 1%
14	R13, R14	RC0805JR-073K6L	Thick Film Resistors - SMD 3.6K OHM 5%

7	U1	PIC16LF18345-I/SS	Microcontroller for sensor boards
1	Actuator	NB034-0	Actuator for hall effect sensors
3	LIDAR Sensor	RB-Pli-06	LIDAR Lite v3 GARMIN
2	Hex Screws	92949A106	18-8 Stainless Steel Hex Drive Rounded Head Screw (Pack of 100)
1	Hex Screws	92210A105	Passivated 18-8 Stainless Steel Hex Drive Flat Head Screw (Pack of 100)

Table 67: Proximity Sensor Board Bill of Material

3.4.6 Proximity Sensor Board Implementation Revisions

Figure 49, Figure 50, and Figure 51 show the revised PSB schematic. Figure 52 shows the revised PCB layout for the proximity sensor board. The Only change made was the selection of a different RS485 transceiver which required the usage of I/O pins on the microcontroller to toggle the transceiver's driver input and receiver output in high and low impedances modes. This avoided holding the communication line and allowing multiple slaves to communicate to the master. (MPH)

It was decided in the end that the hall effect sensors would not be used. Due to the amount of time it would take to drill holes into the chassis and reassemble the robot, it was decided to just account for a distance threshold instead. To elaborate, distances the indicate the LIDAR sensors are blocked were ignored and all other distances beyond that threshold were acknowledged. (MPH)



Figure 48: PSB Completed Board

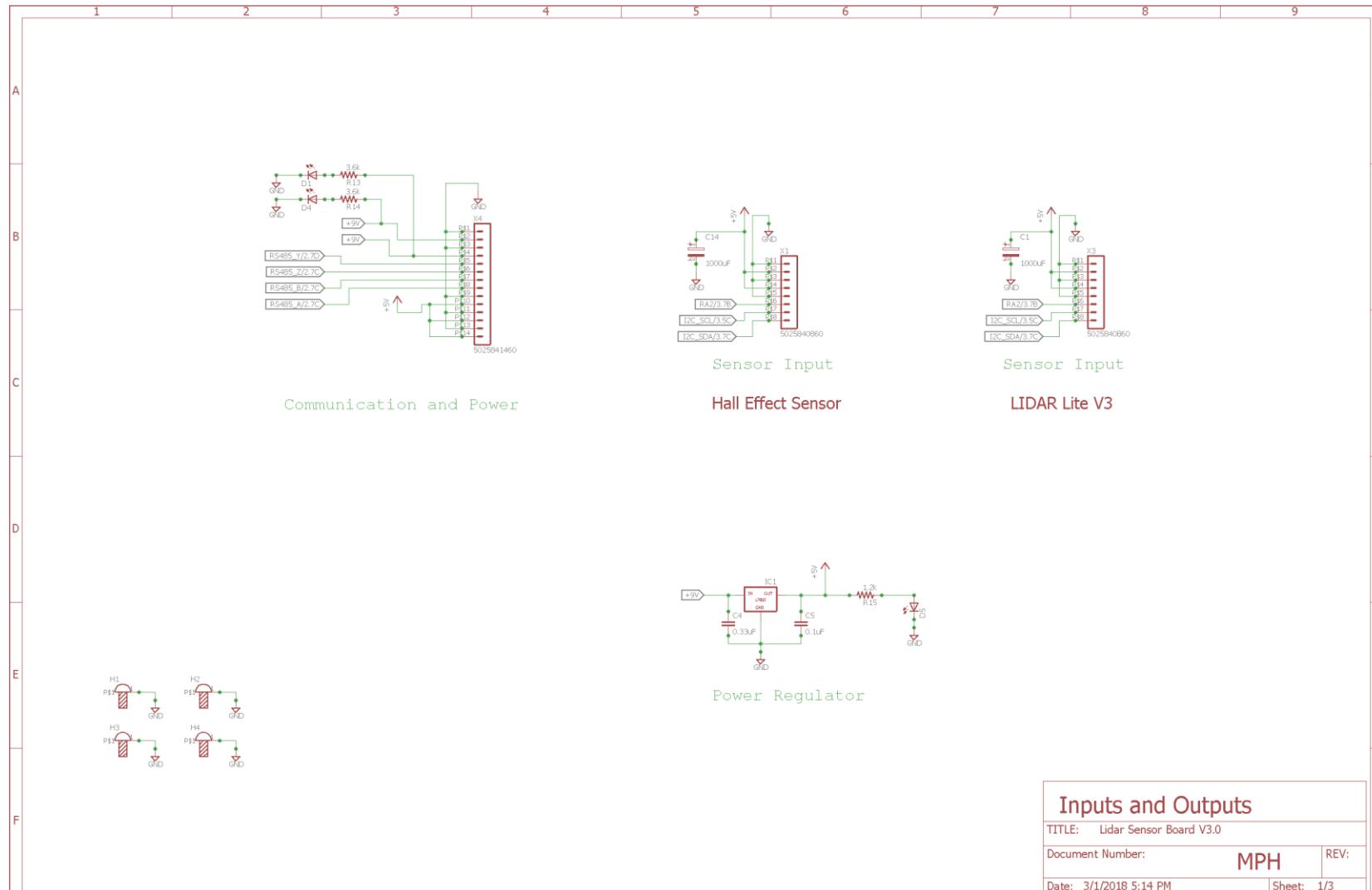


Figure 49: Revised Proximity Sensor Board Inputs and Outputs Schematic

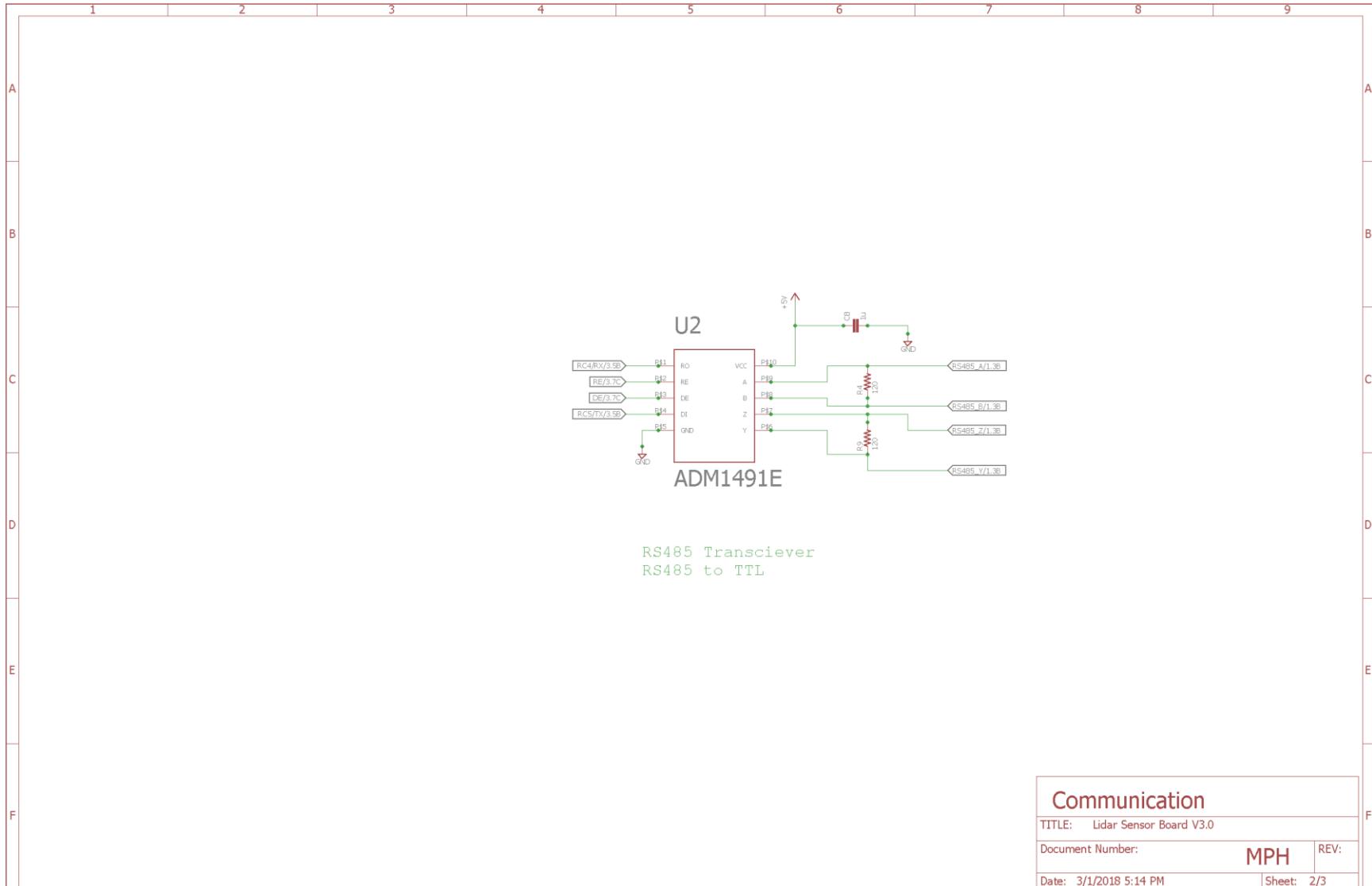


Figure 50: Revised Proximity Sensor Board Communication Schematic

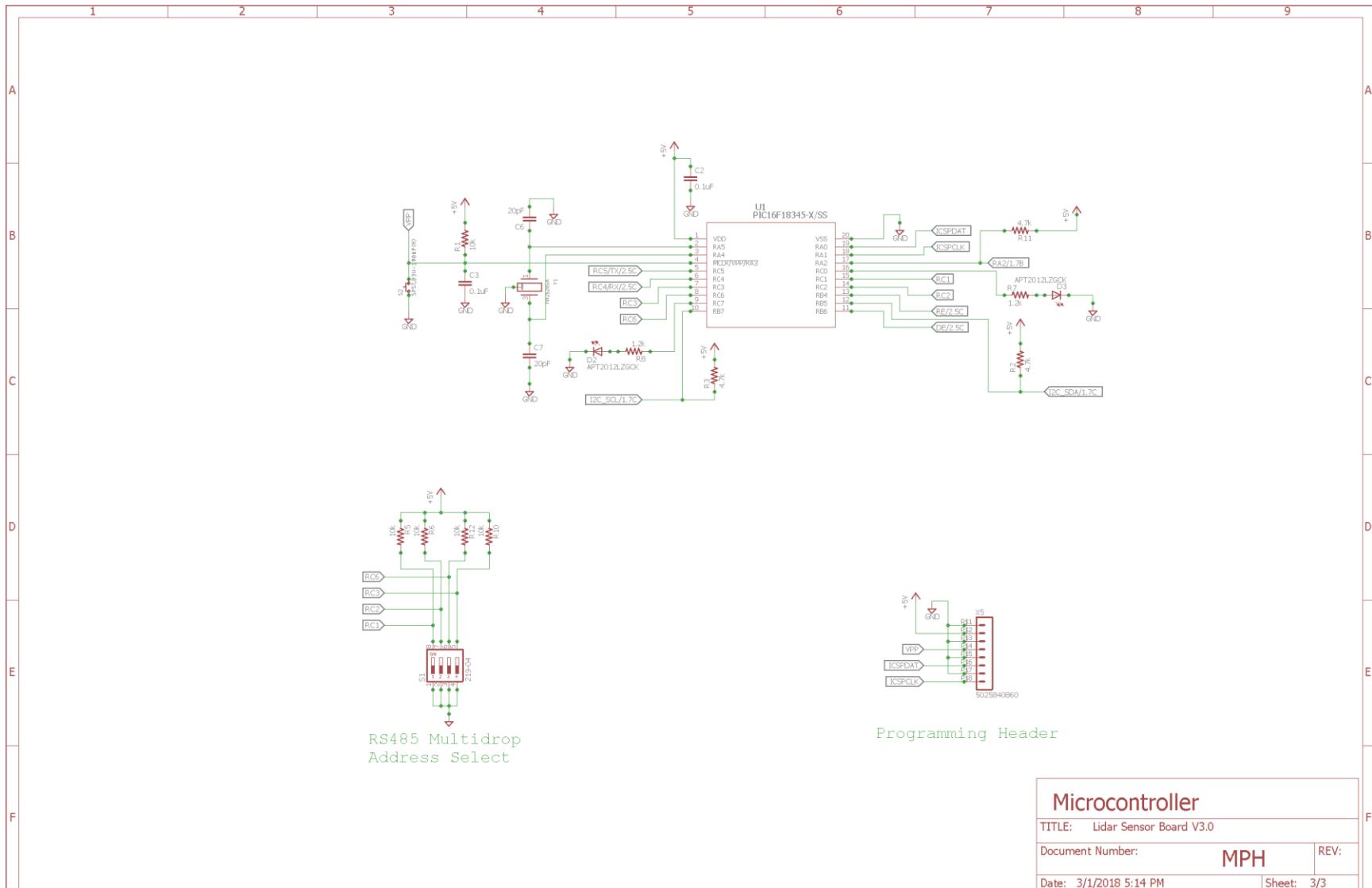


Figure 51: Revised Proximity Sensor Board Microcontroller Schematic

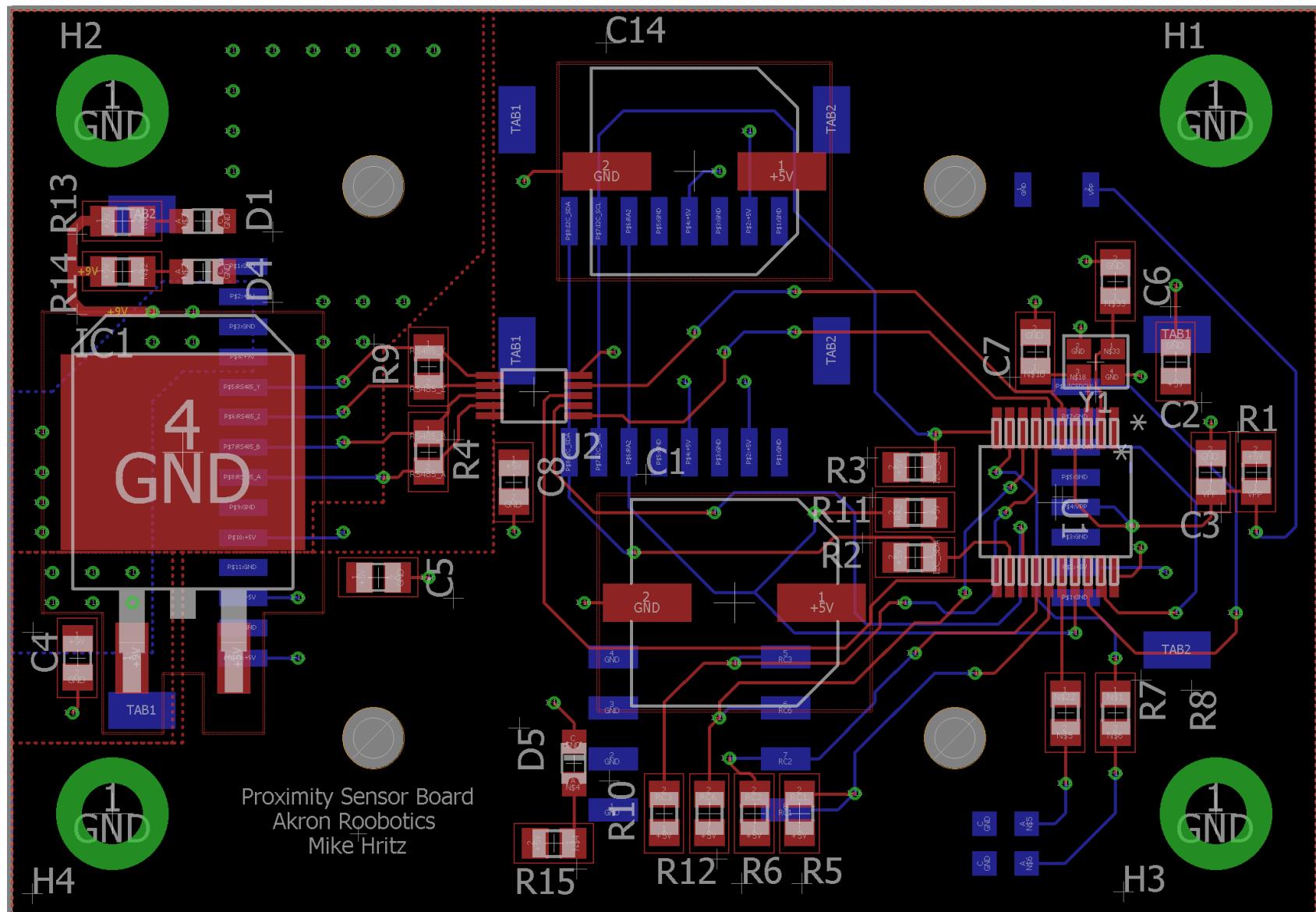


Figure 52: Revised Proximity Sensor Board PCB Layout

3.4.7 Proximity Sensor Board Code

```
1  /**
2   * Generated Main Source File
3
4   * Company:
5   *   Microchip Technology Inc.
6
7   * File Name:
8   *   main.c
9
10  * Summary:
11  *   This is the main file generated using MPLAB(c) Code Configurator
12
13  * Description:
14  *   This header file provides implementations for driver APIs for all modules
15  *   selected in the GUI.
16  *   Generation Information :
17  *     Product Revision : MPLAB(c) Code Configurator - 4.15.1
18  *     Device          : PIC16F18345
19  *     Driver Version  : 2.00
20  *   The generated drivers are tested against the following:
21  *     Compiler       : XC8 1.35
22  *     MPLAB          : MPLAB X 3.40
23
24 */
25
26  /*
27  *   (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
28  *   software and any derivatives exclusively with Microchip products.
29
30  *   THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
31  *   EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
32  *   WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
33  *   PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
34  *   WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
35
36  *   IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
37  *   INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
38  *   WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
39  *   BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
40  *   FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
41  *   ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
42  *   THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
43
44 */
45
46 #include "mcc_generated_files/mcc.h"
47 #include "LIDAR.h"
48 #include "FastTransfer.h"
49 #include <xc.h>
50 #include "mcc_generated_files/tmr0.h"
51 #include "mcc_generated_files/tmr1.h"
52 #include "mcc_generated_files/tmr2.h"
53 #include "RPM_FIFO.h"
54
55 uint16_t ms_30 = 0x3AA; //22 ms delay
56 int LIDAR_Addr = 0;
57
58 /*
59          Main application
60 */
61
62
63
64
```

```

65 void main(void)
66 {
67     // initialize the device
68     SYSTEM_Initialize();
69
70     // When using interrupts, you need to set the Global and Peripheral
71     // Interrupt Enable bits
72     // Use the following macros to:
73
74     // Enable the Global Interrupts
75     INTERRUPT_GlobalInterruptEnable();
76
77     // Enable the Peripheral Interrupts
78     INTERRUPT_PeripheralInterruptEnable();
79
80     // Disable the Global Interrupts
81     //INTERRUPT_GlobalInterruptDisable();
82
83     // Disable the Peripheral Interrupts
84     //INTERRUPT_PeripheralInterruptDisable();
85     LATBbits.LATB4 = 0x00; //low level enables the receiver output (low
86     // impedance)
87     LIDAR_Addr =(PORTCbits.RC1) + ((PORTCbits.RC2 << 1)) + ((PORTCbits.RC6) <<
88     2) + ((PORTCbits.RC3) << 3); //inverted dip switches, on is off and off is on
89     TMR0_Wipe();
90     ms_delay30(ms_30); //LIDAR power up initialize delay 30 ms
91     TMR0_StopTimer();
92     Fast_Xfer_Config(LIDAR_Addr);
93     LIDAR_Config();
94     while (1)
95     {
96         execute_Read();
97     }
98 /**
99  * End of File
99 */

```

```

1  #include <xc.h>
2  #include <stdbool.h>
3  #include <stdlib.h>
4  #include "FastTransfer.h"
5  #include "mcc_generated_files/eusart.h"
6
7  void wipeBuf(unsigned char * buf, uint16_t size);
8
9  circBuff_t ringBuff;
10
11 //Captures address of receive array, the max data address, the address of the
12 //module, true/false if AKNAKs are wanted and the Serial address
13 void begin(int * ptr, unsigned char maxSize, unsigned char givenAddress, bool
14 error, void (*stufftosend)(unsigned char), unsigned char
15 (*stufftoreceive)(void), int (*stuffavailable)(void), unsigned char
16 (*stuffpeek)(void))
17 {
18     receiveArrayAddress = ptr;
19     moduleAddress = givenAddress;
20     serial_write = stufftosend;
21     serial_available = stuffavailable;
22     serial_peek = stuffpeek;
23     serial_read = stufftoreceive;
24     maxDataAddress = maxSize / 2;
25     AKNAKsend = error;
26     alignErrorCounter = 0;
27     ringBuff.head=0;
28     ringBuff.tail=0;
29     ringBuff.count=0;
30 }
31
32 //CRC Calculator
33 unsigned char CRC8(const unsigned char * data, unsigned char len)
34 {
35     unsigned char crc = 0x00;
36     while (len--)
37     {
38         unsigned char extract = *data++;
39         unsigned char tempI;
40         for (tempI = 8; tempI; tempI--)
41         {
42             unsigned char sum = (crc ^ extract) & 0x01;
43             crc >>= 1;
44             if (sum)
45             {
46                 crc ^= polynomial;
47             }
48             extract >>= 1;
49         }
50     }
51     return crc;
52 }
53
54 void testSendByteInternal(void)
55 {
56     serial_write(0x06); //start address
57 }
58
59 //Sends out send buffer with a 2 start bytes, where the packet is going, where
60 //it came from, the size of the data packet, the data and the crc.
61 bool sendData(unsigned char whereToSend)
62 {
63
64     //calculate the crc
65     unsigned char CS = CRC8(ringBuff.buf, ringBuff.count);

```

```

61     serial_write(0x06); //start address
62     serial_write(0x85); //start address
63     serial_write(whereToSend);
64     serial_write(moduleAddress);
65     serial_write(ringBuff.count); //length of packet not including the crc
66
67
68     //send the rest of the packet
69     int i;
70     for (i = 0; i < ringBuff.count; i++)
71     {
72         serial_write(*(ringBuff.buf + i));
73     }
74
75     //send the crc
76     serial_write(CS);
77
78     //record the sent message data for acknak check later
79     crcBufS_put(&crc_buffer, whereToSend, CS, 0);
80
81     // clears the buffer after a sending
82     FastTransfer_buffer_flush(&ringBuff, 1);
83     return true;
84 }
85
86
87 bool receiveData()
88 {
89     //start off by looking for the header bytes. If they were already found in a
90     //previous call, skip it.
91     if (rx_len == 0)
92     {
93         //this size check may be redundant due to the size check below, but for
94         //now I'll leave it the way it is.
95         if (serial_available() > 4)
96         {
97             //LATCbits.LATC0 = 0x01; //LED D3
98             //this will block until a 0x06 is found or buffer size becomes less
99             //than 3.
100            while (serial_read() != 0x06)
101            {
102                //This will trash any preamble junk in the serial buffer
103                //but we need to make sure there is enough in the buffer to
104                //process while we trash the rest
105                //if the buffer becomes too empty, we will escape and try again
106                //on the next call
107                alignErrorCounter++; //increments the counter whenever a byte is
108                //trashed
109                if (serial_available() < 5)
110                    return false;
111            }
112            if (serial_read() == 0x85)
113            {
114                LATCbits.LATC0 = 0x01; //LED D3
115                rx_address = serial_read(); // pulls the address
116                returnAddress = serial_read(); // pulls where the message came
117                from
118                rx_len = serial_read(); // pulls the length
119
120                //make sure the address received is a match for this module if
121                //not throw the packet away
122
123                if (rx_address != moduleAddress)
124                {
125                    addressErrorCounter++; // increments a counter whenever the

```

```

118     wrong address is received
119     //if the address does not match the buffer is flushed for
120     //the size of
121     //    int u;
122     //    for (u = 0; u <= (rx_len + 1); u++)
123     //    {
124     //        serial_read();
125     //    }
126     rx_len = 0; // reset length
127     return false;
128 }
129 }
130 }
131
132 //we get here if we already found the header bytes, the address matched what
133 //we know, and now we are byte aligned.
134 if (rx_len != 0)
135 {
136     //this check is preformed to see if the first data address is a 255, if
137     //it is then this packet is an AKNACK
138     //    if (rx_array_inx == 0)
139     //    {
140     //        while (!serial_available() >= 1));
141     //        if (255 == serial_peek())
142     //        {
143     //            CRCcheck();
144     //            rx_len = 0;
145     //            rx_array_inx = 0;
146     //            wipeBuf(rx_buffer, RX_BUFFER_LENGTH);
147     //            return false;
148     //        }
149
150     while ((serial_available() > 0 && ( rx_array_inx <=
151     rx_len))//(serial_available() > 0) && ( rx_array_inx <= rx_len))
152     {
153         rx_buffer[rx_array_inx++] = serial_read();
154
155         if (rx_len == (rx_array_inx - 1))
156         {
157             //seem to have got whole message
158             //last uint8_t is CS
159             calc_CS = CRC8(rx_buffer, rx_len);
160
161             if (calc_CS == rx_buffer[rx_array_inx - 1])
162             { //CS good
163
164                 //reassembles the data and places it into the receive array
165                 //according to data address.
166                 int r;
167                 for (r = 0; r < rx_len; r = r + 3)
168                 {
169                     if (rx_buffer[r] < maxDataAddress)
170                     {
171                         group.parts[0] = rx_buffer[r + 1];
172                         group.parts[1] = rx_buffer[r + 2];
173
174                         receiveArrayAddress[(rx_buffer[r])] =
175                         group.integer;
176                     } else
177                     {

```

```

176                     dataAddressErrorCounter++;
177                 }
178             }
179         LATBbits.LATB6 = 0x01; //high level enables driver output (low impedance)
180         setTransmitStall(false);
181
182         if (AKNAKsend)
183         { // if enabled sends an AK
184             unsigned char holder[3];
185             holder[0] = 255;
186             holder[1] = 1;
187             holder[2] = rx_buffer[rx_array_inx - 1];
188             unsigned char crcHolder = CRC8(holder, 3);
189             serial_write(0x06);
190             serial_write(0x85);
191             serial_write(returnAddress);
192             serial_write(moduleAddress);
193             serial_write(3);
194             serial_write(255);
195             serial_write(1);
196             serial_write(rx_buffer[rx_array_inx - 1]);
197             serial_write(crcHolder);
198         }
199
200         rx_len = 0;
201         rx_array_inx = 0;
202
203         wipeBuf(rx_buffer,RX_BUFFER_LENGTH);
204
205         return true;
206     }
207     else
208     {
209         crcErrorCounter++; //increments the counter every time a crc fails
210
211         if (AKNAKsend)
212         { // if enabled sends NAK
213             unsigned char holder[3];
214             holder[0] = 255;
215             holder[1] = 2;
216             holder[2] = rx_buffer[rx_array_inx - 1];
217             unsigned char crcHolder = CRC8(holder, 3);
218             serial_write(0x06);
219             serial_write(0x85);
220             serial_write(returnAddress);
221             serial_write(moduleAddress);
222             serial_write(3);
223             serial_write(255);
224             serial_write(2);
225             serial_write(rx_buffer[rx_array_inx - 1]);
226             serial_write(crcHolder);
227         }
228
229         //failed checksum, need to clear this out
230         rx_len = 0;
231         rx_array_inx = 0;
232
233         wipeBuf(rx_buffer,RX_BUFFER_LENGTH);
234         return false;
235     }
236 }
237 }
238
239 return false;
240 }
```

```

241
242
243 // populates to what data address and what info needs to be sent
244 void ToSend(unsigned char where, unsigned int what)
245 {
246     FastTransfer_buffer_put(&ringBuff, where, what);
247 }
248
249
250 // disassembles the data and places it in a buffer to be sent
251 void FastTransfer_buffer_put(circBuff_t *_this, unsigned char towhere, unsigned
252 int towhat)
253 {
254     group.integer = towhat;
255
256     if (_this->count < (BUFFER_SIZE - 3))
257     {
258         _this->buf[_this->head] = towhere;
259         _this->head = FastTransfer_buffer_modulo_inc(_this->head, BUFFER_SIZE);
260         ++_this->count;
261         _this->buf[_this->head] = group.parts[0];
262         _this->head = FastTransfer_buffer_modulo_inc(_this->head, BUFFER_SIZE);
263         ++_this->count;
264         _this->buf[_this->head] = group.parts[1];
265         _this->head = FastTransfer_buffer_modulo_inc(_this->head, BUFFER_SIZE);
266         ++_this->count;
267     }
268
269
270 //pulls info out of the send buffer in a first in first out fashion
271 unsigned char FastTransfer_buffer_get(circBuff_t* _this)
272 {
273     unsigned char c;
274     if (_this->count > 0)
275     {
276         c = _this->buf[_this->tail];
277         _this->tail = FastTransfer_buffer_modulo_inc(_this->tail, BUFFER_SIZE);
278         --_this->count;
279     }
280     else
281     {
282         c = 0;
283     }
284     return (c);
285 }
286
287 //flushes the send buffer to get it ready for new data
288 void FastTransfer_buffer_flush(circBuff_t* _this, const int clearBuffer)
289 {
290     _this->count = 0;
291     _this->head = 0;
292     _this->tail = 0;
293     if (clearBuffer)
294     {
295         wipeBuf(_this->buf, sizeof(_this->buf));
296     }
297 }
298
299 // increments counters for the buffer functions
300 unsigned int FastTransfer_buffer_modulo_inc(const unsigned int value, const
301 unsigned int modulus)
302 {
303     unsigned int my_value = value + 1;
304     if (my_value >= modulus)

```

```

304     {
305         my_value = 0;
306     }
307     return (my_value);
308 }
309
310 //searches the buffer for the status of a message that was sent
311 unsigned char AKNAK(unsigned char module)
312 {
313     int r;
314     for (r = 0; r < CRC_COUNT; r++)
315     {
316         if (module == crcBufS_get(&crc_buffer, r, 0))
317         {
318             return crcBufS_get(&crc_buffer, r, 2);
319         }
320     }
321     return 4;
322 }
323
324
325 //returns align error
326 unsigned int alignError(void)
327 {
328     return alignErrorCounter;
329 }
330
331 //returns CRC error
332 unsigned int CRCError(void)
333 {
334     return crcErrorCounter;
335 }
336
337 //returns address error
338 unsigned int addressError(void)
339 {
340     return addressErrorCounter;
341 }
342
343 unsigned int dataAddressError(void)
344 {
345     return dataAddressErrorCounter;
346 }
347
348 // after a packet is sent records the info of that packet
349 void crcBufS_put(struct crcBufS* _this, unsigned char address, unsigned char
oldCRC, unsigned char status)
350 {
351     _this->buf[_this->head] = address;
352     _this->head++;
353     _this->buf[_this->head] = oldCRC;
354     _this->head++;
355     _this->buf[_this->head] = status;
356     _this->head++;
357     if (_this->head >= CRC_BUFFER_SIZE)
358     {
359         _this->head = 0;
360     }
361 }
362
363 // after a Ak or NAK is received that status is stored
364 void crcBufS_status_put(struct crcBufS* _this, unsigned char time, unsigned char
status)
365 {
366     if (time >= CRC_COUNT)

```

```

367     {
368         time = CRC_COUNT - 1;
369     }
370     time = time + 1;
371     int wantedTime = time * 3;
372     if (wantedTime > _this->head)
373     {
374         wantedTime = (CRC_BUFFER_SIZE) - (wantedTime - _this->head);
375         _this->buf[(_wantedTime + 2)] = status;
376     }
377     else
378     {
379         _this->buf[(_this->head - wantedTime) + 2] = status;
380     }
381 }
382
383 // pulls data from the AKNAK buffer
384 unsigned char crcBufS_get(struct crcBufS* _this, unsigned char time, unsigned
385 char space)
386 {
387     if (time >= CRC_COUNT)
388     {
389         time = CRC_COUNT - 1;
390     }
391     if (space >= CRC_DEPTH)
392     {
393         space = CRC_DEPTH - 1;
394     }
395     time = time + 1;
396     int wantedTime = time * 3;
397     if (wantedTime > _this->head)
398     {
399         wantedTime = (CRC_BUFFER_SIZE) - (wantedTime - _this->head);
400         return (_this->buf[(_wantedTime + space)]);
401     }
402     else
403     {
404         return (_this->buf[(_this->head - wantedTime) + space]);
405     }
406 }
407
408 //when an AK or NAK is received this compares it to the buffer and records the
409 //status
410 void CRCcheck(void)
411 {
412     while (!(serial_available() > 3)); // trap makes sure that there are enough
413     // bytes in the buffer for the AKNAK check
414     unsigned char arrayHolder[3];
415     arrayHolder[0] = serial_read();
416     arrayHolder[1] = serial_read();
417     arrayHolder[2] = serial_read();
418     unsigned char SentCRC = serial_read();
419     unsigned char calculatedCRC = CRC8(arrayHolder, 3);
420
421     if (SentCRC == calculatedCRC)
422     {
423         int rt;
424         for (rt = 0; rt < CRC_COUNT; rt++)
425         {
426             if (returnAddress == crcBufS_get(&crc_buffer, rt, 0))
427             {
428                 if (arrayHolder[2] == crcBufS_get(&crc_buffer, rt, 1))

```

```

429     {
430         if (arrayHolder[1] == 1)
431         {
432             crcBufS_status_put(&crc_buffer, rt, 1);
433             break;
434         }
435         else if (arrayHolder[1] == 2)
436         {
437             crcBufS_status_put(&crc_buffer, rt, 2);
438             break;
439         }
440     }
441 }
442 }
443 else
444 {
445     crcErrorCounter++;
446 } //increments the counter every time a crc fails
447 }
448 void wipeBuf(unsigned char * buf, uint16_t size)
449 {
450     int i=0;
451     for(i=0;i<size;i++)
452     {
453         buf[i]=0;
454     }
455 }
```

```

1  /*
2   * File:    FastTransfer.h
3   * Author:  Igor
4   *
5   * Created on March 23, 2015, 1:21 PM
6   */
7
8 #ifndef FASTTRANSFER_H
9 #define FASTTRANSFER_H
10
11 #include <stdint.h>
12
13 #define RX_BUFFER_LENGTH 30
14 //the capital D is so there is no interference with the lower case d of
15 //EasyTransfer
16 #define Details(name) (int*)&name,sizeof(name)
17
18 void (*serial_write)(unsigned char);
19 unsigned char (*serial_read)(void);
20 int (*serial_available)(void);
21 unsigned char (*serial_peek)(void);
22 unsigned char rx_buffer[RX_BUFFER_LENGTH]; //address for temporary storage and
23 //parsing buffer
24 unsigned char rx_array_inx; //index for RX parsing buffer
25 unsigned char rx_len; //RX packet length according to the packet
26 unsigned char calc_CS; //calculated Checksum
27 unsigned char moduleAddress; // the address of this module
28 unsigned char returnAddress; //the address to send the crc back to
29 unsigned char maxDataAddress; //max address allowable
30 int * receiveArrayAddress; // this is where the data will go when it is received
31 unsigned char * sendStructAddress; // this is where the data will be sent from
32 bool AKNAKsend; // turns the acknowledged or not acknowledged on/off
33 unsigned int alignErrorCounter; //counts the align errors
34 unsigned int crcErrorCounter; // counts any failed crcs
35 unsigned int addressErrorCounter; // counts every time a wrong address is received
36 unsigned int dataAddressErrorCounter; // counts if the received data fall outside
37 //of the receive array
38 unsigned char rx_address; //RX address received
39 #define polynomial 0x8C //polynomial used to calculate crc
40 #define BUFFER_SIZE 50 //ring buffer size
41 #define CRC_COUNT 5 // how many AKNAKs are stored
42 #define CRC_DEPTH 3 // how many pieces of data are stored with each CRC send
43 //event
44 #define CRC_BUFFER_SIZE (CRC_COUNT * CRC_DEPTH) //crc buffer size 5 deep and 3
45 //bytes an entry
46
47 typedef struct
48 {
49     unsigned char buf[BUFFER_SIZE];
50     int head;
51     int tail;
52     int count;
53 } circBuff_t;
54
55 union stuff
56 { // this union is used to join and disassemble integers
57     unsigned char parts[2];
58     unsigned int integer;
59 };
56
57 struct crcBufS
58 { // this is where the address where sent to, the sent crc, the status of the
59   AKNAK
60     unsigned char buf[CRC_BUFFER_SIZE];

```

```

60     int head;
61 };
62 struct crcBufS crc_buffer;
63
64 unsigned char CRC8(const unsigned char * data, unsigned char len);
65 void FastTransfer_buffer_put(circBuff_t *_this, const unsigned char towhere,
66 const unsigned int towhat);
67 unsigned char FastTransfer_buffer_get(circBuff_t* _this);
68 void FastTransfer_buffer_flush(circBuff_t* _this, const int clearBuffer);
69 unsigned int FastTransfer_buffer_modulo_inc(const unsigned int value, const
70 unsigned int modulus);
71 void crcBufS_put(struct crcBufS* _this, unsigned char address, unsigned char
72 oldCRC, unsigned char status);
73 void crcBufS_status_put(struct crcBufS* _this, unsigned char time, unsigned char
74 status);
75 unsigned char crcBufS_get(struct crcBufS* _this, unsigned char time, unsigned
76 char space);
77 void CRCcheck(void);
78
79 void begin(int * ptr, unsigned char maxSize, unsigned char givenAddress, bool
80 error, void (*stufftosend)(unsigned char), unsigned char
81 (*stufftoreceive)(), int (*stuffavailable)(), unsigned char
82 (*stuffpeek)());
83 bool sendData(unsigned char whereToSend);
84 bool receiveData();
85 void ToSend(unsigned char where, unsigned int what);
86 unsigned char AKNACK(unsigned char module);
87 unsigned int alignError(void);
88 unsigned int CRCError(void);
89 unsigned int addressError(void);
90 unsigned int dataAddressError(void);
91
92 void testSendByteInternal(void);
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589 #endif /* FASTTRANSFER_H */
```

```

1  /*
2   * File:    LIDAR.c
3   * Author:  Mike Hritz
4   *
5   * Created on March 13, 2018, 12:00 PM
6   */
7
8
9 #include "mcc_generated_files/mcc.h"
10 #include "mcc_generated_files/i2c2.h"
11 #include "mcc_generated_files/ext_int.h"
12 #include "mcc_generated_files/interrupt_manager.h"
13 #include "mcc_generated_files/tmr0.h"
14 #include "mcc_generated_files/tmr1.h"
15 #include "mcc_generated_files/tmr2.h"
16 #include "FastTransfer.h"
17 #include "RPM_FIFO.h"
18 #include "WBOXC_Filter.h"
19 #include <xc.h>
20
21
22 #define balanced_mode
23 //##define short_range_high_speed
24 //##define higher_speed_short_range
25 //##define max_range
26 //##define high_sense_high_error
27 //##define low_sense_low_error
28 //##define custom_model
29 //##define custom_mode2
30
31 #define continuous_trigger
32 #define no_array
33
34
35 static unsigned char writeBuffer[5];
36 static unsigned char readBuffer[5];
37 static I2C2_MESSAGE_STATUS status;
38 int receiver_bias_count = 0;
39 int x = 0;
40 int x_snapshot = 0;
41 int y = 0;
42 int zero_inc = 0;
43 int trigger = 0;
44 int rev_count = 0;
45 int cycle_count = 0;
46 float weapon_rpm = 0;
47 int filtered_distance = 0;
48 unsigned int LIDAR_Read_Val[50];
49 unsigned int LIDAR_Value = 0;
50 int weapon_inactivity = 0;
51 int LIDAR_RX_Buffer[3];
52 int rpm_print_delay = 0;
53 uint16_t ms_1 = 0x82DC;
54 uint16_t us_250 = 0xE1;
55 uint16_t ms_30 = 0x3AA; //30 ms delay
56 int PSB_Addr = 0;
57 bool TMR0_overflow = 0;
58 bool index_compare = 0;
59
60
61 void Fast_Xfer_Config(int addr){
62     PSB_Addr = addr;
63     begin(LIDAR_RX_Buffer, sizeof(LIDAR_RX_Buffer), PSB_Addr, false,
64           EUSART_Write, EUSART_Read, EUSART_Bytes_Available, EUSART_Peek);
64 }

```

```

65
66 char I2C_W(int Slave_Addr,unsigned char Reg_Addr,unsigned char packets,unsigned
67   char packet_1,unsigned char packet_2,unsigned char packet_3,unsigned char
68   packet_4) //Slave_Addr = slave address;Reg_Addr=start register for write;
69   packets=number of data bytes to send; packet_x = write packets;
70
71 {
72     int idle_count = 0;
73
74     writeBuffer[0] = Reg_Addr; // Pointer for 1st register
75     writeBuffer[1] = packet_1; // 1st data register
76     writeBuffer[2] = packet_2; // 2nd data register
77     writeBuffer[3] = packet_3; // 3rd data register
78     writeBuffer[4] = packet_4; // 4th data register
79     I2C2_MasterWrite(writeBuffer,(packets+1),Slave_Addr,&status);
80     ms_delay1(ms_1);
81     while (status == I2C2_MESSAGE_PENDING){
82         respond_to_MCB();
83
84         checkCommsEnable();
85     }
86     if (status == I2C2_MESSAGE_COMPLETE)
87     {
88         idle_count = 0;
89         return 0;
90     }
91     if (status == I2C2_MESSAGE_ADDRESS_NO_ACK)
92     {
93         return 1;
94     }
95     if (status == I2C2_MESSAGE_FAIL)
96     {
97         idle_count = 0;
98         return 2;
99     }
100    if (status == I2C2_STUCK_START)
101    {
102        return 3;
103    }
104    if (status == I2C2_DATA_NO_ACK)
105    {
106        return 4;
107    }
108    if (status == I2C2_LOST_STATE)
109    {
110        return 5;
111    }
112    else
113        return 6; //LIDAR read malfunction
114
115 unsigned char I2C_R(unsigned int Slave_Addr,unsigned char Reg_Addr,unsigned char
116   packets)
117 {
118     I2C_W(Slave_Addr,Reg_Addr,0,0,0,0,0);
119     I2C2_MasterRead(readBuffer,packets,Slave_Addr,&status);
120     ms_delay1(ms_1);
121     if (status == I2C2_MESSAGE_COMPLETE)
122     {
123         return 0;
124     }
125     if (status == I2C2_MESSAGE_ADDRESS_NO_ACK)

```

```

125     {
126         return 1;
127     }
128     if (status == I2C2_MESSAGE_FAIL)
129     {
130         return 2;
131     }
132     if (status == I2C2_STUCK_START)
133     {
134         return 3;
135     }
136     if (status == I2C2_DATA_NO_ACK)
137     {
138         return 4;
139     }
140     if (status == I2C2_LOST_STATE)
141     {
142         return 5;
143     }
144 }
145
146
147 void LIDAR_Config(void){
148 //choose mode according to preference
150
151 #ifdef balanced_mode
152     I2C_W(0x62, 0x02, 1, 0x80, 0, 0, 0); //maximum acquisition count config
153     I2C_W(0x62, 0x04, 1, 0x08, 0, 0, 0); //acquisition mode control config
154     I2C_W(0x62, 0x1c, 1, 0x00, 0, 0, 0); //threshold bypass config
155 #endif
156
157 #ifdef short_range_high_speed
158     I2C_W(0x62, 0x02, 1, 0x1D, 0, 0, 0); //maximum acquisition count config
159     I2C_W(0x62, 0x04, 1, 0x08, 0, 0, 0); //acquisition mode control config
160     I2C_W(0x62, 0x1c, 1, 0x00, 0, 0, 0); //threshold bypass config
161
162 #endif
163
164 #ifdef higher_speed_short_range
165     I2C_W(0x62, 0x02, 1, 0x80, 0, 0, 0); //maximum acquisition count config
166     I2C_W(0x62, 0x04, 1, 0x00, 0, 0, 0); //acquisition mode control config
167     I2C_W(0x62, 0x1c, 1, 0x00, 0, 0, 0); //threshold bypass config
168
169 #endif
170
171 #ifdef max_range
172     I2C_W(0x62, 0x02, 1, 0xff, 0, 0, 0); //maximum acquisition count config
173     I2C_W(0x62, 0x04, 1, 0x08, 0, 0, 0); //acquisition mode control config
174     I2C_W(0x62, 0x1c, 1, 0x00, 0, 0, 0); //threshold bypass config
175
176 #endif
177
178 #ifdef high_sense_high_error
179     I2C_W(0x62, 0x02, 1, 0x80, 0, 0, 0); //maximum acquisition count config
180     I2C_W(0x62, 0x04, 1, 0x08, 0, 0, 0); //acquisition mode control config
181     I2C_W(0x62, 0x1c, 1, 0x80, 0, 0, 0); //threshold bypass config
182
183 #endif
184
185 #ifdef low_sense_low_error
186     I2C_W(0x62, 0x02, 1, 0x80, 0, 0, 0); //maximum acquisition count config
187     I2C_W(0x62, 0x04, 1, 0x08, 0, 0, 0); //acquisition mode control config
188     I2C_W(0x62, 0x1c, 1, 0xB0, 0, 0, 0); //threshold bypass config
189

```

```

190     #endif
191
192     #ifdef custom_mode1
193         I2C_W(0x62, 0x02, 1, 0x80, 0, 0, 0); //maximum acquisition count config
194         I2C_W(0x62, 0x04, 1, 0x08, 0, 0, 0); //acquisition mode control config
195         I2C_W(0x62, 0x1c, 1, 0x00, 0, 0, 0); //threshold bypass config
196         I2C_W(0x62, 0x11, 1, 0xff, 0, 0, 0); //burst measurement control
197
198     #endif
199
200     #ifdef custom_mode2
201         I2C_W(0x62, 0x02, 1, 0x80, 0, 0, 0); //maximum acquisition count config
202         I2C_W(0x62, 0x04, 1, 0x00, 0, 0, 0); //acquisition mode control config
203         I2C_W(0x62, 0x1c, 1, 0x00, 0, 0, 0); //threshold bypass config
204         I2C_W(0x62, 0x11, 1, 0xff, 0, 0, 0); //burst measurement control
205
206     #endif
207
208
209 }
210
211 unsigned int LIDAR_Get_Distance(void)
212 {
213     unsigned char LIDAR_busy_check;
214     unsigned char busy_bit = 1;
215     unsigned char distanceArray[2];
216     int distance_Value;
217
218     LIDAR_busy_check = I2C_R(0x62, 0x01, 1); //read busy status
219     busy_bit = (LIDAR_busy_check & 0x01);
220     if(busy_bit == 0){
221
222
223         if(receiver_bias_count < 100){
224             I2C_W(0x62, 0x00, 1, 0x03, 0, 0, 0); //get distance without receiver
225             bias correction
226         }
227         else{
228             I2C_W(0x62, 0x00, 1, 0x04, 0, 0, 0); //get distance with receiver bias
229             correction
230         }
231         I2C_R(0x62, 0x8f, 2); //distance high and low byte, gets high byte first
232         distanceArray[0]= readBuffer[0];
233         distanceArray[1]= readBuffer[1];
234
235         distance_Value = (distanceArray[0] << 8) + distanceArray[1];
236
237         receiver_bias_count++;
238         if(receiver_bias_count > 100){
239             receiver_bias_count = 0;
240         }
241
242         return (distance_Value);
243
244     }
245     else{
246         return 50000; //50000 indicates busy
247     }
248
249 void execute_Read(void){
250
251     #ifdef no_array
252

```

```

253     LIDAR_Value = LIDAR_Get_Distance();
254     Index_snapshot();
255
256     if(LIDAR_Value < 1800 && LIDAR_Value >= 25){ //acknowledge data if
257         distance falls between 2 values, else discard
258         if(LIDAR_Value > 30)
259         {
260             LATCbits.LATC7 = 0x01; //LED D2
261         }
262         if(LIDAR_Value <= 30)
263         {
264             LATCbits.LATC7 = 0x00; //LED D2
265         }
266         x++;
267         if(x >= 50){
268             x = 0;
269         }
270     }
271
272     index_compare = index_match();
273
274
275     if(index_compare == 1){
276         TMR0_check_Inactive(cycle_count);
277         LIDAR_Value = 0;
278     }
279     if(index_compare != 1){
280         TMR0_StopTimer();
281         TMR0_clear_ONS();
282         weapon_inactivity = 0;
283         cycle_count = 0;
284     }
285
286     TMR0_overflow = TMR0_HasOverflowOccured();
287
288     if(TMR0_overflow == 1){
289         weapon_inactivity = 1;
290     }
291
292     if(index_compare != 1){
293         filtered_distance = calculate_wboxc(LIDAR_Value);
294     }
295
296     // impedance)
297     // LATBbits.LATB6 = 0x01; //high level enables driver output (low
298     // setTransmitStall(false);
299     // continuous_send();
300     //debug
301
302     respond_to_MCB();
303
304     if(LIDAR_Value < 25){
305         cycle_count = 1; //indicates first read is done
306     }
307
308
309
310 #endif
311
312 #ifdef continuous_trigger
313 //
314 //     LIDAR_Read_Val[x] = LIDAR_Get_Distance();
315 //     Index_snapshot();

```

```
316 //          if(LIDAR_Read_Val[x] < 1800 && LIDAR_Read_Val[x] >= 25){  
317 //              acknowledge data if distance falls between 2 values, else discard  
318 //              if(LIDAR_Read_Val[x] > 30)  
319 //                  LATCbits.LATC0 = 0x00; //LED D3  
320 //                  LATCbits.LATC7 = 0x01; //LED D2  
321 //              }  
322 //              if(LIDAR_Read_Val[x] <= 30)  
323 //              {  
324 //                  LATCbits.LATC7 = 0x00; //LED D2  
325 //                  //LATCbits.LATC0 = 0x01; //LED D3  
326 //              }  
327 //  
328 //  
329 //
```

```
    ;
330 //  
331 //  
332 //      x++;  
333 //      if(x >= 50){  
334 //          x = 0;  
335 //          y = 49;  
336 //      }  
337 //      if(x != 0){  
338 //          y = x - 1;  
339 //      }  
340 //  
341 //  }  
342 // index_compare = index_match();  
343 //  
344 //
```

```

345 //          if(index_compare == 1){
346 //              TMRO_Check_Inactive(cycle_count);
347 //          }
348 //          if(index_compare != 1){
349 //              TMRO_StopTimer();
350 //              TMRO_Clear_ONS();
351 //              weapon_inactivity = 0;
352 //              cycle_count = 0;
353 //          }
354 //
355 //          TMRO_overflow = TMRO_HasOverflowOccured();
356 //
357 //          if(TMRO_overflow == 1){
358 //              weapon_inactivity = 1;
359 //          }
360 //
361 //
362 //          filtered_distance = calculate_wboxc(LIDAR_Read_Val[y]);
363 //
364 //
365 //        LATBbits.LATB6 = 0x01; //high level enables driver output
366 //        (low impedance)
367 //        setTransmitStall(false);
368 //        continuous_send();
369 //
370 //        respond_to_MCB();
371 //
372 //        if(LIDAR_Read_Val[y] < 25){
373 //            cycle_count = 1; //indicates first read is done
374 //        }
375 //
376
377
378 #endif
379
380
381
382
383 #ifndef continuous_trigger
384
385 //use with hall sensor trigger if desired
386
387 trigger = flag_Read();
388
389
390 if(trigger == 1){
391     LIDAR_Read_Val[x] = LIDAR_Get_Distance();
392     if(LIDAR_Read_Val[x] != 50000){
393         if(LIDAR_Read_Val[x] > 30){
394             //LATCbits.LATC0 = 0x00; //LED D3
395             LATCbits.LATC7 = 0x01; //LED D2
396         }
397         if(LIDAR_Read_Val[x] <= 30){
398             LATCbits.LATC7 = 0x00; //LED D2
399             //LATCbits.LATC0 = 0x01; //LED D3
400         }
401
402
403
404         x++;
405         //rpm_print_delay++;
406         if(x >= 50){
407             x = 0;
408             y = 49;

```

```

409     }
410     if(x != 0){
411         y = x - 1;
412     }
413     _RPM_pulse(rev_count);
414
415     weapon_rpm = RPM_calc();
416
417 //        if(rpm_print_delay >= 2){
418 //            send_RPM_test(); //debug RPM
419 //            rpm_print_delay = 0;
420 //        }
421 //        //continuous_send();
422
423    }
424
425    else{
426
427    }
428
429    rev_count = 1;
430    flag_Write(0);
431
432}
433
434
435
436 //        respond_to_MCB();
437 #endif
438 }
439
440 void respond_to_MCB(void){
441     if(receiveData()){
442         ToSend(1,filtered_distance);
443         ToSend(2,index_compare);
444         ToSend(3,weapon_inactivity);
445         ToSend(0,PSB_Addr);
446         sendData(1);
447     }
448     else
449     {
450
451     }
452     I2C2_Initialize();
453 }
454
455 void continuous_send(void){
456     //UART Debug - print out continuously if calibrating LIDAR
457     ToSend(1,filtered_distance);
458     sendData(1);
459     printf("Distance is: %d\r\n", filtered_distance);
460     I2C2_Initialize();
461 }
462
463 void Index_snapshot(void){
464     x_snapshot = x;
465 }
466
467 bool index_match(void){
468     bool status = 0;
469     if(x_snapshot == x){
470         status = true;
471     }
472     else {
473         status = false;

```

```
474     }
475     return status;
476 }
477
478 void recalibrate_LIDAR(void){
479     //This function is intended to be used for 'soft rebooting' the LIDAR
480
481     TMR0_Wipe();
482     ms_delay30(ms_30); //LIDAR power up initialize delay 30 ms
483     TMR0_StopTimer();
484     LIDAR_Config();
485 }
486 //void send_RPM_test(void) {
487 //    used for calibrating RPM readings
488 //    ToSend(1, weapon_rpm);
489 //    sendData(1);
490 //    printf("RPM is: %f\r\n", weapon_rpm);
491 //    I2C2_Initialize();
492 //}
```

```

1  /*
2   * File:    LIDAR.h
3   * Author:  Mike Hritz
4   *
5   * Created on February 13, 2018, 11:05 AM
6   */
7
8 #ifndef LIDAR_H
9 #define LIDAR_H
10
11 #include <xc.h>
12 #include <stdbool.h>
13 #include <stdint.h>
14 #include <stdio.h>
15
16 #ifdef __cplusplus
17 extern "C" {
18 #endif
19
20 char I2C_W(int Slave_Addr,unsigned char Reg_Addr,unsigned char packets,unsigned
21 char packet_1,unsigned char packet_2,unsigned char packet_3,unsigned char
22 packet_4);
23
24 unsigned char I2C_R(unsigned int Slave_Addr,unsigned char Reg_Addr,unsigned char
25 packets);
26
27 void LIDAR_Config(void);
28
29 unsigned int LIDAR_Get_Distance(void);
30
31 void Fast_Xfer_Config(int addr);
32
33 void execute_Read(void);
34
35 void respond_to_MCB(void);
36
37 void continuous_send(void);
38
39 void send_RPM_test(void);
40
41 void Index_snapshot(void);
42
43 bool index_match(void);
44
45 void recalibrate_LIDAR(void);
46
47 #endif /* LIDAR_H */

```

```

1  /*
2   * File:    RPM_FIFO.c
3   * Author:  Mike Hritz
4   *
5   * Created on February 16, 2018, 9:45 AM
6   *
7   * Intended for measuring the weapon RPM by utilizing hall sensors
8   */
9
10
11 #include "mcc_generated_files/mcc.h"
12 #include "mcc_generated_files/tmr0.h"
13
14 int TMR_Val[2] = {0,0};
15 int FIFO[2] = {0,0}; //FIFO shift register
16 int ONS = 0;
17 int start_RPM_calc = 0;
18 float real_time = 0;
19 int rollovers = 0;
20 float real_RPM = 0;
21 float RPM_to_MCB = 0;
22
23 int FIFO_ONS(void){ //FIFO OneShot function
24     FIFO[0] = TMR3_ReadTimer();
25
26     return FIFO[0];
27 }
28
29 int FIFO_shift(void){ //shift array
30     FIFO[1] = FIFO[0];
31
32     return FIFO[1];
33 }
34
35
36 int FIFO_read_TMR(void){ //overwrite first array element
37     FIFO[0] = TMR3_ReadTimer();
38     rollovers = read_rollover_count();
39     reset_rollover_count();
40
41     return FIFO[0];
42 }
43
44 void _RPM_pulse(int rev_count){
45
46     if(ONS < 1){ //just load the first timer value into start of array
47         TMR_Val[0] = FIFO_ONS();
48         ONS = 1;
49     }
50     else if(ONS == 1 && rev_count == 1){ //start continuous FIFO
51         TMR_Val[1] = FIFO_shift();
52         TMR_Val[0] = FIFO_read_TMR();
53
54         start_RPM_calc = 1;
55     }
56     else{
57
58     }
59 }
60
61 float RPM_calc(void){
62     real_time = 0;
63     real_RPM = 0;
64     RPM_to_MCB = 0;

```

```
66     if(start_RPM_calc == 1){  
67         real_time = (((65535*rollovers) - TMR_Val[0])) +  
68             ((TMR_Val[1]))/8000000.0; //actual conversion  
69         real_RPM = (1/(real_time))*60;  
70         RPM_to_MCB = real_RPM;  
71         rollovers = 0;  
72     }  
73     return RPM_to_MCB;  
74 }  
75 }
```

```
1  /*
2   * File:    RPM_FIFO.h
3   * Author:  Mike Hritz
4   *
5   * Created on February 16, 2018, 9:45 AM
6   */
7
8 #ifndef RPM_FIFO_H
9 #define RPM_FIFO_H
10
11 #include <xc.h>
12 #include <stdbool.h>
13 #include <stdint.h>
14 #include <stdio.h>
15
16 #ifdef __cplusplus
17 extern "C" {
18#endif
19
20 int FIFO_ONS(int count);
21
22 int FIFO_shift(void);
23
24 int FIFO_read_TMR(void);
25
26 void _RPM_pulse(int rev_count);
27
28 float RPM_calc(void);
29
30 #ifdef __cplusplus
31 }
32#endif
33
34#endif /* RPM_FIFO_H */
```



```
54         (wboxc_distance_array[config_1]) + (wboxc_distance_array[config_1])
55         + (wboxc_distance_array[config_2])) >> 3;
56     continuous_flag = 1;
57 }
58 wboxc_index++;
59 if(wboxc_index >= 3){
60     wboxc_index = 0;
61 }
62 start_wboxc = 1;
63 return wboxc_distance;
64 }
```

```
1  /*
2   * File:    WBOXC_Filter.h
3   * Author:  Mike Hritz
4   *
5   * Created on April 8, 2018, 3:00 PM
6   */
7
8 #ifndef WBOXC_Filter_H
9 #define WBOXC_Filter_H
10
11 #include <xc.h>
12 #include <stdbool.h>
13 #include <stdint.h>
14 #include <stdio.h>
15
16 #ifdef __cplusplus
17 extern "C" {
18 #endif
19
20 int calculate_wboxc(unsigned int LIDAR_val);
21
22 #ifdef __cplusplus
23 }
24 #endif
25
26#endif /* WBOXC_Filter_H */
```

```

1  /**
2   * EUSART Generated Driver File
3
4   * @Company
5   *   Microchip Technology Inc.
6
7   * @File Name
8   *   eusart.c
9
10  * @Summary
11  *   This is the generated driver implementation file for the EUSART driver using
12  *   MPLAB(c) Code Configurator
13  * @Description
14  *   This header file provides implementations for driver APIs for EUSART.
15  *   Generation Information :
16  *     Product Revision : MPLAB(c) Code Configurator - 4.15.5
17  *     Device          : PIC16F18345
18  *     Driver Version  : 2.00
19  *     The generated drivers are tested against the following:
20  *       Compiler      : XC8 1.35
21  *       MPLAB         : MPLAB X 3.40
22  */
23
24  /*
25  *   (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
26  *   software and any derivatives exclusively with Microchip products.
27
28  * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
29  * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
30  * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
31  * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
32  * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
33
34  * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
35  * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
36  * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
37  * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
38  * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
39  * ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
40  * THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
41
42  * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
43  * TERMS.
44  */
45
46  /**
47  *   Section: Included Files
48  */
49  #include "eusart.h"
50  #include "tmr2.h"
51  #define us_250 0xE5 //250 us delay
52
53
54  /**
55  *   Section: Macro Declarations
56  */
57  #define EUSART_TX_BUFFER_SIZE 64
58  #define EUSART_RX_BUFFER_SIZE 256
59
60  /**
61  *   Section: Global Variables
62  */
63  volatile bool transmitStall=true, lastTransmitStall=false;
64  volatile uint8_t eusartTxHead = 0;

```

```

65 volatile uint8_t eusartTxTail = 0;
66 volatile uint8_t eusartTxBuffer[EUSART_TX_BUFFER_SIZE];
67 volatile uint8_t eusartTxBufferRemaining;
68
69 volatile uint8_t eusartRxHead = 0;
70 volatile uint8_t eusartRxTail = 0;
71 volatile uint8_t eusartRxBuffer[EUSART_RX_BUFFER_SIZE];
72 volatile uint8_t eusartRxCount;
73
74 /**
75  * Section: EUSART APIs
76 */
77
78 void checkCommsEnable(void)
79 {
80     if(getTransmitStallReset())
81     {
82         setLastTransmitStall(true);
83         LATBbits.LATB6 = 0x00; //low level puts driver output in high impedance
84     }
85 }
86 bool getTransmitStallReset(void)
87 {
88     return (transmitStall && !lastTransmitStall && TMR2_HasOverflowOccured());
89 }
90 void setLastTransmitStall(bool b)
91 {
92     lastTransmitStall=b;
93 }
94 void setTransmitStall(bool b)
95 {
96     transmitStall = b;
97 }
98 void EUSART_Initialize(void)
99 {
100    // disable interrupts before changing states
101    PIE1bits.RCIE = 0;
102    PIE1bits.TXIE = 0;
103
104    // Set the EUSART module to the options selected in the user interface.
105
106    // ABDOVF no_overflow; SCKP Non-Inverted; BRG16 16bit_generator; WUE
107    // disabled; ABDEN disabled;
108    BAUD1CON = 0x08;
109
110    // SPEN enabled; RX9 8-bit; CREN enabled; ADDEN disabled; SREN disabled;
111    RC1STA = 0x90;
112
113    // TX9 8-bit; TX9D 0; SENDB sync_break_complete; TXEN enabled; SYNC
114    // asynchronous; BRGH hi_speed; CSRC slave;
115    TX1STA = 0x24;
116
117    // Baud Rate = 115200; SP1BRGL 68;
118    SP1BRGL = 0x44;
119
120    // Baud Rate = 115200; SP1BRGH 0;
121    SP1BRGH = 0x00;
122
123    // initializing the driver state
124    eusartTxHead = 0;
125    eusartTxTail = 0;
126    eusartTxBufferRemaining = sizeof(eusartTxBuffer);
127    eusartRxHead = 0;

```

```

128     eusartRxTail = 0;
129     eusartRxCount = 0;
130
131     // enable receive interrupt
132     PIE1bits.RCIE = 1;
133 }
134
135 uint8_t EUSART_Read(void)
136 {
137     uint8_t readValue = 0;
138
139     while(0 == eusartRxCount)
140     {
141     }
142
143     readValue = eusartRxBuffer[eusartRxTail++];
144     eusartRxBuffer[eusartRxTail-1]=0;
145     if(sizeof(eusartRxBuffer) <= eusartRxTail)
146     {
147         eusartRxTail = 0;
148     }
149     PIE1bits.RCIE = 0;
150     eusartRxCount--;
151     PIE1bits.RCIE = 1;
152
153     return readValue;
154 }
155
156 unsigned char EUSART_Peek(void)
157 {
158     unsigned char readValue = 0;
159
160     readValue = eusartRxBuffer[eusartRxTail];
161
162
163     return readValue;
164 }
165
166
167 int EUSART_Bytes_Available(void){
168     return eusartRxCount;
169 }
170
171 void EUSART_Write(uint8_t txData)
172 {
173     while(0 == eusartTxBufferRemaining)
174     {
175     }
176
177     if(0 == PIE1bits.TXIE)
178     {
179         TX1REG = txData;
180     }
181     else
182     {
183         PIE1bits.TXIE = 0;
184         eusartTxBuffer[eusartTxHead++] = txData;
185         if(sizeof(eusartTxBuffer) <= eusartTxHead)
186         {
187             eusartTxHead = 0;
188         }
189         eusartTxBufferRemaining--;
190     }
191     TMR2_Wipe();
192     transmitStall=false;

```

```

193     PIE1bits.TXIE = 1;
194 }
195
196 unsigned char getch( void)
197 {
198     return EUSART_Read();
199 }
200
201 void putch(unsigned char txData)
202 {
203     EUSART_Write(txData);
204 }
205
206 void EUSART_Transmit_ISR(void)
207 {
208
209 // add your EUSART interrupt custom code
210 if(sizeof(eusartTxBuffer) > eusartTxBufferRemaining)
211 {
212     TX1REG = eusartTxBuffer[eusartTxTail++];
213     if(sizeof(eusartTxBuffer) <= eusartTxTail)
214     {
215         eusartTxTail = 0;
216     }
217     eusartTxBufferRemaining++;
218 }
219 else
220 {
221     transmitStall=true;
222     lastTransmitStall=false;
223     TMR2_Wipe();
224 //     us_delay250(us_250);
225 //     LATBbits.LATB6 = 0x00; //low level puts driver output in high impedance
226     PIE1bits.TXIE = 0;
227 }
228 }
229
230 void EUSART_Receive_ISR(void)
231 {
232
233 if(1 == RC1STAbits.OERR)
234 {
235     // EUSART error - restart
236
237     RC1STAbits.CREN = 0;
238     RC1STAbits.CREN = 1;
239 }
240
241 // buffer overruns are ignored
242 eusartRxBuffer[eusartRxHead++] = RC1REG;
243 if(sizeof(eusartRxBuffer) <= eusartRxHead)
244 {
245     eusartRxHead = 0;
246 }
247 eusartRxCount++;
248 }
249 /**
250 End of File
251 */

```

```

1  /**
2   EUSART Generated Driver API Header File
3
4  @Company
5   Microchip Technology Inc.
6
7  @File Name
8   eusart.h
9
10 @Summary
11   This is the generated header file for the EUSART driver using MPLAB(c) Code
12   Configurator
13 @Description
14   This header file provides APIs for driver for EUSART.
15   Generation Information :
16     Product Revision : MPLAB(c) Code Configurator - 4.15.1
17     Device          : PIC16F18345
18     Driver Version  : 2.00
19     The generated drivers are tested against the following:
20       Compiler      : XC8 1.35
21       MPLAB         : MPLAB X 3.40
22 */
23
24 /*
25   (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
26   software and any derivatives exclusively with Microchip products.
27
28 THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
29 EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
30 WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
31 PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
32 WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
33
34 IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
35 INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
36 WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
37 BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
38 FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
39 ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
40 THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
41
42 MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
43 TERMS.
44 */
45
46 #ifndef _EUSART_H
47 #define _EUSART_H
48
49 /**
50   Section: Included Files
51 */
52
53 #include <xc.h>
54 #include <stdbool.h>
55 #include <stdint.h>
56 #include <stdio.h>
57
58 #ifdef __cplusplus // Provide C++ Compatibility
59
60     extern "C" {
61
62 #endif
63
64 /**

```

```

65     Section: Macro Declarations
66 */
67
68 #define EUSART_DataReady (eusartRxCount)
69
70 /**
71   Section: Data Type Definitions
72 */
73
74 /**
75   Section: Global variables
76 */
77 extern volatile uint8_t eusartTxBufferRemaining;
78 extern volatile uint8_t eusartRxCount;
79
80
81 /**
82   Section: EUSART APIs
83 */
84
85 /**
86   @Summary
87     Initialization routine that takes inputs from the EUSART GUI.
88
89   @Description
90     This routine initializes the EUSART driver.
91     This routine must be called before any other EUSART routine is called.
92
93   @Preconditions
94     None
95
96   @Param
97     None
98
99   @Returns
100    None
101
102   @Comment
103
104   @Example
105 */
106 void EUSART_Initialize(void);
107
108 /**
109   @Summary
110     Read a byte of data from the EUSART.
111
112   @Description
113     This routine reads a byte of data from the EUSART.
114
115   @Preconditions
116     EUSART_Initialize() function should have been called
117     before calling this function. The transfer status should be checked to see
118     if the receiver is not empty before calling this function.
119
120   EUSART_DataReady is a macro which checks if any byte is received.
121   Call this macro before using this function.
122
123   @Param
124     None
125
126   @Returns
127     A data byte received by the driver.
128
129   @Example

```

```

130      <code>
131          void main(void) {
132              // initialize the device
133              SYSTEM_Initialize();
134              uint8_t data;
135
136              // Enable the Global Interrupts
137              INTERRUPT_GlobalInterruptEnable();
138
139              // Enable the Peripheral Interrupts
140              INTERRUPT_PeripheralInterruptEnable();
141
142              printf("\t\tTEST CODE\n\r");           //Enable
143              redirect STUDIO to USART before using printf
144              statements
145              printf("\t\t---- ----\n\r");
146              printf("\t\tECHO TEST\n\r");
147              printf("\t\t---- ----\n\r");
148              printf("Enter any string: ");
149              do{
150                  data = EUSART1_Read();           // Read data received
151                  EUSART_Write(data);           // Echo back the
152                  data received
153                  }while(!EUSART1_DataReady);     //check if any
154                  data is received
155
156      </code>
157  */
158  uint8_t EUSART_Read(void);
159
160 /**
161  * @Summary
162  *   Writes a byte of data to the EUSART.
163
164  * @Description
165  *   This routine writes a byte of data to the EUSART.
166
167  * @Preconditions
168  *   EUSART_Initialize() function should have been called
169  *   before calling this function. The transfer status should be checked to see
170  *   if transmitter is not busy before calling this function.
171
172  * @Param
173  *   txData - Data byte to write to the EUSART
174
175  * @Returns
176  *   None
177
178  * @Example
179  *   <code>
180  *     Refer to EUSART_Read() for an example
181  *   </code>
182  */
183 void EUSART_Write(uint8_t txData);
184
185 /**
186  * @Summary
187  *   Maintains the driver's transmitter state machine and implements its ISR.
188
189  * @Description
190  *   This routine is used to maintain the driver's internal transmitter state

```

```

191     @Preconditions
192         EUSART_Initialize() function should have been called
193         for the ISR to execute correctly.
194
195     @Param
196         None
197
198     @Returns
199         None
200     */
201 void EUSART_Transmit_ISR(void);
202
203 /**
204     @Summary
205         Maintains the driver's receiver state machine and implements its ISR
206
207     @Description
208         This routine is used to maintain the driver's internal receiver state
209         machine. This interrupt service routine is called when the state of the
210         receiver needs to be maintained in a non polled manner.
211
212     @Preconditions
213         EUSART_Initialize() function should have been called
214         for the ISR to execute correctly.
215
216     @Param
217         None
218
219     @Returns
220         None
221     */
222 void EUSART_Receive_ISR(void);
223
224 uint8_t EUSART_Peek(void);
225
226 int EUSART_Bytes_Available(void);
227 unsigned char getch( void );
228 void putch(unsigned char txData);
229
230 bool getTransmitStallReset(void);
231 void setTransmitStall(bool b);
232 void setLastTransmitStall(bool b);
233
234 void checkCommsEnable(void);
235
236
237 #ifdef __cplusplus // Provide C++ Compatibility
238 }
239
240#endif
241#endif // _EUSART_H
242 /**
243     End of File
244 */
245

```

```

1  /**
2   *      EXT_INT Generated Driver File
3
4   * @Company:
5   *      Microchip Technology Inc.
6
7   * @File Name:
8   *      ext_int.c
9
10  * @Summary
11  *      This is the generated driver implementation file for the EXT_INT
12  *      driver using MPLAB(c) Code Configurator
13
14  * @Description:
15  *      This source file provides implementations for driver APIs for EXT_INT.
16  * Generation Information :
17  *      Product Revision : MPLAB(c) Code Configurator - 4.15.1
18  *      Device          : PIC16F18345
19  *      Driver Version  : 1.0
20  *      The generated drivers are tested against the following:
21  *      Compiler        : XC8 1.35
22  *      MPLAB          : MPLAB X 3.40
23 */
24
25 /**
26   *      Section: Includes
27   */
28 #include <xc.h>
29 #include "ext_int.h"
30
31 int flag = 0;
32
33 //***User Area Begin->code: Add External Interrupt handler specific headers
34
35 //***User Area End->code: Add External Interrupt handler specific headers
36
37 /**
38   *      Section: External Interrupt Handlers
39   */
40 // INTn Dynamic Interrupt Handlers
41 void (*INT_InterruptHandler)(void);
42
43 /**
44   *      Interrupt Handler for EXT_INT - INT
45   */
46
47 int flag_Read(void){
48
49     return flag;
50 }
51
52 void flag_Write(int clear){
53     flag = clear;
54 }
55
56 void INT_ISR(void)
57 {
58     //***User Area Begin->code***
59     flag = 1;
60
61     //***User Area End->code***
62
63     EXT_INT_InterruptFlagClear();
64
65     // Callback function gets called everytime this ISR executes

```

```

66     INT_CallBack();
67 }
68 /**
69  * Callback function for EXT_INT - INT
70 */
71 void INT_CallBack(void)
72 {
73     // Add your custom callback code here
74     if(INT_InterruptHandler)
75     {
76
77         INT_InterruptHandler();
78     }
79 }
80 /**
81  * Allows selecting an interrupt handler for EXT_INT - INT at application runtime
82 */
83 void INT_SetInterruptHandler(void* InterruptHandler){
84     INT_InterruptHandler = InterruptHandler;
85 }
86 /**
87  * Default interrupt handler for EXT_INT - INT
88 */
89 void INT_DefaultInterruptHandler(void){
90     // add your INT interrupt custom code
91     // or set custom function using INT_SetInterruptHandler()
92 }
93 /**
94  * Section: External Interrupt Initializers
95 */
96 /**
97  * void EXT_INT_Initialize(void)
98
99 /**
100    Initializer for the following external interrupts
101    INT
102 */
103 void EXT_INT_Initialize(void)
104 {
105
106     *****
107     * INT
108     * Clear the interrupt flag
109     * Set the external interrupt edge detect
110     * Enable the interrupt, if enabled in the UI.
111     *****/
112     EXT_INT_InterruptFlagClear();
113     EXT_INT_risingEdgeSet();
114     // Set Default Interrupt Handler
115     INT_SetInterruptHandler(INT_DefaultInterruptHandler());
116     EXT_INT_InterruptEnable();
117
118 }
119
120 }
```

```

1  /**
2   * EXT_INT Generated Driver API Header File
3
4  @Company:
5   Microchip Technology Inc.
6
7  @File Name:
8   ext_int.h
9
10 @Summary:
11   This is the generated header file for the EXT_INT driver using MPLAB(c) Code
12  Configurator
13 @Description:
14   This header file provides APIs for driver for EXT_INT.
15  Generation Information :
16    Product Revision : MPLAB(c) Code Configurator - 4.15.1
17    Device          : PIC16F18345
18    Driver Version  : 1.0
19    The generated drivers are tested against the following:
20    Compiler        : XC8 1.35
21    MPLAB          : MPLAB X 3.40
22 */
23
24 /*
25  (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
26  software and any derivatives exclusively with Microchip products.
27
28 THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
29 EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
30 WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
31 PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
32 WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
33
34 IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
35 INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
36 WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
37 BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
38 FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
39 ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
40 THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
41
42 MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
43 TERMS.
44 */
45
46 #ifndef _EXT_INT_H
47 #define _EXT_INT_H
48
49 /**
50  * Section: Includes
51 */
52 #include <xc.h>
53 #include <stdbool.h>
54 #include <stdint.h>
55 #include <stdio.h>
56 #include "../LIDAR.h"
57
58 // Provide C++ Compatibility
59 #ifdef __cplusplus
60
61 extern "C" {
62
63 #endif

```

```

65  /**
66   *      Section: Macros
67   */
68 /**
69  /**
70  @Summary
71  Clears the interrupt flag for INT
72
73 @Description
74  This routine clears the interrupt flag for the external interrupt, INT.
75
76 @Preconditions
77  None.
78
79 @Returns
80  None.
81
82 @Param
83  None.
84
85 @Example
86 <code>
87 void INT_ISR(void)
88 {
89     // User Area Begin->code
90
91     // User Area End->code
92     EXT_INT_InterruptFlagClear();
93 }
94 </code>
95
96 /**
97 #define EXT_INT_InterruptFlagClear()      (PIR0bits.INTF = 0)
98
99 /**
100 @Summary
101 Clears the interrupt enable for INT
102
103 @Description
104  This routine clears the interrupt enable for the external interrupt, INT.
105  After calling this routine, external interrupts on this pin will not be
106  serviced by the
107  interrupt handler, INT.
108
109 @Preconditions
110  None.
111
112 @Returns
113  None.
114
115 @Param
116  None.
117
118 @Example
119  Changing the external interrupt edge detect from negative to positive
120 <code>
121 // clear the interrupt enable
122 EXT_INT_InterruptDisable();
123 // change the edge
124 EXT_INT_risingEdgeSet();
125 // clear the interrupt flag and re-enable the interrupt
126 EXT_INT_InterruptFlagClear();
127 EXT_INT_InterruptEnable();
128 </code>

```

```

129  */
130 #define EXT_INT_Disable()          (PIE0bits.INTE = 0)
131 /**
132  */
133 /**
134   @Summary
135   Sets the interrupt enable for INT
136
137   @Description
138   This routine sets the interrupt enable for the external interrupt, INT.
139   After calling this routine, external interrupts on this pin will be serviced
140   by the
141   interrupt handler, INT.
142
143   @Preconditions
144   None.
145
146   @Returns
147   None.
148
149   @Param
150   None.
151
152   @Example
153   Setting the external interrupt to handle positive edge interrupts
154   <code>
155   // set the edge
156   EXT_INT_risingEdgeSet();
157   // clear the interrupt flag and enable the interrupt
158   EXT_INT_InterruptFlagClear();
159   EXT_INT_InterruptEnable();
160   </code>
161
162 /**
163  */
164 /**
165   @Summary
166   Sets the edge detect of the external interrupt to negative edge.
167
168   @Description
169   This routine set the edge detect of the extern interrupt to negative edge.
170   After this routine is called the interrupt flag will be set when the external
171   interrupt pins level transitions from a high to low level.
172
173   @Preconditions
174   None.
175
176   @Returns
177   None.
178
179   @Param
180   None.
181
182   @Example
183   Setting the external interrupt to handle negative edge interrupts
184   <code>
185   // set the edge
186   EXT_INT_risingEdgeSet();
187   // clear the interrupt flag and enable the interrupt
188   EXT_INT_InterruptFlagClear();
189   EXT_INT_InterruptEnable();
190   </code>
191
192 /**
193  */

```

```

193 /**
194  * @Summary
195  *   Sets the edge detect of the external interrupt to positive edge.
196  *
197  * @Description
198  *   This routine set the edge detect of the extern interrupt to positive edge.
199  *   After this routine is called the interrupt flag will be set when the external
200  *   interrupt pins level transitions from a low to high level.
201  *
202  * @Preconditions
203  *   None.
204  *
205  * @Returns
206  *   None.
207  *
208  * @Param
209  *   None.
210  *
211  * @Example
212  *   Setting the external interrupt to handle positive edge interrupts
213  *   <code>
214  *     // set the edge
215  *     EXT_INT_risingEdgeSet();
216  *     // clear the interrupt flag and enable the interrupt
217  *     EXT_INT_InterruptFlagClear();
218  *     EXT_INT_InterruptEnable();
219  *   </code>
220  *
221  */
222 #define EXT_INT_fallingEdgeSet()          (INTCONbits.INTEDG = 0)
223 /**
224  * @Section: External Interrupt Initializers
225  */
226 /**
227  * @Summary
228  *   Initializes the external interrupt pins
229  *
230  * @Description
231  *   This routine initializes the EXT_INT driver to detect the configured edge,
232  *   clear
233  *   the interrupt flag and enable the interrupt.
234  *
235  * The following external interrupts will be initialized:
236  * INT - EXT_INT
237  *
238  * @Preconditions
239  *   None.
240  *
241  * @Returns
242  *   None.
243  *
244  * @Param
245  *   None.
246  *
247  * @Example
248  *   Initialize the external interrupt pins
249  *   <code>
250  *   //
251  *   EXT_INT_Initialize();
252  *   </code>
253  */
254 void EXT_INT_Initialize(void);
255

```

```

257 /**
258   Section: External Interrupt Handlers
259 */
260 /**
261   @Summary
262     Interrupt Service Routine for EXT_INT - INT pin
263
264   @Description
265     This ISR will execute whenever the signal on the INT pin will transition to
266     the preconfigured state.
267
268   @Preconditions
269     EXT_INT initializer called
270
271   @Returns
272     None.
273
274   @Param
275     None.
276
277 */
278 void INT_ISR(void);
279 /**
280   @Summary
281     Callback function for EXT_INT - INT
282
283   @Description
284     Allows for a specific callback function to be called in the INT ISR.
285     It also allows for a non-specific interrupt handler to be called at runtime.
286
287   @Preconditions
288     EXT_INT initializer called
289
290   @Returns
291     None.
292
293   @Param
294     None.
295
296   @Example
297     EXT_INT_Initializer();
298     void INT_CallBack();
299
300 */
301 void INT_CallBack(void);
302
303 /**
304   @Summary
305     Interrupt Handler Setter for EXT_INT - INT pin
306
307   @Description
308     Allows selecting an interrupt handler for EXT_INT - INT at application runtime
309
310   @Preconditions
311     EXT_INT initializer called
312
313   @Returns
314     None.
315
316   @Param
317     InterruptHandler function pointer.
318
319   @Example
320     EXT_INT_Initializer();

```

```

321     INT_SetInterruptHandler(MyInterruptHandler);
322
323 */
324 void INT_SetInterruptHandler(void* InterruptHandler);
325
326 /**
327  * @Summary
328  *   Dynamic Interrupt Handler for EXT_INT - INT pin
329
330  * @Description
331  *   This is the dynamic interrupt handler to be used together with the
332  *   INT_SetInterruptHandler() method.
333  *   This handler is called every time the INT ISR is executed and allows any
334  *   function to be registered at runtime.
335
336  * @Preconditions
337  *   EXT_INT initializer called
338
339  * @Returns
340  *   None.
341
342  * @Param
343  *   None.
344
345  * @Example
346  *   EXT_INT_Initializer();
347  *   INT_SetInterruptHandler(INT_InterruptHandler);
348
349 */
350 /**
351  * @Summary
352  *   Default Interrupt Handler for EXT_INT - INT pin
353
354  * @Description
355  *   This is a predefined interrupt handler to be used together with the
356  *   INT_SetInterruptHandler() method.
357  *   This handler is called every time the INT ISR is executed.
358
359  * @Preconditions
360  *   EXT_INT initializer called
361
362  * @Returns
363  *   None.
364
365  * @Param
366  *   None.
367
368  * @Example
369  *   EXT_INTInitializer();
370  *   INT_SetInterruptHandler(INT_DefaultInterruptHandler);
371
372 */
373 void INT_DefaultInterruptHandler(void);
374
375 int flag_Read(void);
376
377 void flag_Write(int clear);
378
379 // Provide C++ Compatibility
380 #ifdef __cplusplus
381 }
382

```

383 #endif
384 #endif

```

1  /**
2   * I2C2 Generated Driver File
3
4  @Company
5   Microchip Technology Inc.
6
7  @File Name
8   i2c2.c
9
10 @Summary
11   This is the generated header file for the I2C2 driver using MPLAB(c) Code
12  Configurator
13 @Description
14   This header file provides APIs for driver for I2C2.
15  Generation Information :
16      Product Revision : MPLAB(c) Code Configurator - 4.15.5
17      Device          : PIC16F18345
18      Driver Version  : 2.00
19      The generated drivers are tested against the following:
20      Compiler        : XC8 1.35
21      MPLAB          : MPLAB X 3.40
22 */
23
24 /*
25  (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
26  software and any derivatives exclusively with Microchip products.
27
28 THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
29 EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
30 WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
31 PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
32 WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
33
34 IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
35 INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
36 WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
37 BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
38 FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
39 ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
40 THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
41
42 MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
43 TERMS.
44 */
45
46 /**
47  Section: Included Files
48 */
49
50 #include "i2c2.h"
51
52 /**
53  I2C Driver Queue Status Type
54
55 @Summary
56   Defines the type used for the transaction queue status.
57
58 @Description
59   This defines type used to keep track of the queue status.
60 */
61
62 typedef union
63 {
64     struct

```

```

65      {
66          uint8_t full:1;
67          uint8_t empty:1;
68          uint8_t reserved:6;
69      }s;
70      uint8_t status;
71 }I2C_TR_QUEUE_STATUS;
72 /**
73  I2C Driver Queue Entry Type
74
75  @Summary
76      Defines the object used for an entry in the i2c queue items.
77
78  @Description
79      This defines the object in the i2c queue. Each entry is a composed
80      of a list of TRBs, the number of the TRBs and the status of the
81      currently processed TRB.
82 */
83  typedef struct
84  {
85      uint8_t count;           // a count of trb's in
86      the_trb_list;
87      I2C2_TRANSACTION_REQUEST_BLOCK *ptrb_list;    // pointer to the trb list
88      I2C2_MESSAGE_STATUS *pTrFlag;        // set with the error of the
89      last_trb sent.
90
91      } I2C_TR_QUEUE_ENTRY;
92
93 /**
94  I2C Master Driver Object Type
95
96  @Summary
97      Defines the object that manages the i2c master.
98
99  @Description
100     This defines the object that manages the sending and receiving of
101     i2c master transactions.
102 */
103
104  typedef struct
105  {
106      /* Read/Write Queue */
107      I2C_TR_QUEUE_ENTRY *pTrTail;      // tail of the queue
108      I2C_TR_QUEUE_ENTRY *pTrHead;      // head of the queue
109      I2C_TR_QUEUE_STATUS trStatus;    // status of the last transaction
110      uint8_t i2cDoneFlag;            // flag to indicate the
111      current                         // transaction is done
112      uint8_t i2cErrors;              // keeps track of errors
113
114  } I2C_OBJECT ;
115
116 /**
117  I2C Master Driver State Enumeration
118
119  @Summary
120      Defines the different states of the i2c master.
121
122  @Description
123      This defines the different states that the i2c master

```

```

125     used to process transactions on the i2c bus.
126 */
127
128 typedef enum
129 {
130     S_MASTER_IDLE,
131     S_MASTER_RESTART,
132     S_MASTER_SEND_ADDR,
133     S_MASTER_SEND_DATA,
134     S_MASTER_SEND_STOP,
135     S_MASTER_ACK_ADDR,
136     S_MASTER_RCV_DATA,
137     S_MASTER_RCV_STOP,
138     S_MASTER_ACK_RCV_DATA,
139     S_MASTER_NOACK_STOP,
140     S_MASTER_SEND_ADDR_10BIT_LSB,
141     S_MASTER_10BIT_RESTART,
142
143 } I2C_MASTER_STATES;
144
145 /**
146  Section: Macro Definitions
147 */
148
149 /* defined for I2C2 */
150
151 #ifndef I2C2_CONFIG_TR_QUEUE_LENGTH
152     #define I2C2_CONFIG_TR_QUEUE_LENGTH 1
153 #endif
154
155 #define I2C2_TRANSMIT_REG           SSP2BUF          // Defines the transmit register used to send data.
156 #define I2C2_RECEIVE_REG           SSP2BUF          // Defines the receive register used to receive data.
157
158 // The following control bits are used in the I2C state machine to manage
159 // the I2C module and determine next states.
160 #define I2C2_WRITE_COLLISION_STATUS_BIT    SSP2CON1bits.WCOL      // Defines the write collision status bit.
161 #define I2C2_MODE_SELECT_BITS           SSP2CON1bits.SSPM      // I2C Master Mode control bit.
162 #define I2C2_MASTER_ENABLE_CONTROL_BITS   SSP2CON1bits.SSPEN     // I2C port enable control bit.
163
164 #define I2C2_START_CONDITION_ENABLE_BIT   SSP2CON2bits.SEN       // I2C START control bit.
165 #define I2C2_REPEAT_START_CONDITION_ENABLE_BIT SSP2CON2bits.RSEN    // I2C Repeated START control bit.
166 #define I2C2_RECEIVE_ENABLE_BIT          SSP2CON2bits.RCEN      // I2C Receive enable control bit.
167 #define I2C2_STOP_CONDITION_ENABLE_BIT   SSP2CON2bits.PEN       // I2C STOP control bit.
168 #define I2C2_ACKNOWLEDGE_ENABLE_BIT     SSP2CON2bits.ACKEN     // I2C ACK start control bit.
169 #define I2C2_ACKNOWLEDGE_DATA_BIT      SSP2CON2bits.ACKDT     // I2C ACK data control bit.
170 #define I2C2_ACKNOWLEDGE_STATUS_BIT    SSP2CON2bits.ACKSTAT   // I2C ACK status bit.
171
172 #define I2C2_7bit     true
173 /**
174  Section: Local Functions
175 */
176
177 void I2C2_FunctionComplete(void);

```

```

178 void I2C2_Stop(I2C2_MESSAGE_STATUS completion_code);
179 /**
180  * Section: Local Variables
181 */
182
183 static I2C_TR_QUEUE_ENTRY
184 i2c2_tr_queue[I2C2_CONFIG_TR_QUEUE_LENGTH];
185 static I2C_OBJECT i2c2_object;
186 static I2C_MASTER_STATES i2c2_state = S_MASTER_IDLE;
187 static uint8_t i2c2_trb_count = 0;
188
189 static I2C2_TRANSACTION_REQUEST_BLOCK *p_i2c2_trb_current = NULL;
190 static I2C_TR_QUEUE_ENTRY *p_i2c2_current = NULL;
191
192 /**
193  * Section: Driver Interface
194 */
195
196
197
198 void I2C2_Initialize(void)
199 {
200     i2c2_object.pTrHead = i2c2_tr_queue;
201     i2c2_object.pTrTail = i2c2_tr_queue;
202     i2c2_object.trStatus.s.empty = true;
203     i2c2_object.trStatus.s.full = false;
204
205     i2c2_object.i2cErrors = 0;
206
207     // R_nW write_noTX; P_stopbit_notdetected; S_startbit_notdetected; BF
208     // RCinprocess_TXcomplete; SMP_High Speed; UA_dontupdate; CKE_disabled; D_nA
209     // lastbyte_address;
210     SSP2STAT = 0x00;
211     // SSPEN_enabled; WCOL_no_collision; CKP_Idle:Low, Active:High; SSPM
212     // FOSC/4_SSPxADD_I2C; SPPOV_no_overflow;
213     SSP2CON1 = 0x28;
214     // ACKTIM_ackseq; SBCDE_disabled; BOEN_disabled; SCIE_disabled; PCIE
215     // disabled; DHEN_disabled; SDAHT_100ns; AHEN_disabled;
216     SSP2CON3 = 0x00;
217     // Baud Rate Generator Value: SSP2ADD 19;
218     SSP2ADD = 0x13;
219
220     // clear the master interrupt flag
221     PIR2bits.SSP2IF = 0;
222     // enable the master interrupt
223     PIE2bits.SSP2IE = 1;
224 }
225
226 uint8_t I2C2_ErrorCountGet(void)
227 {
228     uint8_t ret;
229
230     ret = i2c2_object.i2cErrors;
231     return ret;
232 }
233
234 void I2C2_ISR ( void )
235 {
236     static uint8_t *pi2c_buf_ptr;
237     static uint16_t i2c_address = 0;

```

```

238     static uint8_t i2c_bytes_left      = 0;
239     static uint8_t i2c_10bit_address_restart = 0;
240
241     PIR2bits.SSP2IF = 0;
242
243     // Check first if there was a collision.
244     // If we have a Write Collision, reset and go to idle state */
245     if(I2C2_WRITE_COLLISION_STATUS_BIT)
246     {
247         // clear the Write colision
248         I2C2_WRITE_COLLISION_STATUS_BIT = 0;
249         i2c2_state = S_MASTER_IDLE;
250         *(p_i2c2_current->pTrFlag) = I2C2_MESSAGE_FAIL;
251
252         // reset the buffer pointer
253         p_i2c2_current = NULL;
254
255         return;
256     }
257
258     /* Handle the correct i2c state */
259     switch(i2c2_state)
260     {
261     case S_MASTER_IDLE:    /* In reset state, waiting for data to send */
262
263         if(i2c2_object.trStatus.s.empty != true)
264         {
265             // grab the item pointed by the head
266             p_i2c2_current = i2c2_object.pTrHead;
267             i2c2_trb_count = i2c2_object.pTrHead->count;
268             p_i2c2_trb_current = i2c2_object.pTrHead->ptrb_list;
269
270             i2c2_object.pTrHead++;
271
272             // check if the end of the array is reached
273             if(i2c2_object.pTrHead == (i2c2_tr_queue +
274                                         I2C2_CONFIG_TR_QUEUE_LENGTH))
275             {
276                 // adjust to restart at the beginning of the array
277                 i2c2_object.pTrHead = i2c2_tr_queue;
278             }
279
280             // since we moved one item to be processed, we know
281             // it is not full, so set the full status to false
282             i2c2_object.trStatus.s.full = false;
283
284             // check if the queue is empty
285             if(i2c2_object.pTrHead == i2c2_object.pTrTail)
286             {
287                 // it is empty so set the empty status to true
288                 i2c2_object.trStatus.s.empty = true;
289             }
290
291             // send the start condition
292             I2C2_START_CONDITION_ENABLE_BIT = 1;
293
294             // start the i2c request
295             i2c2_state = S_MASTER_SEND_ADDR;
296         }
297
298         break;
299
300     case S_MASTER_RESTART:
301         /* check for pending i2c Request */

```

```

302
303     // ... trigger a REPEATED START
304     I2C2_REPEAT_START_CONDITION_ENABLE_BIT = 1;
305
306     // start the i2c request
307     i2c2_state = S_MASTER_SEND_ADDR;
308
309     break;
310
311 case S_MASTER_SEND_ADDR_10BIT_LSB:
312
313     if(I2C2_ACKNOWLEDGE_STATUS_BIT)
314     {
315         i2c2_object.i2cErrors++;
316         I2C2_Stop(I2C2_MESSAGE_ADDRESS_NO_ACK);
317     }
318     else
319     {
320         // Remove bit 0 as R/W is never sent here
321         I2C2_TRANSMIT_REG = (i2c_address >> 1) & 0x00FF;
322
323         // determine the next state, check R/W
324         if(i2c_address & 0x01)
325         {
326             // if this is a read we must repeat start
327             // the bus to perform a read
328             i2c2_state = S_MASTER_10BIT_RESTART;
329         }
330         else
331         {
332             // this is a write continue writing data
333             i2c2_state = S_MASTER_SEND_DATA;
334         }
335     }
336
337     break;
338
339 case S_MASTER_10BIT_RESTART:
340
341     if(I2C2_ACKNOWLEDGE_STATUS_BIT)
342     {
343         i2c2_object.i2cErrors++;
344         I2C2_Stop(I2C2_MESSAGE_ADDRESS_NO_ACK);
345     }
346     else
347     {
348         // ACK Status is good
349         // restart the bus
350         I2C2_REPEAT_START_CONDITION_ENABLE_BIT = 1;
351
352         // fudge the address so S_MASTER_SEND_ADDR works correctly
353         // we only do this on a 10-bit address resend
354         i2c_address = 0x00F0 | ((i2c_address >> 8) & 0x0006);
355
356         // set the R/W flag
357         i2c_address |= 0x0001;
358
359         // set the address restart flag so we do not change the address
360         i2c_10bit_address_restart = 1;
361
362         // Resend the address as a read
363         i2c2_state = S_MASTER_SEND_ADDR;
364     }
365
366     break;

```

```

367
368     case S_MASTER_SEND_ADDR:
369
370         /* Start has been sent, send the address byte */
371
372         /* Note:
373             On a 10-bit address resend (done only during a 10-bit
374             device read), the original i2c_address was modified in
375             S_MASTER_10BIT_RESTART state. So the check if this is
376             a 10-bit address will fail and a normal 7-bit address
377             is sent with the R/W bit set to read. The flag
378             i2c_10bit_address_restart prevents the address to
379             be re-written.
380         */
381         if(i2c_10bit_address_restart != 1)
382         {
383             // extract the information for this message
384             i2c_address = p_i2c2_trb_current->address;
385             pi2c_buf_ptr = p_i2c2_trb_current->pbuffer;
386             i2c_bytes_left = p_i2c2_trb_current->length;
387         }
388
389         // check for 10-bit address
390         if(!I2C2_7bit && (0x0 != i2c_address))
391         {
392             if (0 == i2c_10bit_address_restart)
393             {
394                 // we have a 10 bit address
395                 // send bits<9:8>
396                 // mask bit 0 as this is always a write
397                 I2C2_TRANSMIT_REG = 0xF0 | ((i2c_address >> 8) & 0x0006);
398                 i2c2_state = S_MASTER_SEND_ADDR_10BIT_LSB;
399             }
380             else
381             {
382                 // resending address bits<9:8> to trigger read
383                 I2C2_TRANSMIT_REG = i2c_address;
384                 i2c2_state = S_MASTER_ACK_ADDR;
385                 // reset the flag so the next access is ok
386                 i2c_10bit_address_restart = 0;
387             }
388         }
389         else
390         {
391             // Transmit the address
392             I2C2_TRANSMIT_REG = i2c_address;
393             if(i2c_address & 0x01)
394             {
395                 // Next state is to wait for address to be acked
396                 i2c2_state = S_MASTER_ACK_ADDR;
397             }
398             else
399             {
400                 // Next state is transmit
401                 i2c2_state = S_MASTER_SEND_DATA;
402             }
403         }
404         break;
405
406     case S_MASTER_SEND_DATA:
407
408         // Make sure the previous byte was acknowledged
409         if(I2C2_ACKNOWLEDGE_STATUS_BIT)
410         {
411             // Transmission was not acknowledged

```

```

432     i2c2_object.i2cErrors++;
433
434     // Reset the Ack flag
435     I2C2_ACKNOWLEDGE_STATUS_BIT = 0;
436
437     // Send a stop flag and go back to idle
438     I2C2_Stop(I2C2_DATA_NO_ACK);
439 }
440
441 else
442 {
443     // Did we send them all ?
444     if(i2c_bytes_left-- == 0U)
445     {
446         // yup sent them all!
447
448         // update the trb pointer
449         p_i2c2_trb_current++;
450
451         // are we done with this string of requests?
452         if(--i2c2_trb_count == 0)
453         {
454             I2C2_Stop(I2C2_MESSAGE_COMPLETE);
455         }
456         else
457         {
458             // no!, there are more TRB to be sent.
459             //I2C2_START_CONDITION_ENABLE_BIT = 1;
460
461             // In some cases, the slave may require
462             // a restart instead of a start. So use this one
463             // instead.
464             I2C2_REPEAT_START_CONDITION_ENABLE_BIT = 1;
465
466             // start the i2c request
467             i2c2_state = S_MASTER_SEND_ADDR;
468
469         }
470     }
471     else
472     {
473         // Grab the next data to transmit
474         I2C2_TRANSMIT_REG = *pi2c_buf_ptr++;
475     }
476 }
477 break;
478
479 case S_MASTER_ACK_ADDR:
480
481     /* Make sure the previous byte was acknowledged */
482     if(I2C2_ACKNOWLEDGE_STATUS_BIT)
483     {
484
485         // Transmission was not acknowledged
486         i2c2_object.i2cErrors++;
487
488         // Send a stop flag and go back to idle
489         I2C2_Stop(I2C2_MESSAGE_ADDRESS_NO_ACK);
490
491         // Reset the Ack flag
492         I2C2_ACKNOWLEDGE_STATUS_BIT = 0;
493     }
494     else
495     {
496         I2C2_RECEIVE_ENABLE_BIT = 1;

```

```

1  /**
2   MSSP2 Generated Driver API Header File
3
4  @Company
5   Microchip Technology Inc.
6
7  @File Name
8   i2c2.h
9
10 @Summary
11   This is the generated header file for the I2C2 driver using MPLAB(c) Code
12  Configurator
13 @Description
14   This header file provides APIs for driver for I2C2.
15  Generation Information :
16      Product Revision : MPLAB(c) Code Configurator - 4.15.5
17      Device          : PIC16F18345
18      Driver Version  : 2.00
19      The generated drivers are tested against the following:
20      Compiler        : XC8 1.35
21      MPLAB          : MPLAB X 3.40
22 */
23
24 /*
25  (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
26  software and any derivatives exclusively with Microchip products.
27
28 THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
29 EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
30 WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
31 PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
32 WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
33
34 IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
35 INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
36 WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
37 BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
38 FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
39 ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
40 THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
41
42 MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
43 TERMS.
44 */
45
46 #ifndef _I2C2_H
47 #define _I2C2_H
48
49 /**
50  Section: Included Files
51 */
52 #include <stdint.h>
53 #include <stdbool.h>
54 #include <stddef.h>
55 #include <xc.h>
56
57 #ifdef __cplusplus // Provide C++ Compatibility
58
59     extern "C" {
60
61 #endif
62
63 /**
64  Section: Data Type Definitions

```

```

65     */
66
67 /**
68     I2C Driver Message Status Type Enumeration
69
70     @Summary
71         Defines the different message status when processing
72         TRBs.
73
74     @Description
75         This type enumeration specifies the different types of status
76         that an i2c request will have. For every submission of an i2c
77         transaction, the status of that transaction is available.
78         Based on the status, new transactions can be requested to the
79         driver or a recovery can be performed to resend the transaction.
80
81 */
82
83 typedef enum
84 {
85     I2C2_MESSAGE_COMPLETE,
86     I2C2_MESSAGE_FAIL,
87     I2C2_MESSAGE_PENDING,
88     I2C2_STUCK_START,
89     I2C2_MESSAGE_ADDRESS_NO_ACK,
90     I2C2_DATA_NO_ACK,
91     I2C2_LOST_STATE
92 } I2C2_MESSAGE_STATUS;
93
94 /**
95     I2C Driver Transaction Request Block (TRB) type definition.
96
97     @Summary
98         This defines the Transaction Request Block (TRB) used by the
99         i2c master in sending/receiving data to the i2c bus.
100
101    @Description
102        This data type is the i2c Transaction Request Block (TRB) that
103        the needs to be built and sent to the driver to handle each i2c requests.
104        Using the TRB, simple to complex i2c transactions can be constructed
105        and sent to the i2c bus. This data type is only used by the master mode.
106
107 */
108 typedef struct
109 {
110     uint16_t address;           // Bits <10:1> are the 10 bit address.
111                           // Bits <7:1> are the 7 bit address
112                           // Bit 0 is R/W (1 for read)
113     uint8_t length;            // the # of bytes in the buffer
114     uint8_t *pbuffer;          // a pointer to a buffer of length bytes
115 } I2C2_TRANSACTION_REQUEST_BLOCK;
116
117 /**
118     Section: Interface Routines
119 */
120 /**
121     @Summary
122         Initializes the MSSP instance : 2
123
124     @Description
125         This routine initializes the i2c2 driver instance for : 2
126         index, making it ready for clients to open and use it.
127         This routine must be called before any other I2C2 routine is called.
128         This routine should only be called once during system initialization.
129

```

```

130     @Preconditions
131         None.
132
133     @Param
134         None.
135
136     @Returns
137         None.
138
139     @Example
140         <code>
141             #define SLAVE_I2C_GENERIC_RETRY_MAX      100
142
143             // initialize the module
144             I2C2_Initialize();
145
146             // write to an EEPROM Device
147
148             uint16_t      dataAddress;
149             uint8_t       sourceData[16] = { 0xA0, 0xA1, 0xA2, 0xA3,
150                                         0xA4, 0xA5, 0xA6, 0xA7,
151                                         0xA8, 0xA9, 0xAA, 0xAB,
152                                         0xAC, 0xAD, 0xAE, 0xAF };
153             uint8_t       *pData;
154             uint16_t      nCount;
155
156             uint8_t       writeBuffer[3];
157             uint8_t       *pD;
158             uint16_t      counter, timeOut;
159
160             I2C2_MESSAGE_STATUS status = I2C2_MESSAGE_PENDING;
161
162             dataAddress = 0x10;           // starting EEPROM address
163             pD = sourceData;           // initialize the source of the data
164             nCount = 16;                // number of bytes to write
165
166             for (counter = 0; counter < nCount; counter++)
167             {
168
169                 // build the write buffer first
170                 // starting address of the EEPROM memory
171                 writeBuffer[0] = (dataAddress >> 8);           // high address
172                 writeBuffer[1] = (uint8_t)(dataAddress);          // low low address
173
174                 // data to be written
175                 writeBuffer[2] = *pD++;
176
177                 // Now it is possible that the slave device will be slow.
178                 // As a work around on these slaves, the application can
179                 // retry sending the transaction
180                 timeOut = 0;
181                 while(status != I2C2_MESSAGE_FAIL)
182                 {
183                     // write one byte to EEPROM (3 is the number of bytes to write)
184                     I2C2_MasterWrite( writeBuffer,
185                                     3,
186                                     slaveDeviceAddress,
187                                     &status);
188
189                     // wait for the message to be sent or status has changed.
190                     while(status == I2C2_MESSAGE_PENDING);
191
192                     if (status == I2C2_MESSAGE_COMPLETE)
193                         break;
194

```

```

195     // if status is I2C2_MESSAGE_ADDRESS_NO_ACK,
196     //         or I2C2_DATA_NO_ACK,
197     // The device may be busy and needs more time for the last
198     // write so we can retry writing the data, this is why we
199     // use a while loop here
200
201     // check for max retry and skip this byte
202     if (timeOut == SLAVE_I2C_GENERIC_RETRY_MAX)
203         break;
204     else
205         timeOut++;
206 }
207
208     if (status == I2C2_MESSAGE_FAIL)
209     {
210         break;
211     }
212     dataAddress++;
213
214 }
215
216 </code>
217 */
218
219 void I2C2_Initialize(void);
220
221 /**
222 * @Summary
223 * Handles one i2c master write transaction with the
224 * supplied parameters.
225
226 * @Description
227 * This function prepares a TRB, then inserts it on the i2c queue.
228 * Finally, it waits for the transaction to complete and returns
229 * the result.
230
231 * @Preconditions
232 * None
233
234 * @Param
235 * address - The address of the i2c peripheral to be accessed
236
237 * @Param
238 * length - The length of the data block to be sent
239
240 * @Param
241 * *pdata - A pointer to the block of data to be sent
242
243 * @Param
244 * *pstatus - A pointer to the status variable that the i2c driver
245 *             updates during the execution of the message.
246
247 * @Returns
248 * I2C2_MESSAGE_STATUS
249
250 * @Example
251 * <code>
252 *     Refer to I2C2_Initialize() and
253 *     I2C2_MasterRead() for an examples
254 * </code>
255
256 */
257 void I2C2_MasterWrite(

```

```

260                               uint8_t *pdata,
261                               uint8_t length,
262                               uint16_t address,
263                               I2C2_MESSAGE_STATUS *pstatus);
264
265 /**
266  * @Summary
267  * Handles one i2c master read transaction with the
268  * supplied parameters.
269
270  * @Description
271  * This function prepares a TRB, then inserts it on the i2c queue.
272  * Finally, it waits for the transaction to complete and returns
273  * the result.
274
275  * @Preconditions
276  * None
277
278  * @Param
279  * address - The address of the i2c peripheral to be accessed
280
281  * @Param
282  * length - The length of the data block to be sent
283
284  * @Param
285  * *pdata - A pointer to the memory location where received data will
286  * be stored
287
288  * @Param
289  * *pstatus - A pointer to the status variable that the i2c driver
290  * updates during the execution of the message.
291
292  * @Returns
293  * I2C2_MESSAGE_STATUS
294
295  * @Example
296  * <code>
297
298      #define MCHP24AA512_RETRY_MAX 100 // define the retry count
299      #define MCHP24AA512_ADDRESS 0x50 // slave device address
300
301      uint8_t MCHP24AA512_Read(
302                                  uint16_t address,
303                                  uint8_t *pData,
304                                  uint16_t nCount)
305  {
306      I2C2_MESSAGE_STATUS status;
307      uint8_t writeBuffer[3];
308      uint16_t timeOut;
309      uint16_t counter;
310      uint8_t *pD, ret;
311
312      pD = pData;
313
314      for (counter = 0; counter < nCount; counter++)
315  {
316
317          // build the write buffer first
318          // starting address of the EEPROM memory
319          writeBuffer[0] = (address >> 8); // high
320          address
321          writeBuffer[1] = (uint8_t)(address); // low low
322          address
323
324          // Now it is possible that the slave device will be slow.

```

```

323 // As a work around on these slaves, the application can
324 // retry sending the transaction
325 timeOut = 0;
326 while(status != I2C2_MESSAGE_FAIL)
327 {
328     // write one byte to EEPROM (2 is the count of bytes to
329     // write)
330     I2C2_MasterWrite(    writeBuffer,
331                         2,
332                         MCHP24AA512_ADDRESS,
333                         &status);
334
335     // wait for the message to be sent or status has changed.
336     while(status == I2C2_MESSAGE_PENDING);
337
338     if (status == I2C2_MESSAGE_COMPLETE)
339         break;
340
341     // if status is I2C2_MESSAGE_ADDRESS_NO_ACK,
342     // or I2C2_DATA_NO_ACK,
343     // The device may be busy and needs more time for the last
344     // write so we can retry writing the data, this is why we
345     // use a while loop here
346
347     // check for max retry and skip this byte
348     if (timeOut == MCHP24AA512_RETRY_MAX)
349         break;
350     else
351         timeOut++;
352 }
353
354 if (status == I2C2_MESSAGE_COMPLETE)
355 {
356
357     // this portion will read the byte from the memory
358     // location.
359     timeOut = 0;
360     while(status != I2C2_MESSAGE_FAIL)
361     {
362         // write one byte to EEPROM (2 is the count of bytes
363         // to write)
364         I2C2_MasterRead(    pD,
365                           1,
366                           MCHP24AA512_ADDRESS,
367                           &status);
368
369         // wait for the message to be sent or status has
370         // changed.
371         while(status == I2C2_MESSAGE_PENDING);
372
373         if (status == I2C2_MESSAGE_COMPLETE)
374             break;
375
376         // if status is I2C2_MESSAGE_ADDRESS_NO_ACK,
377         // or I2C2_DATA_NO_ACK,
378         // The device may be busy and needs more time for
379         // the last
380         // write so we can retry writing the data, this is
381         // why we
382         // use a while loop here
383
384         // check for max retry and skip this byte
385         if (timeOut == MCHP24AA512_RETRY_MAX)
386             break;
387         else

```

```

382                     timeOut++;
383                 }
384             }
385         }
386         // exit if the last transaction failed
387         if (status == I2C2_MESSAGE_FAIL)
388         {
389             ret = 0;
390             break;
391         }
392         pD++;
393         address++;
394     }
395     return (ret);
396 }
397
398 */
399
400
401 </code>
402 */
403
404 /**
405 void I2C2_MasterRead(
406     uint8_t *pdata,
407     uint8_t length,
408     uint16_t address,
409     I2C2_MESSAGE_STATUS *pstatus);
410
411 /**
412 @Summary
413     Inserts a list of i2c transaction requests into the i2c
414     transaction queue.
415
416 @Description
417     The i2c processes lists of transaction requests. Each transaction
418     list is handled as a string of i2c restarts. When the list of
419     transactions is complete, an i2c stop is produced, the flag is set
420     with the correct condition code and the next list is processed
421     from the queue.
422
423     This function inserts lists of requests prepared by the user
424     application into the queue along with a pointer to the completion
425     flag.
426
427     The transaction is inserted into the list only if there is space
428     in the list. If there is no space, the function exits with the
429     flag set to I2C2_MESSAGE_FAIL.
430
431 @Preconditions
432     None
433
434 @Param
435     count - The numer of transaction requests in the trb_list.
436
437 @Param
438     *ptrb_list - A pointer to an array of transaction requests (TRB).
439     See I2C2_TRANSACTION_REQUEST_BLOCK definition for details.
440
441 @Param
442     *pflag - A pointer to a completion flag.
443
444 @Returns
445     None
446

```

```

447
448     @Example
449         <code>
450
451     void EMULATED_EEPROM_Read(
452                                     uint16_t slaveDeviceAddress,
453                                     uint16_t dataAddress,
454                                     uint8_t *pData,
455                                     uint16_t nCount)
456     {
457
458         I2C2_MESSAGE_STATUS status;
459         I2C2_TRANSACTION_REQUEST_BLOCK readTRB[2];
460         uint8_t      writeBuffer[3];
461         uint16_t     timeOut;
462
463         // this initial value is important
464         status = I2C2_MESSAGE_PENDING;
465
466         // build the write buffer first
467         // starting address of the EEPROM memory
468         writeBuffer[0] = (dataAddress >> 8);           // high address
469         writeBuffer[1] = (uint8_t)(dataAddress);          // low address
470
471         // we need to create the TRBs for a random read sequence to the
472         // EEPROM
473         // Build TRB for sending address
474         I2C2_MasterWriteTRBBuild(   &readTRB[0],
475                                   writeBuffer,
476                                   2,
477                                   slaveDeviceAddress);
478         // Build TRB for receiving data
479         I2C2_MasterReadTRBBuild(   &readTRB[1],
480                               pData,
481                               nCount,
482                               slaveDeviceAddress);
483
484         timeOut = 0;
485
486         while(status != I2C2_MESSAGE_FAIL)
487         {
488             // now send the transactions
489             I2C2_MasterTRBInsert(2, readTRB, &status);
490
491             // wait for the message to be sent or status has changed.
492             while(status == I2C2_MESSAGE_PENDING);
493
494             if (status == I2C2_MESSAGE_COMPLETE)
495                 break;
496
497             // if status is I2C2_MESSAGE_ADDRESS_NO_ACK,
498             // or I2C2_DATA_NO_ACK,
499             // The device may be busy and needs more time for the last
500             // write so we can retry writing the data, this is why we
501             // use a while loop here
502
503             // check for max retry and skip this byte
504             if (timeOut == SLAVE_I2C_GENERIC_RETRY_MAX)
505                 break;
506             else
507                 timeOut++;
508         }

```

```

509
510         }
511     
```

`</code>`

```

512 /**
513 */
514 void I2C2_MasterTRBInsert(
515     uint8_t count,
516     I2C2_TRANSACTION_REQUEST_BLOCK *ptrb_list,
517     I2C2_MESSAGE_STATUS *pflag);
518
519 /**
520 ** @Summary
521     This function populates a trb supplied by the calling function
522     with the parameters supplied by the calling function.
523
524     @Description
525     All i2c requests are in the form of TRB's. This helper function
526     takes standard parameters and correctly formats the TRB. The R/W
527     bit is set to ensure that the resulting TRB describes an i2c read
528     operation.
529
530     This function does not send the transaction. To send the transaction,
531     the TRB insert function (I2C2_MasterTRBInsert()) must be called.
532
533     @Preconditions
534     None
535
536     @Param
537         *ptrb - A pointer to a caller supplied TRB.
538
539     @Param
540         *pdata - A pointer to the block of data to be received
541
542     @Param
543         length - The length of the data block to be received
544
545     @Param
546         address - The address of the i2c peripheral to be accessed
547
548     @Returns
549         None
550
551     @Example
552         <code>
553             Refer to I2C2_MasterTRBInsert() for an example
554         </code>
555
556 /**
557 */
558 void I2C2_MasterReadTRBBuild(
559     I2C2_TRANSACTION_REQUEST_BLOCK *ptrb,
560     uint8_t *pdata,
561     uint8_t length,
562     uint16_t address);
563
564 /**
565 ** @Summary
566     This function populates a trb supplied by the calling function
567     with the parameters supplied by the calling function.
568
569     @Description
570     All i2c requests are in the form of TRB's. This helper function
571     takes standard parameters and correctly formats the TRB. The R/W

```

```

574     bit is cleared to ensure that the resulting TRB describes an i2c
575     write operation.
576
577     This function does not send the transaction. To send the transaction,
578     the TRB insert function (I2C2_MasterTRBInsert()) must be called.
579
580     @Preconditions
581         None
582
583     @Param
584         *ptrb - A pointer to a caller supplied TRB.
585
586     @Param
587         *pdata - A pointer to the block of data to be sent
588
589     @Param
590         length - The length of the data block to be sent
591
592     @Param
593         address - The address of the i2c peripheral to be accessed
594
595     @Returns
596         None
597
598     @Example
599         <code>
600             Refer to I2C2_MasterTRBInsert() for an example
601         </code>
602
603     */
604
605     void I2C2_MasterWriteTRBBuild(
606             I2C2_TRANSACTION_REQUEST_BLOCK *ptrb,
607             uint8_t *pdata,
608             uint8_t length,
609             uint16_t address);
610
611     /**
612     @Summary
613         This function returns the empty status of the Master
614         queue.
615
616     @Description
617         This function returns the empty status of the Master
618         queue. Use this function to check if the queue is empty.
619         This can verify if the Master is currently idle.
620
621     @Preconditions
622         None
623
624     @Param
625         None
626
627     @Returns
628         True if the queue is empty and false if the queue is not empty.
629
630     @Example
631         <code>
632             #define MCHP24AA512_ADDRESS      0x50 // slave device address
633
634             // check until queue is empty
635             while(I2C2_MasterQueueIsEmpty() == false);
636
637             // now send more data (assume readBuffer is initialized)
638             I2C2_MasterRead(    readBuffer,

```

```

639                               3,
640                               MCHP24AA512_ADDRESS,
641                               &status);
642
643             </code>
644
645         */
646
647     bool I2C2_MasterQueueIsEmpty(void);
648
649 /**
650     @Summary
651         This function returns the full status of the Master
652         queue.
653
654     @Description
655         This function returns the full status of the Master
656         queue. Use this function to check if the queue is full.
657         This can verify if the Master will not be able to accept
658         addition transactions.
659
660     @Preconditions
661         None
662
663     @Param
664         None
665
666     @Returns
667         True if the queue is full and false if the queue is not full.
668
669     @Example
670         <code>
671             #define MCHP24AA512_ADDRESS      0x50 // slave device address
672
673             // check until queue has space
674             while(I2C2_MasterQueueIsFull() == true);
675
676             // now send more data (assume readBuffer is initialized)
677             I2C2_MasterRead(    readBuffer,
678                               3,
679                               MCHP24AA512_ADDRESS,
680                               &status);
681
682         </code>
683
684     */
685
686     bool I2C2_MasterQueueIsFull(void);
687     void I2C2_BusCollisionISR( void );
688     void I2C2_ISR ( void );
689
690 #ifdef __cplusplus // Provide C++ Compatibility
691 }
692
693 #endif
694
695 #endif // _I2C2_H
696
697 /**
698     End of File
699 */
700

```

```

1  /**
2   * Generated Interrupt Manager Source File
3
4   * @Company:
5   *   Microchip Technology Inc.
6
7   * @File Name:
8   *   interrupt_manager.c
9
10  * @Summary:
11  *   This is the Interrupt Manager file generated using MPLAB(c) Code Configurator
12
13  * @Description:
14  *   This header file provides implementations for global interrupt handling.
15  *   For individual peripheral handlers please see the peripheral driver for
16  *   all modules selected in the GUI.
17  * Generation Information :
18  *   Product Revision : MPLAB(c) Code Configurator - 4.15.5
19  *   Device          : PIC16F18345
20  *   Driver Version  : 1.02
21  * The generated drivers are tested against the following:
22  *   Compiler        : XC8 1.35
23  *   MPLAB          : MPLAB X 3.40
24 */
25
26 /*
27  * (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
28  * software and any derivatives exclusively with Microchip products.
29
30  * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
31  * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
32  * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
33  * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
34  * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
35
36  * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
37  * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
38  * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
39  * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
40  * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
41  * ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
42  * THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
43
44  * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
45  * TERMS.
46 */
47
48 #include "interrupt_manager.h"
49 #include "mcc.h"
50
51 void interrupt INTERRUPT_InterruptManager (void)
52 {
53     // interrupt handler
54     if(INTCONbits.PEIE == 1 && PIE1bits.RCIE == 1 && PIR1bits.RCIF == 1)
55     {
56         EUSART_Receive_ISR();
57     }
58     else if(INTCONbits.PEIE == 1 && PIE1bits.TXIE == 1 && PIR1bits.TXIF == 1)
59     {
60         EUSART_Transmit_ISR();
61     }
62     else if(INTCONbits.PEIE == 1 && PIE2bits.BCL2IE == 1 && PIR2bits.BCL2IF == 1)
63     {
64         I2C2_BusCollisionISR();
65     }
}

```

```
66     else if(INTCONbits.PEIE == 1 && PIE2bits.SSP2IE == 1 && PIR2bits.SSP2IF == 1)
67     {
68         I2C2_ISR();
69     }
70     else if(PIE0bits.INTE == 1 && PIR0bits.INTF == 1)
71     {
72         INT_ISR();
73     }
74     else
75     {
76         //Unhandled Interrupt
77     }
78 }
79 /**
80 End of File
81 */
```

```

1  /**
2   * Generated Interrupt Manager Header File
3
4   * @Company:
5   *   Microchip Technology Inc.
6
7   * @File Name:
8   *   interrupt_manager.h
9
10  * @Summary:
11  *   This is the Interrupt Manager file generated using MPLAB(c) Code Configurator
12
13  * @Description:
14  *   This header file provides implementations for global interrupt handling.
15  *   For individual peripheral handlers please see the peripheral driver for
16  *   all modules selected in the GUI.
17  * Generation Information :
18  *   Product Revision : MPLAB(c) Code Configurator - 4.15.5
19  *   Device          : PIC16F18345
20  *   Driver Version  : 2.00
21  * The generated drivers are tested against the following:
22  *   Compiler        : XC8 1.35
23  *   MPLAB          : MPLAB X 3.40
24 */
25
26 /*
27  * (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
28  * software and any derivatives exclusively with Microchip products.
29
30  * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
31  * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
32  * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
33  * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
34  * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
35
36  * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
37  * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
38  * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
39  * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
40  * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
41  * ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
42  * THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
43
44  * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
45  * TERMS.
46 */
47
48 #ifndef INTERRUPT_MANAGER_H
49 #define INTERRUPT_MANAGER_H
50
51
52 /**
53  * @Param
54  *   none
55  * @Returns
56  *   none
57  * @Description
58  *   This macro will enable global interrupts.
59  * @Example
60  *   INTERRUPT_GlobalInterruptEnable();
61  */
62 #define INTERRUPT_GlobalInterruptEnable() (INTCONbits.GIE = 1)
63
64 /**
65  * @Param

```

```

66     none
67     * @Returns
68     none
69     * @Description
70     This macro will disable global interrupts.
71     * @Example
72     INTERRUPT_GlobalInterruptDisable();
73     */
74 #define INTERRUPT_GlobalInterruptDisable() (INTCONbits.GIE = 0)
75
76 /**
77  * @Param
78  none
79  * @Returns
80  none
81  * @Description
82  This macro will enable peripheral interrupts.
83  * @Example
84  INTERRUPT_PeripheralInterruptEnable();
85  */
86 #define INTERRUPT_PeripheralInterruptEnable() (INTCONbits.PEIE = 1)
87
88 /**
89  * @Param
90  none
91  * @Returns
92  none
93  * @Description
94  This macro will disable peripheral interrupts.
95  * @Example
96  INTERRUPT_PeripheralInterruptDisable();
97  */
98 #define INTERRUPT_PeripheralInterruptDisable() (INTCONbits.PEIE = 0)
99
100 /**
101  * @Param
102  none
103  * @Returns
104  none
105  * @Description
106  Main interrupt service routine. Calls module interrupt handlers.
107  * @Example
108  INTERRUPT_InterruptManager();
109  */
110 void interrupt INTERRUPT_InterruptManager(void);
111
112
113 #endif // INTERRUPT_MANAGER_H
114 /**
115  End of File
116 */

```

```

1  /**
2   * @Generated MPLAB(c) Code Configurator Source File
3
4   * @Company:
5   *   Microchip Technology Inc.
6
7   * @File Name:
8   *   mcc.c
9
10  * @Summary:
11  *   This is the mcc.c file generated using MPLAB(c) Code Configurator
12
13  * @Description:
14  *   This header file provides implementations for driver APIs for all modules
15  *   selected in the GUI.
16  *   Generation Information :
17  *     Product Revision : MPLAB(c) Code Configurator - 4.15.5
18  *     Device          : PIC16F18345
19  *     Driver Version  : 1.02
20  *   The generated drivers are tested against the following:
21  *     Compiler        : XC8 1.35
22  *     MPLAB          : MPLAB X 3.40
23  */
24
25  /*
26  *   (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
27  *   software and any derivatives exclusively with Microchip products.
28
29  * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
30  * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
31  * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
32  * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
33  * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
34
35  * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
36  * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
37  * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
38  * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
39  * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
40  * ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
41  * THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
42
43  * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
44  * TERMS.
45
46  // Configuration bits: selected in the GUI
47
48  // CONFIG1
49  #pragma config FEXTOSC = HS      // FEXTOSC External Oscillator mode Selection
50  // bits->HS (crystal oscillator) above 4 MHz
51  #pragma config RSTOSC = EXTIX    // Power-up default value for COSC bits->EXTOSC
52  // operating per FEXTOSC bits
53  #pragma config CLKOUTEN = OFF    // Clock Out Enable bit->CLKOUT function is
54  // disabled; I/O or oscillator function on OSC2
55  #pragma config CSWEN = ON       // Clock Switch Enable bit->Writing to NOSC and
56  // NDIV is allowed
57  #pragma config FCMEN = ON       // Fail-Safe Clock Monitor Enable->Fail-Safe Clock
58  // Monitor is enabled
59
60  // CONFIG2
61  #pragma config MCLRE = ON       // Master Clear Enable bit->MCLR/VPP pin function
62  // is MCLR; Weak pull-up enabled
63  #pragma config PWRTE = OFF      // Power-up Timer Enable bit->PWRT disabled
64  #pragma config WDTE = OFF       // Watchdog Timer Enable bits->WDT disabled; SWDTEN

```

```

is ignored
59 #pragma config LPBOREN = OFF      // Low-power BOR enable bit->ULPBOR disabled
60 #pragma config BOREN = ON        // Brown-out Reset Enable bits->Brown-out Reset
61 enabled, SBORN bit ignored
61 #pragma config BORV = LOW       // Brown-out Reset Voltage selection bit->Brown-out
62 voltage (Vbor) set to 2.45V
62 #pragma config PPS1WAY = ON     // PPSLOCK bit One-Way Set Enable bit->The
63 PPSLOCK bit can be cleared and set only once; PPS registers remain locked after
one clear/set cycle
63 #pragma config STVREN = ON      // Stack Overflow/Underflow Reset Enable
bit->Stack Overflow or Underflow will cause a Reset
64 #pragma config DEBUG = OFF      // Debugger enable bit->Background debugger disabled
65
66 // CONFIG3
67 #pragma config WRT = OFF        // User NVM self-write protection bits->Write
protection off
68 #pragma config LVP = ON         // Low Voltage Programming Enable bit->Low Voltage
programming enabled. MCLR/VPP pin function is MCLR. MCLRE configuration bit is
ignored.
69
70 // CONFIG4
71 #pragma config CP = OFF        // User NVM Program Memory Code Protection bit->User
NVM code protection disabled
72 #pragma config CPD = OFF        // Data NVM Memory Code Protection bit->Data NVM
code protection disabled
73
74 #include "mcc.h"
75
76 void SYSTEM_Initialize(void)
77 {
78
79     PIN_MANAGER_Initialize();
80     OSCILLATOR_Initialize();
81     WDT_Initialize();
82     I2C2_Initialize();
83     TMR2_Initialize();
84     TMRI_Initialize();
85     EXT_INT_Initialize();
86     TMRO_Initialize();
87     EUSART_Initialize();
88 }
89
90 void OSCILLATOR_Initialize(void)
91 {
92     // NOSC EXTOSC; NDIV 1;
93     OSCCON1 = 0x70;
94     // CSWOLD may proceed; SOSCPWR Low power; SOSCBE crystal oscillator;
95     OSCCON3 = 0x00;
96     // LFOEN disabled; ADOEN disabled; SOSCEN disabled; EXTOEN disabled; HFOEN
disabled;
97     OSCEN = 0x00;
98     // HFFRQ 32_MHz;
99     OSCFRQ = 0x07;
100    // HFTUN 0;
101    OSCTUNE = 0x00;
102 }
103
104 void WDT_Initialize(void)
105 {
106     // WDTPS 1:65536; SWDTEN OFF;
107     WDTCON = 0x16;
108 }
109
110 /**
111 End of File

```

112 * /

```

1  /**
2   * @Generated MPLAB(c) Code Configurator Header File
3
4   * @Company:
5   *   Microchip Technology Inc.
6
7   * @File Name:
8   *   mcc.h
9
10  * @Summary:
11  *   This is the mcc.h file generated using MPLAB(c) Code Configurator
12
13  * @Description:
14  *   This header file provides implementations for driver APIs for all modules
15  *   selected in the GUI.
16  *   Generation Information :
17  *     Product Revision : MPLAB(c) Code Configurator - 4.15.5
18  *     Device          : PIC16F18345
19  *     Version         : 1.02
20  *   The generated drivers are tested against the following:
21  *     Compiler       : XC8 1.35
22  *     MPLAB          : MPLAB X 3.40
23  */
24
25  /*
26  *   (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
27  *   software and any derivatives exclusively with Microchip products.
28
29  * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
30  * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
31  * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
32  * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
33  * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
34
35  * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
36  * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
37  * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
38  * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
39  * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
40  * ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
41  * THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
42
43  * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
44  * TERMS.
45  */
46
47 #ifndef MCC_H
48 #define MCC_H
49 #include <xc.h>
50 #include "pin_manager.h"
51 #include <stdint.h>
52 #include <stdbool.h>
53 #include "interrupt_manager.h"
54 #include "tmr0.h"
55 #include "tmr1.h"
56 #include "tmr3.h"
57 #include "tmr2.h"
58 #include "eusart.h"
59 #include "i2c2.h"
60 #include "ext_int.h"
61 #define _XTAL_FREQ 32000000
62
63
64 /**

```

```

65  * @Param
66  none
67  * @Returns
68  none
69  * @Description
70   Initializes the device to the default states configured in the
71   *          MCC GUI
72  * @Example
73   SYSTEM_Initialize(void);
74  */
75 void SYSTEM_Initialize(void);

76 /**
77  * @Param
78  none
79  * @Returns
80  none
81  * @Description
82   Initializes the oscillator to the default states configured in the
83   *          MCC GUI
84  * @Example
85   OSCILLATOR_Initialize(void);
86  */
87 void OSCILLATOR_Initialize(void);

88 /**
89  * @Param
90  none
91  * @Returns
92  none
93  * @Description
94   Initializes the WDT module to the default states configured in the
95   *          MCC GUI
96  * @Example
97   WDT_Initialize(void);
98  */
99 void WDT_Initialize(void);

100 #endif /* MCC_H */
101 /**
102  End of File
103 */
104 */

```

```

1  /**
2   * Generated Pin Manager File
3
4   * Company:
5   *   Microchip Technology Inc.
6
7   * File Name:
8   *   pin_manager.c
9
10  * Summary:
11  *   This is the Pin Manager file generated using MPLAB(c) Code Configurator
12
13  * Description:
14  *   This header file provides implementations for pin APIs for all pins selected
15  *   in the GUI.
16  *   Generation Information :
17  *     Product Revision : MPLAB(c) Code Configurator - 4.15.5
18  *     Device          : PIC16F18345
19  *     Driver Version  : 1.02
20  *   The generated drivers are tested against the following:
21  *     Compiler       : XC8 1.35
22  *     MPLAB          : MPLAB X 3.40
23
24  Copyright (c) 2013 - 2015 released Microchip Technology Inc. All rights
25  reserved.
26
27  Microchip licenses to you the right to use, modify, copy and distribute
28  Software only when embedded on a Microchip microcontroller or digital signal
29  controller that is integrated into your product or third party product
30  (pursuant to the sublicense terms in the accompanying license agreement).
31
32  You should refer to the license agreement accompanying this Software for
33  additional information regarding your rights and obligations.
34
35  SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
36  EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
37  MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
38  IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
39  CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
40  OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
41  INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
42  CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
43  SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
44  (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
45
46 */
47
48 #include <xc.h>
49 #include "pin_manager.h"
50 #include "stdbool.h"
51
52 void PIN_MANAGER_Initialize(void)
53 {
54     /**
55      LATx registers
56      */
57     LATA = 0x00;
58     LATB = 0x00;
59     LATC = 0x00;
60
61     /**
62      TRIStx registers
63      */
64     TRISA = 0x37;

```

```

64      TRISB = 0xA0;
65      TRISC = 0x5E;
66
67      /**
68      ANSELx registers
69      */
70      ANSELC = 0x00;
71      ANSELB = 0x00;
72      ANSELA = 0x33;
73
74      /**
75      WPUx registers
76      */
77      WPUB = 0x00;
78      WPUA = 0x00;
79      WPUC = 0x00;
80
81      /**
82      ODX registers
83      */
84      ODCNA = 0x00;
85      ODCNB = 0x00;
86      ODCNC = 0x00;
87
88
89
90
91
92
93     bool state = GIE;
94     GIE = 0;
95     PPSLOCK = 0x55;
96     PPSLOCK = 0xAA;
97     PPSLOCKbits.PPSLOCKED = 0x00; // unlock PPS
98
99     RXPPSbits.RXPPS = 0x14;    //RC4->EUSART:RX;
100    RB7PPSbits.RB7PPS = 0x1A;   //RB7->MSSP2:SCL2;
101    INTPPSbits.INTPPS = 0x02;   //RA2->EXT_INT:INT;
102    SSP2DATPPSbits.SSP2DATPPS = 0x0D;  //RB5->MSSP2:SDA2;
103    SSP2CLKPPSbits.SSP2CLKPPS = 0x0F;  //RB7->MSSP2:SCL2;
104    RB5PPSbits.RB5PPS = 0x1B;   //RB5->MSSP2:SDA2;
105    RC5PPSbits.RC5PPS = 0x14;   //RC5->EUSART:TX;
106
107    PPSLOCK = 0x55;
108    PPSLOCK = 0xAA;
109    PPSLOCKbits.PPSLOCKED = 0x01; // lock PPS
110
111    GIE = state;
112 }
113
114 void PIN_MANAGER_IOC(void)
115 {
116
117 }
118
119 /**
120  End of File
121 */

```

```

1  /**
2   * @Generated Pin Manager Header File
3
4   * @Company:
5   *     Microchip Technology Inc.
6
7   * @File Name:
8   *     pin_manager.h
9
10  * @Summary:
11  *     This is the Pin Manager file generated using MPLAB(c) Code Configurator
12
13  * @Description:
14  *     This header file provides implementations for pin APIs for all pins selected
15  *     in the GUI.
16  *     Generation Information :
17  *         Product Revision : MPLAB(c) Code Configurator - 4.15.5
18  *         Device          : PIC16F18345
19  *         Version         : 1.01
20  *     The generated drivers are tested against the following:
21  *         Compiler       : XC8 1.35
22  *         MPLAB          : MPLAB X 3.40
23
24  Copyright (c) 2013 - 2015 released Microchip Technology Inc. All rights
25  reserved.
26
27  Microchip licenses to you the right to use, modify, copy and distribute
28  Software only when embedded on a Microchip microcontroller or digital signal
29  controller that is integrated into your product or third party product
30  (pursuant to the sublicense terms in the accompanying license agreement).
31
32  You should refer to the license agreement accompanying this Software for
33  additional information regarding your rights and obligations.
34
35  SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
36  EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
37  MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
38  IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
39  CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
40  OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
41  INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
42  CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
43  SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
44  (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
45
46 */
47 #ifndef PIN_MANAGER_H
48 #define PIN_MANAGER_H
49
50 #define INPUT    1
51 #define OUTPUT   0
52
53 #define HIGH     1
54 #define LOW      0
55
56 #define ANALOG   1
57 #define DIGITAL  0
58
59 #define PULL_UP_ENABLED    1
60 #define PULL_UP_DISABLED   0
61
62 // get/set RA2 procedures
63 #define RA2_SetHigh()      do { LATAbits.LATA2 = 1; } while(0)

```

```

64  #define RA2_SetLow()    do { LATAbits.LATA2 = 0; } while(0)
65  #define RA2_Toggle()   do { LATAbits.LATA2 = ~LATAbits.LATA2; } while(0)
66  #define RA2_GetValue()      PORTAbits.RA2
67  #define RA2_SetDigitalInput() do { TRISAbits.TRISA2 = 1; } while(0)
68  #define RA2_SetDigitalOutput() do { TRISAbits.TRISA2 = 0; } while(0)
69  #define RA2_SetPullup()     do { WPUAbits.WPUA2 = 1; } while(0)
70  #define RA2_ResetPullup()   do { WPUAbits.WPUA2 = 0; } while(0)
71  #define RA2_SetAnalogMode() do { ANSELAbits.ANSA2 = 1; } while(0)
72  #define RA2_SetDigitalMode() do { ANSELAbits.ANSA2 = 0; } while(0)
73
74 // get/set IO_RB4 aliases
75 #define IO_RB4_TRIS          TRISBbits.TRISB4
76 #define IO_RB4_LAT           LATBbits.LATB4
77 #define IO_RB4_PORT          PORTBbits.RB4
78 #define IO_RB4_WPU           WPUBBbits.WPUB4
79 #define IO_RB4_OD            ODCONBbits.ODCB4
80 #define IO_RB4_ANS           ANSELBbits.ANSB4
81 #define IO_RB4_SetHigh()     do { LATBbits.LATB4 = 1; } while(0)
82 #define IO_RB4_SetLow()      do { LATBbits.LATB4 = 0; } while(0)
83 #define IO_RB4_Toggle()      do { LATBbits.LATB4 = ~LATBbits.LATB4; }
84 while(0)
85 #define IO_RB4_GetValue()    PORTBbits.RB4
86 #define IO_RB4_SetDigitalInput() do { TRISBbits.TRISB4 = 1; } while(0)
87 #define IO_RB4_SetDigitalOutput() do { TRISBbits.TRISB4 = 0; } while(0)
88 #define IO_RB4_SetPullup()   do { WPUBBbits.WPUB4 = 1; } while(0)
89 #define IO_RB4_ResetPullup() do { WPUBBbits.WPUB4 = 0; } while(0)
90 #define IO_RB4_SetPushPull() do { ODCONBbits.ODCB4 = 1; } while(0)
91 #define IO_RB4_SetOpenDrain() do { ODCONBbits.ODCB4 = 0; } while(0)
92 #define IO_RB4_SetAnalogMode() do { ANSELBbits.ANSB4 = 1; } while(0)
93 #define IO_RB4_SetDigitalMode() do { ANSELBbits.ANSB4 = 0; } while(0)
94
95 // get/set SDA2 aliases
96 #define SDA2_TRIS          TRISBbits.TRISB5
97 #define SDA2_LAT           LATBbits.LATB5
98 #define SDA2_PORT          PORTBbits.RB5
99 #define SDA2_WPU           WPUBBbits.WPUB5
100 #define SDA2_OD            ODCONBbits.ODCB5
101 #define SDA2_ANS           ANSELBbits.ANSB5
102 #define SDA2_SetHigh()     do { LATBbits.LATB5 = 1; } while(0)
103 #define SDA2_SetLow()      do { LATBbits.LATB5 = 0; } while(0)
104 #define SDA2_Toggle()      do { LATBbits.LATB5 = ~LATBbits.LATB5; }
105 while(0)
106 #define SDA2_GetValue()    PORTBbits.RB5
107 #define SDA2_SetDigitalInput() do { TRISBbits.TRISB5 = 1; } while(0)
108 #define SDA2_SetDigitalOutput() do { TRISBbits.TRISB5 = 0; } while(0)
109 #define SDA2_SetPullup()   do { WPUBBbits.WPUB5 = 1; } while(0)
110 #define SDA2_ResetPullup() do { WPUBBbits.WPUB5 = 0; } while(0)
111 #define SDA2_SetPushPull() do { ODCONBbits.ODCB5 = 1; } while(0)
112 #define SDA2_SetOpenDrain() do { ODCONBbits.ODCB5 = 0; } while(0)
113 #define SDA2_SetAnalogMode() do { ANSELBbits.ANSB5 = 1; } while(0)
114 #define SDA2_SetDigitalMode() do { ANSELBbits.ANSB5 = 0; } while(0)
115
116 // get/set IO_RB6 aliases
117 #define IO_RB6_TRIS          TRISBbits.TRISB6
118 #define IO_RB6_LAT           LATBbits.LATB6
119 #define IO_RB6_PORT          PORTBbits.RB6
120 #define IO_RB6_WPU           WPUBBbits.WPUB6
121 #define IO_RB6_OD            ODCONBbits.ODCB6
122 #define IO_RB6_ANS           ANSELBbits.ANSB6
123 #define IO_RB6_SetHigh()     do { LATBbits.LATB6 = 1; } while(0)
124 #define IO_RB6_SetLow()      do { LATBbits.LATB6 = 0; } while(0)
125 #define IO_RB6_Toggle()      do { LATBbits.LATB6 = ~LATBbits.LATB6; }
126 while(0)
127 #define IO_RB6_GetValue()    PORTBbits.RB6
128 #define IO_RB6_SetDigitalInput() do { TRISBbits.TRISB6 = 1; } while(0)

```

```

126 #define IO_RB6_SetDigitalOutput()    do { TRISBbits.TRISB6 = 0; } while(0)
127 #define IO_RB6_SetPullup()         do { WPUBBbits.WPUB6 = 1; } while(0)
128 #define IO_RB6_ResetPullup()      do { WPUBBbits.WPUB6 = 0; } while(0)
129 #define IO_RB6_SetPushPull()       do { ODCONBbits.ODCB6 = 1; } while(0)
130 #define IO_RB6_SetOpenDrain()      do { ODCONBbits.ODCB6 = 0; } while(0)
131 #define IO_RB6_SetAnalogMode()     do { ANSELBbits.ANSB6 = 1; } while(0)
132 #define IO_RB6_SetDigitalMode()    do { ANSELBbits.ANSB6 = 0; } while(0)
133
134 // get/set SCL2 aliases
135 #define SCL2_TRIS                  TRISBbits.TRISB7
136 #define SCL2_LAT                   LATBbits.LATB7
137 #define SCL2_PORT                  PORTBbits.RB7
138 #define SCL2_WPU                   WPUBBbits.WPUB7
139 #define SCL2_OD                    ODCONBbits.ODCB7
140 #define SCL2_ANS                   ANSELBbits.ANSB7
141 #define SCL2_SetHigh()             do { LATBbits.LATB7 = 1; } while(0)
142 #define SCL2_SetLow()              do { LATBbits.LATB7 = 0; } while(0)
143 #define SCL2_Toggle()              do { LATBbits.LATB7 = ~LATBbits.LATB7; }
144 while(0)
145 #define SCL2_GetValue()            PORTBbits.RB7
146 #define SCL2_SetDigitalInput()      do { TRISBbits.TRISB7 = 1; } while(0)
147 #define SCL2_SetDigitalOutput()     do { TRISBbits.TRISB7 = 0; } while(0)
148 #define SCL2_SetPullup()           do { WPUBBbits.WPUB7 = 1; } while(0)
149 #define SCL2_ResetPullup()         do { WPUBBbits.WPUB7 = 0; } while(0)
150 #define SCL2_SetPushPull()         do { ODCONBbits.ODCB7 = 1; } while(0)
151 #define SCL2_SetOpenDrain()        do { ODCONBbits.ODCB7 = 0; } while(0)
152 #define SCL2_SetAnalogMode()       do { ANSELBbits.ANSB7 = 1; } while(0)
153 #define SCL2_SetDigitalMode()      do { ANSELBbits.ANSB7 = 0; } while(0)
154
155 // get/set IO_RC0 aliases
156 #define IO_RC0_TRIS                TRISCbits.TRISCO
157 #define IO_RC0_LAT                 LATCbits.LATC0
158 #define IO_RC0_PORT                PORTCbits.RC0
159 #define IO_RC0_WPU                 WPUCbits.WPUC0
160 #define IO_RC0_OD                  ODCONCbits.ODCC0
161 #define IO_RC0_ANS                 ANSELCbits.ANSC0
162 #define IO_RC0_SetHigh()           do { LATCbits.LATC0 = 1; } while(0)
163 #define IO_RC0_SetLow()             do { LATCbits.LATC0 = 0; } while(0)
164 while(0)
165 #define IO_RC0_Toggle()            do { LATCbits.LATC0 = ~LATCbits.LATC0; }
166 #define IO_RC0_GetValue()          PORTCbits.RC0
167 #define IO_RC0_SetDigitalInput()    do { TRISCbits.TRISCO = 1; } while(0)
168 #define IO_RC0_SetDigitalOutput()   do { TRISCbits.TRISCO = 0; } while(0)
169 #define IO_RC0_SetPullup()          do { WPUCbits.WPUC0 = 1; } while(0)
170 #define IO_RC0_ResetPullup()        do { WPUCbits.WPUC0 = 0; } while(0)
171 #define IO_RC0_SetPushPull()        do { ODCONCbits.ODCC0 = 1; } while(0)
172 #define IO_RC0_SetOpenDrain()       do { ODCONCbits.ODCC0 = 0; } while(0)
173 #define IO_RC0_SetAnalogMode()      do { ANSELCbits.ANSC0 = 1; } while(0)
174 #define IO_RC0_SetDigitalMode()     do { ANSELCbits.ANSC0 = 0; } while(0)
175
176 // get/set IO_RC1 aliases
177 #define IO_RC1_TRIS                TRISCbits.TRISC1
178 #define IO_RC1_LAT                 LATCbits.LATC1
179 #define IO_RC1_PORT                PORTCbits.RC1
180 #define IO_RC1_WPU                 WPUCbits.WPUC1
181 #define IO_RC1_OD                  ODCONCbits.ODCC1
182 #define IO_RC1_ANS                 ANSELCbits.ANSC1
183 #define IO_RC1_SetHigh()           do { LATCbits.LATC1 = 1; } while(0)
184 #define IO_RC1_SetLow()             do { LATCbits.LATC1 = 0; } while(0)
185 #define IO_RC1_Toggle()             do { LATCbits.LATC1 = ~LATCbits.LATC1; }
186 #define IO_RC1_GetValue()          PORTCbits.RC1
187 #define IO_RC1_SetDigitalInput()    do { TRISCbits.TRISC1 = 1; } while(0)
188 #define IO_RC1_SetDigitalOutput()   do { TRISCbits.TRISC1 = 0; } while(0)
189 #define IO_RC1_SetPullup()          do { WPUCbits.WPUC1 = 1; } while(0)

```

```

188 #define IO_RC1_ResetPullup()      do { WPUCBbits.WPUC1 = 0; } while(0)
189 #define IO_RC1_SetPushPull()     do { ODCONCbbits.ODCC1 = 1; } while(0)
190 #define IO_RC1_SetOpenDrain()    do { ODCONCbbits.ODCC1 = 0; } while(0)
191 #define IO_RC1_SetAnalogMode()   do { ANSELBbits.ANSC1 = 1; } while(0)
192 #define IO_RC1_SetDigitalMode()  do { ANSELBbits.ANSC1 = 0; } while(0)
193
194 // get/set IO_RC2 aliases
195 #define IO_RC2_TRIS           TRISCbits.TRISC2
196 #define IO_RC2_LAT            LATCbits.LATC2
197 #define IO_RC2_PORT          PORTCbits.RC2
198 #define IO_RC2_WPU            WPUCBbits.WPUC2
199 #define IO_RC2_OD             ODCONCbbits.ODCC2
200 #define IO_RC2_ANS            ANSELBbits.ANSC2
201 #define IO_RC2_SetHigh()       do { LATCbits.LATC2 = 1; } while(0)
202 #define IO_RC2_SetLow()        do { LATCbits.LATC2 = 0; } while(0)
203 #define IO_RC2_Toggle()        do { LATCbits.LATC2 = ~LATCbits.LATC2; }
204 while(0)
205 #define IO_RC2_GetValue()      PORTCbits.RC2
206 #define IO_RC2_SetDigitalInput() do { TRISCbits.TRISC2 = 1; } while(0)
207 #define IO_RC2_SetDigitalOutput() do { TRISCbits.TRISC2 = 0; } while(0)
208 #define IO_RC2_SetPullup()     do { WPUCBbits.WPUC2 = 1; } while(0)
209 #define IO_RC2_ResetPullup()   do { WPUCBbits.WPUC2 = 0; } while(0)
210 #define IO_RC2_SetPushPull()   do { ODCONCbbits.ODCC2 = 1; } while(0)
211 #define IO_RC2_SetOpenDrain()  do { ODCONCbbits.ODCC2 = 0; } while(0)
212 #define IO_RC2_SetAnalogMode() do { ANSELBbits.ANSC2 = 1; } while(0)
213 #define IO_RC2_SetDigitalMode() do { ANSELBbits.ANSC2 = 0; } while(0)
214
215 // get/set IO_RC3 aliases
216 #define IO_RC3_TRIS           TRISCbits.TRISC3
217 #define IO_RC3_LAT            LATCbits.LATC3
218 #define IO_RC3_PORT          PORTCbits.RC3
219 #define IO_RC3_WPU            WPUCBbits.WPUC3
220 #define IO_RC3_OD             ODCONCbbits.ODCC3
221 #define IO_RC3_ANS            ANSELBbits.ANSC3
222 #define IO_RC3_SetHigh()       do { LATCbits.LATC3 = 1; } while(0)
223 #define IO_RC3_SetLow()        do { LATCbits.LATC3 = 0; } while(0)
224 #define IO_RC3_Toggle()        do { LATCbits.LATC3 = ~LATCbits.LATC3; }
225 while(0)
226 #define IO_RC3_GetValue()      PORTCbits.RC3
227 #define IO_RC3_SetDigitalInput() do { TRISCbits.TRISC3 = 1; } while(0)
228 #define IO_RC3_SetDigitalOutput() do { TRISCbits.TRISC3 = 0; } while(0)
229 #define IO_RC3_SetPullup()     do { WPUCBbits.WPUC3 = 1; } while(0)
230 #define IO_RC3_ResetPullup()   do { WPUCBbits.WPUC3 = 0; } while(0)
231 #define IO_RC3_SetPushPull()   do { ODCONCbbits.ODCC3 = 1; } while(0)
232 #define IO_RC3_SetOpenDrain()  do { ODCONCbbits.ODCC3 = 0; } while(0)
233 #define IO_RC3_SetAnalogMode() do { ANSELBbits.ANSC3 = 1; } while(0)
234 #define IO_RC3_SetDigitalMode() do { ANSELBbits.ANSC3 = 0; } while(0)
235
236 // get/set RC4 procedures
237 #define RC4_SetHigh()         do { LATCbits.LATC4 = 1; } while(0)
238 #define RC4_SetLow()          do { LATCbits.LATC4 = 0; } while(0)
239 #define RC4_Toggle()          do { LATCbits.LATC4 = ~LATCbits.LATC4; } while(0)
240 #define RC4_GetValue()        PORTCbits.RC4
241 #define RC4_SetDigitalInput()  do { TRISCbits.TRISC4 = 1; } while(0)
242 #define RC4_SetDigitalOutput() do { TRISCbits.TRISC4 = 0; } while(0)
243 #define RC4_SetPullup()       do { WPUCBbits.WPUC4 = 1; } while(0)
244 #define RC4_ResetPullup()     do { WPUCBbits.WPUC4 = 0; } while(0)
245 #define RC4_SetPushPull()     do { ANSELBbits.ANSC4 = 1; } while(0)
246 #define RC4_SetOpenDrain()    do { ANSELBbits.ANSC4 = 0; } while(0)
247
248 // get/set RC5 procedures
249 #define RC5_SetHigh()         do { LATCbits.LATC5 = 1; } while(0)
250 #define RC5_SetLow()          do { LATCbits.LATC5 = 0; } while(0)
251 #define RC5_Toggle()          do { LATCbits.LATC5 = ~LATCbits.LATC5; } while(0)
252 #define RC5_GetValue()        PORTCbits.RC5

```

```

251 #define RC5_SetDigitalInput()    do { TRISCbits.TRISC5 = 1; } while(0)
252 #define RC5_SetDigitalOutput()   do { TRISCbits.TRISC5 = 0; } while(0)
253 #define RC5_SetPullup()        do { WPUCbits.WPUC5 = 1; } while(0)
254 #define RC5_ResetPullup()      do { WPUCbits.WPUC5 = 0; } while(0)
255 #define RC5_SetAnalogMode()    do { ANSELCbits.ANSC5 = 1; } while(0)
256 #define RC5_SetDigitalMode()   do { ANSELCbits.ANSC5 = 0; } while(0)
257
258 // get/set IO_RC6 aliases
259 #define IO_RC6_TRIS           TRISCbits.TRISC6
260 #define IO_RC6_LAT             LATCbits.LATC6
261 #define IO_RC6_PORT            PORTCbits.RC6
262 #define IO_RC6_WPU              WPUCbits.WPUC6
263 #define IO_RC6_OD               ODCONCbits.ODCC6
264 #define IO_RC6_ANS              ANSELCbits.ANSC6
265 #define IO_RC6_SetHigh()       do { LATCbits.LATC6 = 1; } while(0)
266 #define IO_RC6_SetLow()        do { LATCbits.LATC6 = 0; } while(0)
267 #define IO_RC6_Toggle()        do { LATCbits.LATC6 = ~LATCbits.LATC6; }
268 while(0)
269 #define IO_RC6_GetValue()      PORTCbits.RC6
270
271 #define IO_RC6_SetDigitalInput() do { TRISCbits.TRISC6 = 1; } while(0)
272 #define IO_RC6_SetDigitalOutput() do { TRISCbits.TRISC6 = 0; } while(0)
273 #define IO_RC6_SetPullup()     do { WPUCbits.WPUC6 = 1; } while(0)
274 #define IO_RC6_ResetPullup()   do { WPUCbits.WPUC6 = 0; } while(0)
275 #define IO_RC6_SetPushPull()   do { ODCONCbits.ODCC6 = 1; } while(0)
276 #define IO_RC6_SetOpenDrain()  do { ODCONCbits.ODCC6 = 0; } while(0)
277 #define IO_RC6_SetAnalogMode() do { ANSELCbits.ANSC6 = 1; } while(0)
278 #define IO_RC6_SetDigitalMode() do { ANSELCbits.ANSC6 = 0; } while(0)
279
280 // get/set IO_RC7 aliases
281 #define IO_RC7_TRIS           TRISCbits.TRISC7
282 #define IO_RC7_LAT             LATCbits.LATC7
283 #define IO_RC7_PORT            PORTCbits.RC7
284 #define IO_RC7_WPU              WPUCbits.WPUC7
285 #define IO_RC7_OD               ODCONCbits.ODCC7
286 #define IO_RC7_ANS              ANSELCbits.ANSC7
287 #define IO_RC7_SetHigh()       do { LATCbits.LATC7 = 1; } while(0)
288 #define IO_RC7_SetLow()        do { LATCbits.LATC7 = 0; } while(0)
289 #define IO_RC7_Toggle()        do { LATCbits.LATC7 = ~LATCbits.LATC7; }
290
291 #define IO_RC7_GetValue()      PORTCbits.RC7
292
293 #define IO_RC7_SetDigitalInput() do { TRISCbits.TRISC7 = 1; } while(0)
294 #define IO_RC7_SetDigitalOutput() do { TRISCbits.TRISC7 = 0; } while(0)
295 #define IO_RC7_SetPullup()     do { WPUCbits.WPUC7 = 1; } while(0)
296 #define IO_RC7_ResetPullup()   do { WPUCbits.WPUC7 = 0; } while(0)
297 #define IO_RC7_SetPushPull()   do { ODCONCbits.ODCC7 = 1; } while(0)
298 #define IO_RC7_SetOpenDrain()  do { ODCONCbits.ODCC7 = 0; } while(0)
299 #define IO_RC7_SetAnalogMode() do { ANSELCbits.ANSC7 = 1; } while(0)
300 #define IO_RC7_SetDigitalMode() do { ANSELCbits.ANSC7 = 0; } while(0)
301
302 /**
303  * @Param
304  *   none
305  * @Returns
306  *   none
307  * @Description
308  *   GPIO and peripheral I/O initialization
309  * @Example
310  *   PIN_MANAGER_Initialize();
311  */
312 void PIN_MANAGER_Initialize (void);
313
314 /**
315  * @Param
316  *   none
317  * @Returns

```

```
314     none
315     * @Description
316     Interrupt on Change Handling routine
317     * @Example
318     PIN_MANAGER_IOC();
319     */
320 void PIN_MANAGER_IOC(void);
321
322
323
324 #endif // PIN_MANAGER_H
325 /**
326  End of File
327 */
```

```

1  /**
2   * TMR0 Generated Driver File
3
4   * @Company
5   *   Microchip Technology Inc.
6
7   * @File Name
8   *   tmr0.c
9
10  * @Summary
11  *   This is the generated driver implementation file for the TMR0 driver using
12  *   MPLAB(c) Code Configurator
13
14  * @Description
15  *   This source file provides APIs for TMR0.
16  *   Generation Information :
17  *     Product Revision : MPLAB(c) Code Configurator - 4.15.5
18  *     Device          : PIC16F18345
19  *     Driver Version  : 2.00
20  *     The generated drivers are tested against the following:
21  *       Compiler      : XC8 1.35
22  *       MPLAB         : MPLAB X 3.40
23
24 */
25
26 /*
27  * (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
28  * software and any derivatives exclusively with Microchip products.
29
30  * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
31  * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
32  * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
33  * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
34  * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
35
36  * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
37  * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
38  * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
39  * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
40  * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
41  * ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
42  * THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
43
44  * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
45  * TERMS.
46 */
47
48 /**
49  * Section: Included Files
50 */
51
52
53 /**
54  * Section: Global Variables Definitions
55 */
56
57 volatile uint16_t timer0ReloadVal16bit;
58 int ONS = 0;
59
60 /**
61  * Section: TMR0 APIs
62 */
63
64

```

```

65 void TMRO_Wipe(void)
66 {
67     TMRO_Write16bitTimer(0);
68     PIR0bits.TMROIF=0;
69 }
70
71 void TMRO_Initialize(void)
72 {
73     // Set TMRO to the options selected in the User Interface
74
75     // TOOUTPS 1:1; TOEN disabled; T016BIT 16-bit;
76     TOCON0 = 0x10;
77
78     // TOCS FOSC/4; TOCKPS 1:256; T0ASYNC synchronised;
79     TOCON1 = 0x48;
80
81     // TMROH 11;
82     TMROH = 0x0B;
83
84     // TMROL 220;
85     TMROL = 0xDC;
86
87     // Load TMRO value to the 16-bit reload variable
88     timer0ReloadVal16bit = (TMROH << 8) | TMROL;
89
90
91     // Clearing IF flag
92     PIR0bits.TMROIF = 0;
93
94     // Start TMRO
95     TMRO_StartTimer();
96 }
97
98 void TMRO_StartTimer(void)
99 {
100    // Start the Timer by writing to TMROON bit
101    TOCON0bits.TOEN = 1;
102 }
103
104 void TMRO_StopTimer(void)
105 {
106    // Stop the Timer by writing to TMROON bit
107    TOCON0bits.TOEN = 0;
108 }
109
110 uint16_t TMRO_Read16bitTimer(void)
111 {
112     uint16_t readVal;
113     uint8_t readValLow;
114     uint8_t readValHigh;
115
116     readValLow = TMROL;
117     readValHigh = TMROH;
118     readVal = (uint16_t)readValHigh << 8) + readValLow;
119
120     return readVal;
121 }
122
123 void TMRO_Write16bitTimer(uint16_t timerVal)
124 {
125     // Write to the Timer0 register
126     TMROH = timerVal >> 8;
127     TMROL = (uint8_t) timerVal;
128 }
129

```

```

130 void TMR0_Reload16bit(void)
131 {
132     // Write to the Timer0 register
133     TMROH = timer0ReloadVal16bit >> 8;
134     TMROL = (uint8_t) timer0ReloadVal16bit;
135 }
136
137 bool TMR0_HasOverflowOccured(void)
138 {
139     // check if overflow has occurred by checking the TMR0IF bit
140     //return(PIR0bits.TMR0IF);
141     bool status = PIR0bits.TMR0IF;
142     if(status)
143     {
144         // Clearing IF flag.
145         PIR0bits.TMR0IF = 0;
146     }
147     return status;
148 }
149
150
151 void ms_delay30(uint16_t delay){
152     uint16_t timerVal = 0;
153     TMRO_Write16bitTimer(0);
154     while(delay > timerVal){
155         timerVal = TMRO_Read16bitTimer();
156     }
157 }
158
159 void TMR0_check_Inactive(int next_read){
160     if (ONS < 1 && next_read < 1) { //just load the first timer value
161         TMRO_Wipe();
162         TMRO_StartTimer();
163         ONS = 1;
164     } else if (ONS == 1 && next_read == 1) { //start continuous
165     } else {
166     }
167 }
168
169
170 void TMR0_clear_ONS(void){
171     ONS = 0;
172 }
173
174
175 /**
176  * End of File
177 */

```

```

1  /**
2   TMRO Generated Driver API Header File
3
4  @Company
5   Microchip Technology Inc.
6
7  @File Name
8   tmr0.h
9
10 @Summary
11   This is the generated header file for the TMRO driver using MPLAB(c) Code
12  Configurator
13 @Description
14   This header file provides APIs for TMRO.
15  Generation Information :
16    Product Revision : MPLAB(c) Code Configurator - 4.15.5
17    Device          : PIC16F18345
18    Driver Version  : 2.00
19    The generated drivers are tested against the following:
20    Compiler        : XC8 1.35
21    MPLAB          : MPLAB X 3.40
22 */
23
24 /*
25  (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
26  software and any derivatives exclusively with Microchip products.
27
28 THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
29 EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
30 WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
31 PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
32 WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
33
34 IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
35 INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
36 WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
37 BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
38 FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
39 ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
40 THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
41
42 MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
43 TERMS.
44 */
45
46 #ifndef _TMRO_H
47 #define _TMRO_H
48
49 /**
50  Section: Included Files
51 */
52
53 #include <stdint.h>
54 #include <stdbool.h>
55
56 #ifdef __cplusplus // Provide C++ Compatibility
57
58     extern "C" {
59
60 #endif
61
62
63
64 /**

```

```

65     Section: TMR0 APIs
66 */
67 /**
68  * @Summary
69  *   Initializes the TMR0.
70  *
71  * @Description
72  *   This function initializes the TMR0 Registers.
73  *   This function must be called before any other TMR0 function is called.
74  *
75  * @Preconditions
76  *   None
77  *
78  * @Param
79  *   None
80  *
81  * @Returns
82  *   None
83  *
84  * @Comment
85  *
86  *
87  * @Example
88  *   <code>
89  *     main()
90  *   {
91  *     // Initialize TMR0 module
92  *     TMR0_Initialize();
93  *
94  *     // Do something else...
95  *   }
96  *   </code>
97  */
98 void TMR0_Initialize(void);
99
100 /**
101 * @Summary
102 *   This function starts the TMR0.
103 *
104 * @Description
105 *   This function starts the TMR0 operation.
106 *   This function must be called after the initialization of TMR0.
107 *
108 * @Preconditions
109 *   Initialize the TMR0 before calling this function.
110 *
111 * @Param
112 *   None
113 *
114 * @Returns
115 *   None
116 *
117 * @Example
118 *   <code>
119 *     // Initialize TMR0 module
120 *
121 *     // Start TMR0
122 *     TMR0_StartTimer();
123 *
124 *     // Do something else...
125 *   </code>
126 */
127 void TMR0_StartTimer(void);
128
129

```

```

130  /**
131   * @Summary
132   *   This function stops the TMRO.
133
134   * @Description
135   *   This function stops the TMRO operation.
136   *   This function must be called after the start of TMRO.
137
138   * @Preconditions
139   *   Initialize the TMRO before calling this function.
140
141   * @Param
142   *   None
143
144   * @Returns
145   *   None
146
147   * @Example
148   *   <code>
149   *     // Initialize TMRO module
150
151   *     // Start TMRO
152   *     TMRO_StartTimer();
153
154   *     // Do something else...
155
156   *     // Stop TMRO;
157   *     TMRO_StopTimer();
158   *   </code>
159 */
160 void TMRO_StopTimer(void);

162 /**
163  * @Summary
164  *   Reads the 16 bits TMRO register value.
165
166  * @Description
167  *   This function reads the 16 bits TMRO register value and return it.
168
169  * @Preconditions
170  *   Initialize the TMRO before calling this function.
171
172  * @Param
173  *   None
174
175  * @Returns
176  *   This function returns the 16 bits value of TMRO register.
177
178  * @Example
179  *   <code>
180  *     // Initialize TMRO module
181
182  *     // Start TMRO
183  *     TMRO_StartTimer();
184
185  *     // Read the current value of TMRO
186  *     if(0 == TMRO_Read16bitTimer())
187  *     {
188  *       // Do something else...
189
190  *       // Reload the TMR value
191  *       TMRO_Reload();
192  *     }
193  *   </code>

```

```

195  */
196  uint16_t TMR0_Read16bitTimer(void);
197 /**
198  * @Summary
199  *   Writes the 16 bits value to TMR0 register.
200
201  * @Description
202  *   This function writes the 16 bits value to TMR0 register.
203  *   This function must be called after the initialization of TMR0.
204
205  * @Preconditions
206  *   Initialize the TMR0 before calling this function.
207
208  * @Param
209  *   timerVal - Value to write into TMR0 register.
210
211  * @Returns
212  *   None
213
214  * @Example
215  *   <code>
216  *     #define PERIOD 0x8000
217  *     #define ZERO    0x0000
218
219
220     while(1)
221     {
222         //Read the TMR0 register
223         if(ZERO == TMR0_Read16bitTimer())
224         {
225             // Do something else...
226
227             // Write the TMR0 register
228             TMR0_Write16bitTimer(PERIOD);
229         }
230
231         // Do something else...
232     }
233     </code>
234 */
235 void TMR0_Write16bitTimer(uint16_t timerVal);
236
237 /**
238  * @Summary
239  *   Reload the 16 bits value to TMR0 register.
240
241  * @Description
242  *   This function reloads the 16 bit value to TMR0 register.
243  *   This function must be called to write initial value into TMR0 register.
244
245  * @Preconditions
246  *   Initialize the TMR0 before calling this function.
247
248  * @Param
249  *   None
250
251  * @Returns
252  *   None
253
254  * @Example
255  *   <code>
256  *     while(1)
257  *     {
258         if(TMR0IF)
259         {

```

```

260         // Do something else...
261
262         // clear the TMR0 interrupt flag
263         TMR0IF = 0;
264
265         // Reload the initial value of TMR0
266         TMR0_Reload16bit();
267     }
268 }
269 </code>
270 */
271 void TMR0_Reload16bit(void);
272
273 /**
274  * @Summary
275  * Boolean routine to poll or to check for the overflow flag on the fly.
276  *
277  * @Description
278  * This function is called to check for the timer overflow flag.
279  * This function is used in timer polling method.
280  *
281  * @Preconditions
282  * Initialize the TMR0 module before calling this routine.
283  *
284  * @Param
285  * None
286  *
287  * @Returns
288  * true - timer overflow has occurred.
289  * false - timer overflow has not occurred.
290  *
291  * @Example
292  * <code>
293  * while(1)
294  * {
295  *     // check the overflow flag
296  *     if(TMR0_HasOverflowOccured())
297  *     {
298  *         // Do something else...
299  *
300  *         // clear the TMR0 interrupt flag
301  *         TMR0IF = 0;
302  *
303  *         // Reload the TMR0 value
304  *         TMR0_Reload16bit();
305  *     }
306  * }
307 </code>
308 */
309 bool TMR0_HasOverflowOccured(void);
310
311 void ms_delay30(uint16_t delay);
312
313 void TMR0_Wipe(void);
314
315 void TMR0_check_Inactive(int next_read);
316
317 void TMR0_clear_ONS(void);
318
319 #ifdef __cplusplus // Provide C++ Compatibility
320 }
321
322 #endif
323
324

```

```
325 #endif // _TMR0_H
326 /**
327 End of File
328 */
```

```

1  /**
2   * TMR1 Generated Driver File
3
4   * @Company
5   *   Microchip Technology Inc.
6
7   * @File Name
8   *   tmr1.c
9
10  * @Summary
11  *   This is the generated driver implementation file for the TMR1 driver using
12  *   MPLAB(c) Code Configurator
13
14  * @Description
15  *   This source file provides APIs for TMR1.
16  *   Generation Information :
17  *     Product Revision : MPLAB(c) Code Configurator - 4.15.5
18  *     Device          : PIC16F18345
19  *     Driver Version  : 2.00
20  *     The generated drivers are tested against the following:
21  *       Compiler      : XC8 1.35
22  *       MPLAB         : MPLAB X 3.40
23
24  */
25
26  /*
27  *   (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
28  *   software and any derivatives exclusively with Microchip products.
29
30  * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
31  * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
32  * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
33  * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
34  * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
35
36  * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
37  * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
38  * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
39  * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
40  * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
41  * ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
42  * THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
43
44  * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
45  * TERMS.
46
47  /**
48  *   Section: Included Files
49  */
50
51  #include <xc.h>
52  #include "tmr1.h"
53  #include "../FastTransfer.h"
54  #include "../LIDAR.h"
55  #include "eusart.h"
56
57  /**
58  *   Section: Global Variables Definitions
59  */
60  volatile uint16_t timer1ReloadVal;
61  void (*TMR1_InterruptHandler)(void);
62
63  /**
64  *   Section: TMR1 APIs
65  */

```

```

65
66 void TMR1_Initialize(void)
67 {
68     //Set the Timer to the options selected in the GUI
69
70     //T1CKPS 1:1; T1SOSC T1CKI_enabled; T1SYNC synchronize; TMR1CS FOSC; TMR1ON
71     //off;
72     T1CON = 0x40;
73
74     //T1GSS T1G_pin; TMR1GE disabled; T1GTM disabled; T1GPOL low; T1GGO_nDONE
75     //done; T1GSPM disabled;
76     T1GCON = 0x00;
77
78     //TMR1H 131;
79     TMR1H = 0x83;
80
81     //TMR1L 0;
82     TMR1L = 0x00;
83
84     // Load the TMR value to reload variable
85     timer1ReloadVal=(TMR1H << 8) | TMR1L;
86
87     // Clearing IF flag.
88     PIR1bits.TMR1IF = 0;
89
90 }
91
92 void TMR1_StartTimer(void)
93 {
94     // Start the Timer by writing to TMRxON bit
95     T1CONbits.TMR1ON = 1;
96 }
97
98 void TMR1_StopTimer(void)
99 {
100    // Stop the Timer by writing to TMRxON bit
101    T1CONbits.TMR1ON = 0;
102 }
103
104 uint16_t TMR1_ReadTimer(void)
105 {
106     uint16_t readVal;
107
108     readVal = (TMR1H << 8) | TMR1L;
109
110     return readVal;
111 }
112
113 void TMR1_WriteTimer(uint16_t timerVal)
114 {
115     if (T1CONbits.T1SYNC == 1)
116     {
117         // Stop the Timer by writing to TMRxON bit
118         T1CONbits.TMR1ON = 0;
119
120         // Write to the Timer1 register
121         TMR1H = (timerVal >> 8);
122         TMR1L = timerVal;
123
124         // Start the Timer after writing to the register
125         T1CONbits.TMR1ON =1;
126     }
127     else

```

```

128     {
129         // Write to the Timer1 register
130         TMR1H = (timerVal >> 8);
131         TMR1L = timerVal;
132     }
133 }
134
135 void TMR1_Reload(void)
136 {
137     //Write to the Timer1 register
138     TMR1H = (timer1ReloadVal >> 8);
139     TMR1L = timer1ReloadVal;
140 }
141
142 void TMR1_StartSinglePulseAcquisition(void)
143 {
144     T1GCONbits.T1GGO_nDONE = 1;
145 }
146
147 uint8_t TMR1_CheckGateValueStatus(void)
148 {
149     return (T1GCONbits.T1GVAL);
150 }
151
152 bool TMR1_HasOverflowOccured(void)
153 {
154     // check if overflow has occurred by checking the TMRIF bit
155     return(PIR1bits.TMR1IF);
156 }
157
158 void ms_delay1(uint16_t delay){
159     uint16_t timerVal = 0;
160     TMR1_WriteTimer(0);
161     while(delay > timerVal){
162         respond_to_MCB();
163         checkCommsEnable();
164         timerVal = TMR1_ReadTimer();
165     }
166 }
167 /**
168  * End of File
169 */

```

```

1  /**
2   * TMR1 Generated Driver API Header File
3
4  @Company
5   Microchip Technology Inc.
6
7  @File Name
8   tmr1.h
9
10 @Summary
11   This is the generated header file for the TMR1 driver using MPLAB(c) Code
12  Configurator
13 @Description
14   This header file provides APIs for driver for TMR1.
15  Generation Information :
16      Product Revision : MPLAB(c) Code Configurator - 4.15.5
17      Device          : PIC16F18345
18      Driver Version  : 2.00
19      The generated drivers are tested against the following:
20      Compiler        : XC8 1.35
21      MPLAB          : MPLAB X 3.40
22 */
23
24 /*
25  (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
26  software and any derivatives exclusively with Microchip products.
27
28 THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
29 EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
30 WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
31 PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
32 WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
33
34 IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
35 INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
36 WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
37 BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
38 FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
39 ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
40 THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
41
42 MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
43 TERMS.
44 */
45
46 #ifndef _TMR1_H
47 #define _TMR1_H
48
49 /**
50  Section: Included Files
51 */
52
53 #include <stdbool.h>
54 #include <stdint.h>
55
56 #ifdef __cplusplus // Provide C++ Compatibility
57
58     extern "C" {
59
60 #endif
61
62 /**
63  Section: TMR1 APIs

```

```

65  */
66 /**
67  * @Summary
68  *     Initializes the TMR1
69
70  * @Description
71  *     This routine initializes the TMR1.
72  *     This routine must be called before any other TMR1 routine is called.
73  *     This routine should only be called once during system initialization.
74
75  * @Preconditions
76  *     None
77
78  * @Param
79  *     None
80
81  * @Returns
82  *     None
83
84  * @Comment
85
86
87  * @Example
88  *     <code>
89  *         main()
90  *     {
91  *         // Initialize TMR1 module
92  *         TMR1_Initialize();
93
94  *         // Do something else...
95  *     }
96  *     </code>
97  */
98 void TMR1_Initialize(void);
99
100 /**
101  * @Summary
102  *     This function starts the TMR1.
103
104  * @Description
105  *     This function starts the TMR1 operation.
106  *     This function must be called after the initialization of TMR1.
107
108  * @Preconditions
109  *     Initialize the TMR1 before calling this function.
110
111  * @Param
112  *     None
113
114  * @Returns
115  *     None
116
117  * @Example
118  *     <code>
119  *         // Initialize TMR1 module
120
121  *         // Start TMR1
122  *         TMR1_StartTimer();
123
124  *         // Do something else...
125  *     </code>
126  */
127 void TMR1_StartTimer(void);
128
129

```

```

130  /**
131   * @Summary
132   *   This function stops the TMR1.
133
134   * @Description
135   *   This function stops the TMR1 operation.
136   *   This function must be called after the start of TMR1.
137
138   * @Preconditions
139   *   Initialize the TMR1 before calling this function.
140
141   * @Param
142   *   None
143
144   * @Returns
145   *   None
146
147   * @Example
148   *   <code>
149   *     // Initialize TMR1 module
150
151   *     // Start TMR1
152   *     TMR1_StartTimer();
153
154   *     // Do something else...
155
156   *     // Stop TMR1;
157   *     TMR1_StopTimer();
158   *   </code>
159 */
160 void TMR1_StopTimer(void);
161
162 /**
163  * @Summary
164  *   Reads the TMR1 register.
165
166  * @Description
167  *   This function reads the TMR1 register value and return it.
168
169  * @Preconditions
170  *   Initialize the TMR1 before calling this function.
171
172  * @Param
173  *   None
174
175  * @Returns
176  *   This function returns the current value of TMR1 register.
177
178  * @Example
179  *   <code>
180  *     // Initialize TMR1 module
181
182  *     // Start TMR1
183  *     TMR1_StartTimer();
184
185  *     // Read the current value of TMR1
186  *     if(0 == TMR1_ReadTimer())
187  *     {
188  *       // Do something else...
189
190  *       // Reload the TMR value
191  *       TMR1_Reload();
192  *     }
193  *   </code>
194 */

```

```

195  uint16_t TMR1_ReadTimer(void);
196
197 /**
198  * @Summary
199  *   Writes the TMR1 register.
200
201  * @Description
202  *   This function writes the TMR1 register.
203  *   This function must be called after the initialization of TMR1.
204
205  * @Preconditions
206  *   Initialize the TMR1 before calling this function.
207
208  * @Param
209  *   timerVal - Value to write into TMR1 register.
210
211  * @Returns
212  *   None
213
214  * @Example
215  * <code>
216  * #define PERIOD 0x80
217  * #define ZERO    0x00
218
219  * while(1)
220  {
221      // Read the TMR1 register
222      if(ZERO == TMR1_ReadTimer())
223      {
224          // Do something else...
225
226          // Write the TMR1 register
227          TMR1_WriteTimer(PERIOD);
228      }
229
230      // Do something else...
231  }
232  </code>
233 */
234 void TMR1_WriteTimer(uint16_t timerVal);
235
236 /**
237  * @Summary
238  *   Reload the TMR1 register.
239
240  * @Description
241  *   This function reloads the TMR1 register.
242  *   This function must be called to write initial value into TMR1 register.
243
244  * @Preconditions
245  *   Initialize the TMR1 before calling this function.
246
247  * @Param
248  *   None
249
250  * @Returns
251  *   None
252
253  * @Example
254  * <code>
255  * while(1)
256  {
257      if(TMR1IF)
258      {
259          // Do something else...

```

```

260             // clear the TMR1 interrupt flag
261             TMR1IF = 0;
262
263             // Reload the initial value of TMR1
264             TMR1_Reload();
265         }
266     }
267 
```

</code>

```

268 */
269 void TMR1_Reload(void);
270
271 /**
272  * @Summary
273  * Starts the single pulse acquisition in TMR1 gate operation.
274
275  * @Description
276  * This function starts the single pulse acquisition in TMR1 gate operation.
277  * This function must be used when the TMR1 gate is enabled.
278
279  * @Preconditions
280  * Initialize the TMR1 with gate enable before calling this function.
281
282  * @Param
283  * None
284
285  * @Returns
286  * None
287
288  * @Example
289  * <code>
290  *     uint16_t xVal;
291  *     uint16_t yVal;
292
293  *     // enable TMR1 singlepulse mode
294  *     TMR1_StartSinglePulseAcquisition();
295
296  *     // check TMR1 gate status
297  *     if(TMR1_CheckGateValueStatus() == 0)
298  *         xVal = TMR1_ReadTimer();
299
300  *     // wait untill gate interrupt occured
301  *     while(TMR1GIF == 0)
302  *     {
303  *     }
304
305  *     yVal = TMR1_ReadTimer();
306  * </code>
307 */
308 void TMR1_StartSinglePulseAcquisition(void);
309
310 /**
311  * @Summary
312  * Check the current state of Timer1 gate.
313
314  * @Description
315  * This function reads the TMR1 gate value and return it.
316  * This function must be used when the TMR1 gate is enabled.
317
318  * @Preconditions
319  * Initialize the TMR1 with gate enable before calling this function.
320
321  * @Param
322  * None
323
324

```

```

325     @Returns
326     None
327
328     @Example
329     <code>
330     uint16_t xVal;
331     uint16_t yVal;
332
333     // enable TMR1 singlepulse mode
334     TMR1_StartSinglePulseAcquisition();
335
336     // check TMR1 gate status
337     if(TMR1_CheckGateValueStatus() == 0)
338         xVal = TMR1_ReadTimer();
339
340     // wait until gate interrupt occurred
341     while(TMR1IF == 0)
342     {
343     }
344
345     yVal = TMR1_ReadTimer();
346     </code>
347 */
348 uint8_t TMR1_CheckGateValueStatus(void);
349
350 /**
351     @Summary
352     Boolean routine to poll or to check for the overflow flag on the fly.
353
354     @Description
355     This function is called to check for the timer overflow flag.
356     This function is used in timer polling method.
357
358     @Preconditions
359     Initialize the TMR1 module before calling this routine.
360
361     @Param
362     None
363
364     @Returns
365     true - timer overflow has occurred.
366     false - timer overflow has not occurred.
367
368     @Example
369     <code>
370     while(1)
371     {
372         // check the overflow flag
373         if(TMR1_HasOverflowOccurred())
374         {
375             // Do something else...
376
377             // clear the TMR1 interrupt flag
378             TMR1IF = 0;
379
380             // Reload the TMR1 value
381             TMR1_Reload();
382         }
383     }
384     </code>
385 */
386 bool TMR1_HasOverflowOccurred(void);
387
388 void ms_delay1(uint16_t delay);
389

```

```
390 #ifdef __cplusplus // Provide C++ Compatibility
391     }
393
394 #endif
395
396 #endif // _TMR1_H
397 /**
398 * End of File
399 */
```

```

1  /**
2   * TMR2 Generated Driver File
3
4   * @Company
5   *   Microchip Technology Inc.
6
7   * @File Name
8   *   tmr2.c
9
10  * @Summary
11  *   This is the generated driver implementation file for the TMR2 driver using
12  *   MPLAB(c) Code Configurator
13
14  * @Description
15  *   This source file provides APIs for TMR2.
16  *   Generation Information :
17  *     Product Revision : MPLAB(c) Code Configurator - 4.15.5
18  *     Device          : PIC16F18345
19  *     Driver Version  : 2.00
20  *     The generated drivers are tested against the following:
21  *       Compiler      : XC8 1.35
22  *       MPLAB         : MPLAB X 3.40
23
24  */
25
26  /*
27  *   (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
28  *   software and any derivatives exclusively with Microchip products.
29
30  * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
31  * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
32  * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
33  * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
34  * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
35
36  * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
37  * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
38  * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
39  * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
40  * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
41  * ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
42  * THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
43
44  * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
45  * TERMS.
46
47  /**
48  *   Section: Included Files
49  */
50
51  #include <xc.h>
52  #include "tmr2.h"
53  #include "../FastTransfer.h"
54  #include "../LIDAR.h"
55
56  /**
57  *   Section: Global Variables Definitions
58  */
59  void (*TMR2_InterruptHandler)(void);
60
61  /**
62  *   Section: TMR2 APIs
63  */
64  void TMR2_Wipe(void)
{

```

```

65     TMR2_WriteTimer(0);
66     PIR1bits.TMR2IF=0;
67 }
68 void TMR2_Initialize(void)
69 {
70     // Set TMR2 to the options selected in the User Interface
71
72     // T2CKPS 1:1; T2OUTPS 1:8; TMR2ON off;
73     T2CON = 0x38;
74
75     // PR2 249;
76     PR2 = 0xF9;
77
78     // TMR2 0;
79     TMR2 = 0x00;
80
81     // Clearing IF flag.
82     PIR1bits.TMR2IF = 0;
83
84     // Start TMR2
85     TMR2_StartTimer();
86 }
87
88 void TMR2_StartTimer(void)
89 {
90     // Start the Timer by writing to TMRxON bit
91     T2CONbits.TMR2ON = 1;
92 }
93
94 void TMR2_StopTimer(void)
95 {
96     // Stop the Timer by writing to TMRxON bit
97     T2CONbits.TMR2ON = 0;
98 }
99
100 uint8_t TMR2_ReadTimer(void)
101 {
102     uint8_t readVal;
103
104     readVal = TMR2;
105
106     return readVal;
107 }
108
109 void TMR2_WriteTimer(uint8_t timerVal)
110 {
111     // Write to the Timer2 register
112     TMR2 = timerVal;
113 }
114
115 void TMR2_LoadPeriodRegister(uint8_t periodVal)
116 {
117     PR2 = periodVal;
118 }
119
120 bool TMR2_HasOverflowOccured(void)
121 {
122     // check if overflow has occurred by checking the TMRIF bit
123     bool status = PIR1bits.TMR2IF;
124     if(status)
125     {
126         // Clearing IF flag.
127         PIR1bits.TMR2IF = 0;
128     }
129     return status;

```

```
130 }
131
132 void us_delay250(uint16_t delay){
133     uint16_t timerVal = 0;
134     TMR2_WriteTimer(0);
135     while(delay > timerVal){
136         timerVal = TMR2_ReadTimer();
137     }
138 }
139 /**
140  * End of File
141 */
```

```

1  /**
2   * TMR2 Generated Driver API Header File
3
4   * @Company
5   *   Microchip Technology Inc.
6
7   * @File Name
8   *   tmr2.h
9
10  * @Summary
11  *   This is the generated header file for the TMR2 driver using MPLAB(c) Code
12  *   Configurator
13  * @Description
14  *   This header file provides APIs for TMR2.
15  *   Generation Information :
16  *     Product Revision : MPLAB(c) Code Configurator - 4.15.5
17  *     Device          : PIC16F18345
18  *     Driver Version  : 2.00
19  *     The generated drivers are tested against the following:
20  *       Compiler      : XC8 1.35
21  *       MPLAB         : MPLAB X 3.40
22  */
23
24  /*
25  *   (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
26  *   software and any derivatives exclusively with Microchip products.
27
28  * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
29  * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
30  * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
31  * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
32  * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
33
34  * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
35  * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
36  * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
37  * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
38  * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
39  * ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
40  * THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
41
42  * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
43  * TERMS.
44  */
45
46 #ifndef _TMR2_H
47 #define _TMR2_H
48
49 /**
50  *   Section: Included Files
51  */
52
53 #include <stdint.h>
54 #include <stdbool.h>
55
56 #ifdef __cplusplus // Provide C++ Compatibility
57
58     extern "C" {
59
60 #endif
61
62 /**
63  *   Section: Macro Declarations

```

```

65  */
66
67 /**
68   Section: TMR2 APIs
69 */
70
71 /**
72   @Summary
73     Initializes the TMR2 module.
74
75   @Description
76     This function initializes the TMR2 Registers.
77     This function must be called before any other TMR2 function is called.
78
79   @Preconditions
80     None
81
82   @Param
83     None
84
85   @Returns
86     None
87
88   @Comment
89
90
91   @Example
92     <code>
93       main()
94     {
95         // Initialize TMR2 module
96         TMR2_Initialize();
97
98         // Do something else...
99     }
100    </code>
101  */
102 void TMR2_Initialize(void);
103
104 /**
105   @Summary
106     This function starts the TMR2.
107
108   @Description
109     This function starts the TMR2 operation.
110     This function must be called after the initialization of TMR2.
111
112   @Preconditions
113     Initialize the TMR2 before calling this function.
114
115   @Param
116     None
117
118   @Returns
119     None
120
121   @Example
122     <code>
123       // Initialize TMR2 module
124
125       // Start TMR2
126       TMR2_StartTimer();
127
128       // Do something else...
129     </code>

```

```

130     */
131     void TMR2_StartTimer(void);
132
133 /**
134     @Summary
135         This function stops the TMR2.
136
137     @Description
138         This function stops the TMR2 operation.
139         This function must be called after the start of TMR2.
140
141     @Preconditions
142         Initialize the TMR2 before calling this function.
143
144     @Param
145         None
146
147     @Returns
148         None
149
150     @Example
151         <code>
152             // Initialize TMR2 module
153
154             // Start TMR2
155             TMR2_StartTimer();
156
157             // Do something else...
158
159             // Stop TMR2;
160             TMR2_StopTimer();
161         </code>
162     */
163     void TMR2_StopTimer(void);
164
165 /**
166     @Summary
167         Reads the TMR2 register.
168
169     @Description
170         This function reads the TMR2 register value and return it.
171
172     @Preconditions
173         Initialize the TMR2 before calling this function.
174
175     @Param
176         None
177
178     @Returns
179         This function returns the current value of TMR2 register.
180
181     @Example
182         <code>
183             // Initialize TMR2 module
184
185             // Start TMR2
186             TMR2_StartTimer();
187
188             // Read the current value of TMR2
189             if(0 == TMR2_ReadTimer())
190             {
191                 // Do something else...
192
193                 // Reload the TMR value
194                 TMR2_Reload();

```

```

195     }
196     </code>
197 */
198 uint8_t TMR2_ReadTimer(void);
199
200 /**
201  * @Summary
202  *   Writes the TMR2 register.
203
204  * @Description
205  *   This function writes the TMR2 register.
206  *   This function must be called after the initialization of TMR2.
207
208  * @Preconditions
209  *   Initialize the TMR2 before calling this function.
210
211  * @Param
212  *   timerVal - Value to write into TMR2 register.
213
214  * @Returns
215  *   None
216
217  * @Example
218  * <code>
219  * #define PERIOD 0x80
220  * #define ZERO    0x00
221
222  * while(1)
223  {
224      // Read the TMR2 register
225      if(ZERO == TMR2_ReadTimer())
226      {
227          // Do something else...
228
229          // Write the TMR2 register
230          TMR2_WriteTimer(PERIOD);
231      }
232
233      // Do something else...
234  }
235 </code>
236 */
237 void TMR2_WriteTimer(uint8_t timerVal);
238
239 /**
240  * @Summary
241  *   Load value to Period Register.
242
243  * @Description
244  *   This function writes the value to PR2 register.
245  *   This function must be called after the initialization of TMR2.
246
247  * @Preconditions
248  *   Initialize the TMR2 before calling this function.
249
250  * @Param
251  *   periodVal - Value to load into TMR2 register.
252
253  * @Returns
254  *   None
255
256  * @Example
257  * <code>
258  * #define PERIOD1 0x80
259  * #define PERIOD2 0x40

```

```

260     #define ZERO      0x00
261
262     while(1)
263     {
264         // Read the TMR2 register
265         if(ZERO == TMR2_ReadTimer())
266         {
267             // Do something else...
268
269             if(flag)
270             {
271                 flag = 0;
272
273                 // Load Period 1 value
274                 TMR2_LoadPeriodRegister(PERIOD1);
275             }
276             else
277             {
278                 flag = 1;
279
280                 // Load Period 2 value
281                 TMR2_LoadPeriodRegister(PERIOD2);
282             }
283         }
284
285         // Do something else...
286     }
287     </code>
288 */
289 void TMR2_LoadPeriodRegister(uint8_t periodVal);
290
291 /**
292  * @Summary
293  * Boolean routine to poll or to check for the match flag on the fly.
294
295  * @Description
296  * This function is called to check for the timer match flag.
297  * This function is used in timer polling method.
298
299  * @Preconditions
300  * Initialize the TMR2 module before calling this routine.
301
302  * @Param
303  * None
304
305  * @Returns
306  * true - timer match has occurred.
307  * false - timer match has not occurred.
308
309  * @Example
310  * <code>
311  * while(1)
312  * {
313  *     // check the match flag
314  *     if(TMR2_HasOverflowOccured())
315  *     {
316  *         // Do something else...
317
318  *         // Reload the TMR2 value
319  *         TMR2_Reload();
320     }
321   </code>
322 */
323 bool TMR2_HasOverflowOccured(void);

```

```
325 void TMR2_Wipe(void);
326 void us_delay250(uint16_t delay);
327
328 #ifdef __cplusplus // Provide C++ Compatibility
329 }
330
331#endif
332
333#endif // _TMR2_H
334 /**
335 * End of File
336 */
337
```

```

1  /**
2   * TMR3 Generated Driver File
3
4   * @Company
5   *   Microchip Technology Inc.
6
7   * @File Name
8   *   tmr3.c
9
10  * @Summary
11  *   This is the generated driver implementation file for the TMR3 driver using
12  *   MPLAB(c) Code Configurator
13
14  * @Description
15  *   This source file provides APIs for TMR3.
16  *   Generation Information :
17  *     Product Revision : MPLAB(c) Code Configurator - 4.15.5
18  *     Device          : PIC16F18345
19  *     Driver Version  : 2.00
20  *     The generated drivers are tested against the following:
21  *       Compiler      : XC8 1.35
22  *       MPLAB         : MPLAB X 3.40
23
24  */
25
26  /*
27  *   (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
28  *   software and any derivatives exclusively with Microchip products.
29
30  * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
31  * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
32  * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
33  * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
34  * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
35
36  * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
37  * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
38  * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
39  * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
40  * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
41  * ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
42  * THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
43
44  * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
45  * TERMS.
46
47  /**
48  *   Section: Included Files
49  */
50
51  #include <xc.h>
52  #include "tmr3.h"
53
54  /**
55  *   Section: Global Variables Definitions
56  */
57  volatile uint16_t timer3ReloadVal;
58  void (*TMR3_InterruptHandler)(void);
59
60  int RPM_rollover_flag = 0;
61  int rollover_count = 0;
62
63  /**
64  *   Section: TMR3 APIs
65  */

```

```

65
66  /*
67   *Note: ISR disabled currently. Enable in MCC to use hall sensor as LIDAR
68   * trigger.
69   */
70
71 void TMR3_Initialize(void)
72 {
73     //Set the Timer to the options selected in the GUI
74
75     //T3CKPS 1:1; T3SOSC T3CKI_enabled; T3SYNC synchronize; TMR3CS FOSC/4;
76     TMR3ON off;
77     T3CON = 0x00;
78
79     //T3GSS T3G_pin; TMR3GE disabled; T3GTM disabled; T3GPOL low; T3GGO_nDONE
80     //done; T3GSPM disabled;
81     T3GCON = 0x00;
82
83     //TMR3H 0;
84     TMR3H = 0x00;
85
86     //TMR3L 0;
87     TMR3L = 0x00;
88
89     // Load the TMR value to reload variable
90     timer3ReloadVal=(TMR3H << 8) | TMR3L;
91
92     // Clearing IF flag before enabling the interrupt.
93     PIR3bits.TMR3IF = 0;
94
95     // Enabling TMR3 interrupt.
96     PIE3bits.TMR3IE = 1;
97
98     // Set Default Interrupt Handler
99     TMR3_SetInterruptHandler(TMR3_DefaultInterruptHandler);
100
101    // Start TMR3
102    TMR3_StartTimer();
103}
104
105void TMR3_StartTimer(void)
106{
107    // Start the Timer by writing to TMRxON bit
108    T3CONbits.TMR3ON = 1;
109}
110
111void TMR3_StopTimer(void)
112{
113    // Stop the Timer by writing to TMRxON bit
114    T3CONbits.TMR3ON = 0;
115}
116
117uint16_t TMR3_ReadTimer(void)
118{
119    uint16_t readVal;
120
121    readVal = (TMR3H << 8) | TMR3L;
122
123    RPM_rollover_flag = 1; //we can start acknowledging rollover counts now
124
125    return readVal;
126}
127
128void TMR3_WriteTimer(uint16_t timerVal)

```

```

127    {
128        if (T3CONbits.T3SYNC == 1)
129        {
130            // Stop the Timer by writing to TMRxON bit
131            T3CONbits.TMR3ON = 0;
132
133            // Write to the Timer3 register
134            TMR3H = (timerVal >> 8);
135            TMR3L = timerVal;
136
137            // Start the Timer after writing to the register
138            T3CONbits.TMR3ON =1;
139        }
140        else
141        {
142            // Write to the Timer3 register
143            TMR3H = (timerVal >> 8);
144            TMR3L = timerVal;
145        }
146    }
147
148 void TMR3_Reload(void)
149 {
150     //Write to the Timer3 register
151     TMR3H = (timer3ReloadVal >> 8);
152     TMR3L = timer3ReloadVal;
153 }
154
155 void TMR3_StartSinglePulseAcquisition(void)
156 {
157     T3GCONbits.T3GGO_nDONE = 1;
158 }
159
160 uint8_t TMR3_CheckGateValueStatus(void)
161 {
162     return (T3GCONbits.T3GVAL);
163 }
164
165 void TMR3_ISR(void)
166 {
167
168     // Clear the TMR3 interrupt flag
169     PIR3bits.TMR3IF = 0;
170
171     if(RPM_rollover_flag == 1){
172         rollover_count++;
173     }
174
175     TMR3H = (timer3ReloadVal >> 8);
176     TMR3L = timer3ReloadVal;
177
178     if(TMR3_InterruptHandler)
179     {
180         TMR3_InterruptHandler();
181     }
182 }
183
184
185 void TMR3_SetInterruptHandler(void* InterruptHandler){
186     TMR3_InterruptHandler = InterruptHandler;
187 }
188
189 void TMR3_DefaultInterruptHandler(void){
190     // add your TMR3 interrupt custom code
191     // or set custom function using TMR3_SetInterruptHandler()

```

```
192 }
193
194 int read_rollover_count(void){
195     return rollover_count;
196 }
197
198 void reset_rollover_count(void){
199     rollover_count = 0;
200 }
201
202
203 /**
204  * End of File
205 */
```

```

1  /**
2   * TMR3 Generated Driver API Header File
3
4   * @Company
5   *   Microchip Technology Inc.
6
7   * @File Name
8   *   tmr3.h
9
10  * @Summary
11  *   This is the generated header file for the TMR3 driver using MPLAB(c) Code
12  *   Configurator
13  * @Description
14  *   This header file provides APIs for driver for TMR3.
15  *   Generation Information :
16  *     Product Revision : MPLAB(c) Code Configurator - 4.15.5
17  *     Device          : PIC16F18345
18  *     Driver Version  : 2.00
19  *     The generated drivers are tested against the following:
20  *       Compiler      : XC8 1.35
21  *       MPLAB         : MPLAB X 3.40
22  */
23
24  /*
25  *   (c) 2016 Microchip Technology Inc. and its subsidiaries. You may use this
26  *   software and any derivatives exclusively with Microchip products.
27
28  * THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
29  * EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
30  * WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
31  * PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
32  * WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
33
34  * IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
35  * INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
36  * WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
37  * BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
38  * FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
39  * ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
40  * THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
41
42  * MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
43  * TERMS.
44 */
45
46 #ifndef _TMR3_H
47 #define _TMR3_H
48
49 /**
50  *   Section: Included Files
51 */
52
53 #include <stdbool.h>
54 #include <stdint.h>
55
56 #ifdef __cplusplus // Provide C++ Compatibility
57
58     extern "C" {
59
60 #endif
61
62 /**
63  *   Section: TMR3 APIs

```

```

65  */
66 /**
67  * @Summary
68  *     Initializes the TMR3
69
70  * @Description
71  *     This routine initializes the TMR3.
72  *     This routine must be called before any other TMR3 routine is called.
73  *     This routine should only be called once during system initialization.
74
75  * @Preconditions
76  *     None
77
78  * @Param
79  *     None
80
81  * @Returns
82  *     None
83
84  * @Comment
85
86
87  * @Example
88  *     <code>
89  *         main()
90  *     {
91  *         // Initialize TMR3 module
92  *         TMR3_Initialize();
93
94         // Do something else...
95     }
96     </code>
97 */
98 void TMR3_Initialize(void);
99
100 /**
101 * @Summary
102 *     This function starts the TMR3.
103
104 * @Description
105 *     This function starts the TMR3 operation.
106 *     This function must be called after the initialization of TMR3.
107
108 * @Preconditions
109 *     Initialize the TMR3 before calling this function.
110
111 * @Param
112 *     None
113
114 * @Returns
115 *     None
116
117 * @Example
118 *     <code>
119 *         // Initialize TMR3 module
120
121         // Start TMR3
122         TMR3_StartTimer();
123
124         // Do something else...
125     </code>
126 */
127 void TMR3_StartTimer(void);
128
129

```

```

130  /**
131   * @Summary
132   *   This function stops the TMR3.
133
134   * @Description
135   *   This function stops the TMR3 operation.
136   *   This function must be called after the start of TMR3.
137
138   * @Preconditions
139   *   Initialize the TMR3 before calling this function.
140
141   * @Param
142   *   None
143
144   * @Returns
145   *   None
146
147   * @Example
148   *   <code>
149   *     // Initialize TMR3 module
150
151   *     // Start TMR3
152   *     TMR3_StartTimer();
153
154   *     // Do something else...
155
156   *     // Stop TMR3;
157   *     TMR3_StopTimer();
158   *   </code>
159 */
160 void TMR3_StopTimer(void);
161
162 /**
163  * @Summary
164  *   Reads the TMR3 register.
165
166  * @Description
167  *   This function reads the TMR3 register value and return it.
168
169  * @Preconditions
170  *   Initialize the TMR3 before calling this function.
171
172  * @Param
173  *   None
174
175  * @Returns
176  *   This function returns the current value of TMR3 register.
177
178  * @Example
179  *   <code>
180  *     // Initialize TMR3 module
181
182  *     // Start TMR3
183  *     TMR3_StartTimer();
184
185  *     // Read the current value of TMR3
186  *     if(0 == TMR3_ReadTimer())
187  *     {
188  *       // Do something else...
189
190  *       // Reload the TMR value
191  *       TMR3_Reload();
192  *     }
193  *   </code>
194 */

```

```

195     uint16_t TMR3_ReadTimer(void);
196
197 /**
198  * @Summary
199  *   Writes the TMR3 register.
200
201  * @Description
202  *   This function writes the TMR3 register.
203  *   This function must be called after the initialization of TMR3.
204
205  * @Preconditions
206  *   Initialize the TMR3 before calling this function.
207
208  * @Param
209  *   timerVal - Value to write into TMR3 register.
210
211  * @Returns
212  *   None
213
214  * @Example
215  * <code>
216  * #define PERIOD 0x80
217  * #define ZERO    0x00
218
219  * while(1)
220  {
221      // Read the TMR3 register
222      if(ZERO == TMR3_ReadTimer())
223      {
224          // Do something else...
225
226          // Write the TMR3 register
227          TMR3_WriteTimer(PERIOD);
228      }
229
230      // Do something else...
231  }
232  </code>
233 */
234 void TMR3_WriteTimer(uint16_t timerVal);
235
236 /**
237  * @Summary
238  *   Reload the TMR3 register.
239
240  * @Description
241  *   This function reloads the TMR3 register.
242  *   This function must be called to write initial value into TMR3 register.
243
244  * @Preconditions
245  *   Initialize the TMR3 before calling this function.
246
247  * @Param
248  *   None
249
250  * @Returns
251  *   None
252
253  * @Example
254  * <code>
255  * while(1)
256  {
257      if(TMR3IF)
258      {
259          // Do something else...

```

```

260             // clear the TMR3 interrupt flag
261             TMR3IF = 0;
262
263             // Reload the initial value of TMR3
264             TMR3_Reload();
265         }
266     }
267 
```

</code>

```

268 */
269 void TMR3_Reload(void);
270
271 /**
272  * @Summary
273  * Starts the single pulse acquisition in TMR3 gate operation.
274  *
275  * @Description
276  * This function starts the single pulse acquisition in TMR3 gate operation.
277  * This function must be used when the TMR3 gate is enabled.
278  *
279  * @Preconditions
280  * Initialize the TMR3 with gate enable before calling this function.
281  *
282  * @Param
283  * None
284  *
285  * @Returns
286  * None
287  *
288  * @Example
289  * <code>
290  *     uint16_t xVal;
291  *     uint16_t yVal;
292  *
293  *     // enable TMR3 singlepulse mode
294  *     TMR3_StartSinglePulseAcquisition();
295  *
296  *     // check TMR3 gate status
297  *     if(TMR3_CheckGateValueStatus() == 0)
298  *         xVal = TMR3_ReadTimer();
299  *
300  *     // wait until gate interrupt occurred
301  *     while(TMR3GIF == 0)
302  *     {
303  *     }
304  *
305  *     yVal = TMR3_ReadTimer();
306  * </code>
307 */
308 void TMR3_StartSinglePulseAcquisition(void);
309
310 /**
311  * @Summary
312  * Check the current state of Timer1 gate.
313  *
314  * @Description
315  * This function reads the TMR3 gate value and return it.
316  * This function must be used when the TMR3 gate is enabled.
317  *
318  * @Preconditions
319  * Initialize the TMR3 with gate enable before calling this function.
320  *
321  * @Param
322  * None
323  *
324

```

```

325     @Returns
326     None
327
328     @Example
329     <code>
330     uint16_t xVal;
331     uint16_t yVal;
332
333     // enable TMR3 singlepulse mode
334     TMR3_StartSinglePulseAcquisition();
335
336     // check TMR3 gate status
337     if(TMR3_CheckGateValueStatus() == 0)
338         xVal = TMR3_ReadTimer();
339
340     // wait until gate interrupt occurred
341     while(TMR3IF == 0)
342     {
343     }
344
345     yVal = TMR3_ReadTimer();
346     </code>
347 */
348 uint8_t TMR3_CheckGateValueStatus(void);
349
350 /**
351     @Summary
352     Timer Interrupt Service Routine
353
354     @Description
355         Timer Interrupt Service Routine is called by the Interrupt Manager.
356
357     @Preconditions
358         Initialize the TMR3 module with interrupt before calling this ISR.
359
360     @Param
361     None
362
363     @Returns
364     None
365 */
366 void TMR3_ISR(void);
367
368 /**
369     @Summary
370     Set Timer Interrupt Handler
371
372     @Description
373         This sets the function to be called during the ISR
374
375     @Preconditions
376         Initialize the TMR3 module with interrupt before calling this.
377
378     @Param
379         Address of function to be set
380
381     @Returns
382     None
383 */
384 void TMR3_SetInterruptHandler(void *InterruptHandler);
385
386 /**
387     @Summary
388     Timer Interrupt Handler
389

```

```

390     @Description
391         This is a function pointer to the function that will be called during the ISR
392
393     @Preconditions
394         Initialize the TMR3 module with interrupt before calling this isr.
395
396     @Param
397         None
398
399     @Returns
400         None
401     */
402     extern void (*TMR3_InterruptHandler)(void);
403
404 /**
405     @Summary
406         Default Timer Interrupt Handler
407
408     @Description
409         This is the default Interrupt Handler function
410
411     @Preconditions
412         Initialize the TMR3 module with interrupt before calling this isr.
413
414     @Param
415         None
416
417     @Returns
418         None
419     */
420     void TMR3_DefaultInterruptHandler(void);
421
422     int read_rollover_count(void);
423
424     void reset_rollover_count(void);
425
426 #ifdef __cplusplus // Provide C++ Compatibility
427 }
428
429#endif
430
431#endif // _TMR3_H
432 /**
433 * End of File
434 */
435

```

3.5.0 Weapon Motor Control Board

The primary function of the weapon motor control board is to command and monitor the weapon motor for the combat robot. The weapon motor control board is the splitting point between two separate power domains. There are two separate batteries that power the robot, one for the logic and locomotion motor system, one for the high power weapon motor. The weapon motor control board provides segmentation that allows for isolated control, geographically remote from the main board and logic power domain, in an effort to provide a more stable control system. (ZDK)

3.5.1 Weapon Motor Control Board Hardware

Figure 53 shows a level 1 block diagram for how the weapon motor controller board will operate. Taking in a set of control signals from the MCB and feedback signals about motor characteristics, the localized microcontroller will control an IGBT to throttle current to the motor. (ZDK)

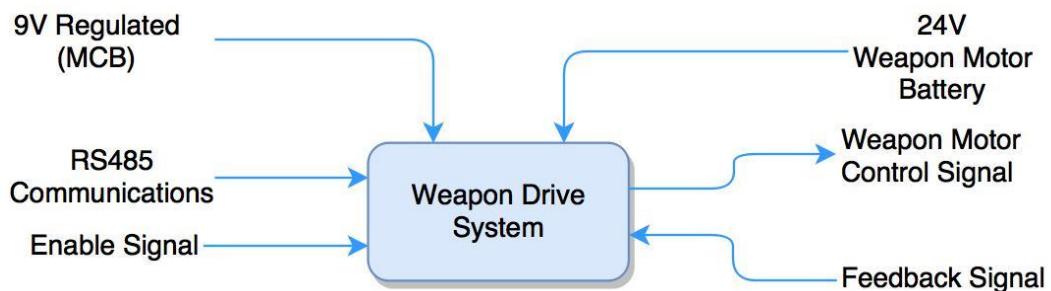


Figure 53: Level 1 Weapon Motor Controller Hardware Block Diagram

Module	Weapon Drive System
Designer	Zachary Kilburn
Inputs	Power: Regulated 9V DC (from MCB) Power: Unregulated 24V (from Weapon Motor Sub-battery) Signal: RS485 Communications (from MCB) Signal: Enable Signal (from MCB) Signal: Feedback Signal (Hall Effect Sensor Feedback)
Outputs	Signal: IGBT Control
Description	The Weapon Drive System takes inputs from the MCB indicating when to operate from the enable signal and how fast to operate from the RS485 communications bus. Using current feedback from a hall effect sensor, the motor control signal is generated. 9V DC powers the logic unit while 24V powers the switching unit.

Table 68: Level 1 Weapon Motor Control Hardware Microcontroller Module

Figure 54 shows a level 2 block diagram for the weapon motor controller board.

Expanding on level 1, the microcontroller will have TTL level communications, converted from RS485, from the main control board with information about how to throttle the weapon motor as well as a digital enable signal to enable or disable the motor drive system. A Hall Effect sensor will provide feedback about the current delivered to the motor and allow the detection of faults, and power used by the motor. (ZDK)

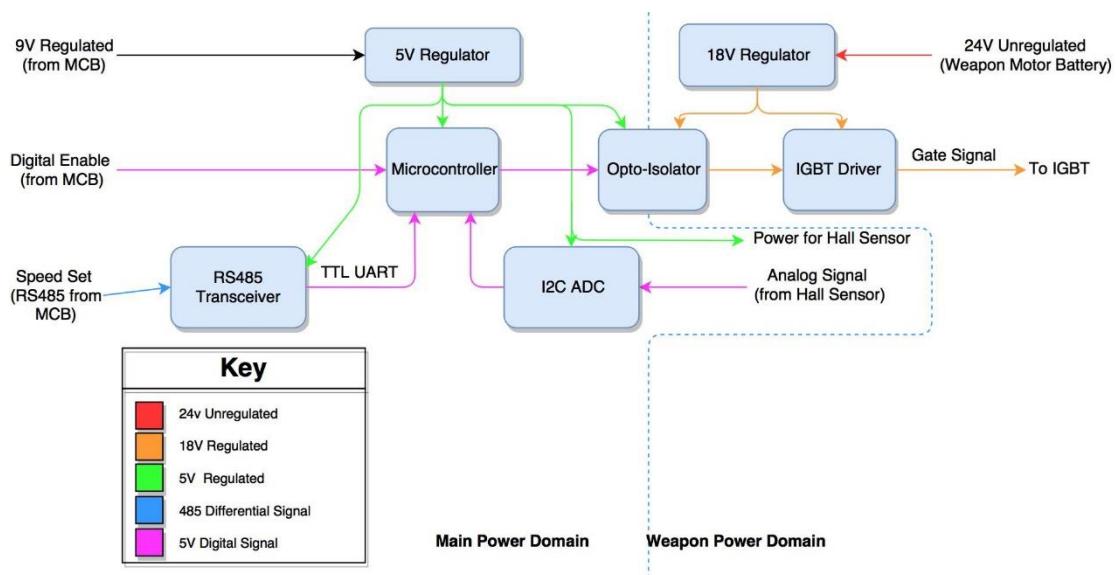


Figure 54: Level 2 Weapon Motor Control Hardware Block Diagram

Module	Microcontroller
Designer	Zachary Kilburn
Inputs	Signal: Digital Enable Signal Signal: TTL UART Signal: Analog Signal (from hall sensor) Power: 5V DC
Outputs	Signal: Gate drive signal (to opto-isolator)
Description	The microcontroller will produce the gate drive signal for the motor when the digital enable signal from the MCB enables the drive system. The TTL UART contains data sent from the MCB that will control the speed at which the microcontroller commands the motor. Hall sensor feedback allows for information about motor conditions and affects the gate drive signal control.

Table 69: Level 2 Weapon Motor Control Hardware Microcontroller Module

Module	RS485 Transceiver
Designer	Zachary Kilburn
Inputs	Signal: 485 Differential Signals from Main board
Outputs	Signal: TTL Level UART
Description	The RS485 transceiver converts 4-wire differential communications lines to 2 wire TTL signals for the microcontroller.

Table 70: Level 2 Weapon Motor Control Hardware RS485 Transceiver Module

Module	Opto-Isolator
Designer	Zachary Kilburn
Inputs	Signal: PWM signal from microcontroller Power: 5V DC Power: 18V DC
Outputs	Signal: PWM signal gate drive IC
Description	The opto-isolator provides isolation for the gate drive system. It translates the 5V logic signal from the microcontroller to the 18V gate drive IC, providing galvanic isolation from the noisy battery pack that the weapon motor is powered from.

Table 71: Level 2 Weapon Motor Control Hardware Opto-Isolator Module

Module	18V Regulator
Designer	Zachary Kilburn
Inputs	Power: 24V Unregulated Weapon Motor Sub-battery
Outputs	Power: 18V Regulated
Description	The 18V regulator takes the weapon battery power, from one of the series battery packs and converts it to 18V for the IGBT driver and the opto-isolater that converts the PWM signal to the isolated weapon motor power domain.

Table 72: Level 2 Weapon Motor Control Hardware 18V Regulator Module

Module	5V Regulator
Designer	Zachary Kilburn
Inputs	Power: 9V Regulated DC
Outputs	Power: 5V Regulated DC
Description	The 5V regulator provides logic level power for all components on the main power domain including: the microcontroller, hall effect sensor, opto-isolator and RS485 transceiver.

Table 73: Level 2 Weapon Motor Control Hardware 5V Regulator

Module	IGBT Driver
Designer	Zachary Kilburn
Inputs	Signal: PWM Signal from Opto-isolator Power: 18V Regulated DC
Outputs	Signal: PWM signal for IGBT
Description	The IGBT driver converts the opto-isolated PWM signal from the microcontroller to a higher voltage capable of driving the gate of the IGBT to saturation.

Table 74: Level 2 Weapon Motor Control Hardware IGBT Driver Module

Module	I2C ADC
Designer	Zachary Kilburn
Inputs	Signal: Analog Signal (from Hall Sensor) Power: 5V Regulated DC
Outputs	Signal: I2C Current Data
Description	The I2C ADC takes in the analog signal from the Hall Effect current transducer and converts it to a digital signal that is available to the microcontroller upon request.

Table 75: Level 2 Weapon Motor Control Hardware I2C ADC Module

3.5.2 Weapon Motor Control Board Software

The Weapon Motor Control Board software provides motor control to the weapon motor. It process the requests from the Main Control Board relating to the enabling and disabling the operation of the weapon motor via an enable signal. The Weapon Motor Board will constantly be requesting and receiving information from the RS485 Bus which is used to communicate to the Main Control Board. Once a message has been retrieved, the microcontroller will compare the received data from the bus to the current data of the weapon speed from the Sensor Measurements to compute the required Duty cycle for the PWM signal sent to the IGBT. Current feedback from the motor is used to detect faults in operations, weapon ready statuses and general control scheme for driving the PWM signal to the IGBT which will switch power to the motor. (HL)

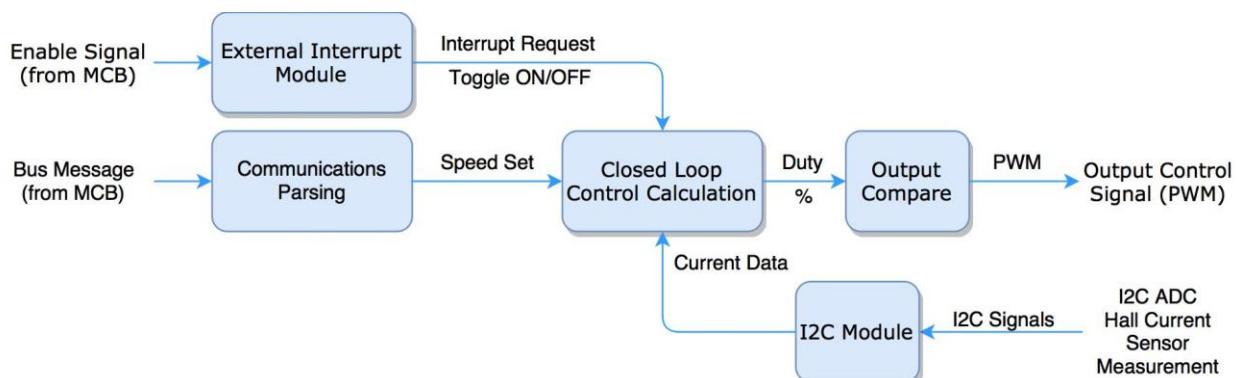


Figure 55: Level 1 Weapon Motor Control Board Software Block Diagram

Module	External Interrupt Module
Designer	Zachary Kilburn
Inputs	Signal: Enable Signal (from MCB)
Outputs	Signal/Event: Interrupt Request, Toggle ON/OFF
Description	The external interrupt module monitors the enable signal pin, looking for transitions of the logic signal. Upon a high->low or low->high transition, the module flags an interrupt for the main control thread to handle, which will operate as an immediate state change from the operating condition to idle and vice versa.

Table 76: Level 1 Weapon Motor Control Software External Interrupt Module

Module	Communications Parsing
Designer	Zachary Kilburn
Inputs	Signal: Bus Message
Outputs	Signal: Speed Set
Description	Communications parsing will occur within the microcontroller to identify messages on the bus that are designated for the Weapon Control Board. When a message is identified as addressed for the weapon control board, the speed set data will be extracted from the message and given to the closed loop control calculation for use.

Table 77: Level 1 Weapon Motor Control Software Communications Parsing

Module	Closed Loop Control Calculation
Designer	Zachary Kilburn
Inputs	Signal: Interrupt Request, Toggle ON/OFF Signal: Speed Set Signal: Current Data
Outputs	Signal: Duty (percentage)
Description	The closed loop control calculation takes a speed set point and current data from the motor operation and makes decisions for how much power to apply to the motor in the form a duty cycle (from 0-100%). The output can be switched off from the MCB by setting the Enable Signal low, triggering an interrupt from the external interrupt module, which will set the duty output to 0%.

Table 78: Level 1 Weapon Motor Control Software Closed Loop Control Calculation

Module	Output Compare Module
Designer	Zachary Kilburn
Inputs	Signal: Duty (%)
Outputs	Signal: PWM
Description	The output compare module is a module internal to the microcontroller which produces a PWM signal. The closed control loop calculation will provide a duty % to the output compare module. The PWM will be changed accordingly and sent to the driver IC.

Table 79: Level 1 Weapon Motor Control Software Output Compare Module

Module	I2C Module
Designer	Zachary Kilburn
Inputs	Signal: I2C Signals (SDA and SCL)
Outputs	Signal: Analog Measurement Data (Current Data)
Description	The I2C module will interface between the microcontroller and the I2C ADC unit. Upon requests by the microcontroller the I2C will gather data from the ADC and store it for use when calculating the speed control system.

Table 80: Level 1 Weapon Motor Control Software I2C Module

3.5.3 Weapon Motor Control Board Schematic

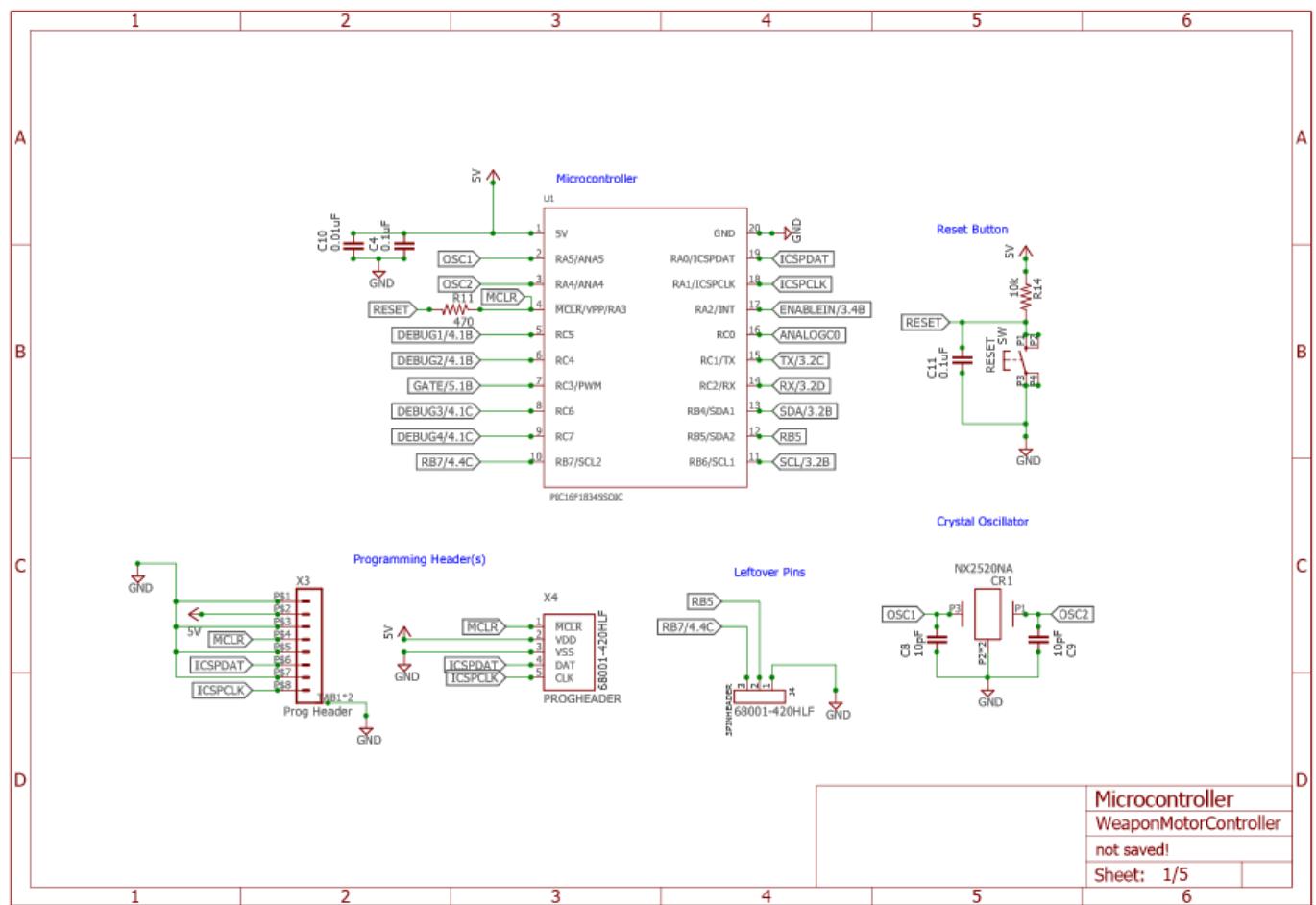


Figure 56: Weapon Motor Control Board Microcontroller Schematic

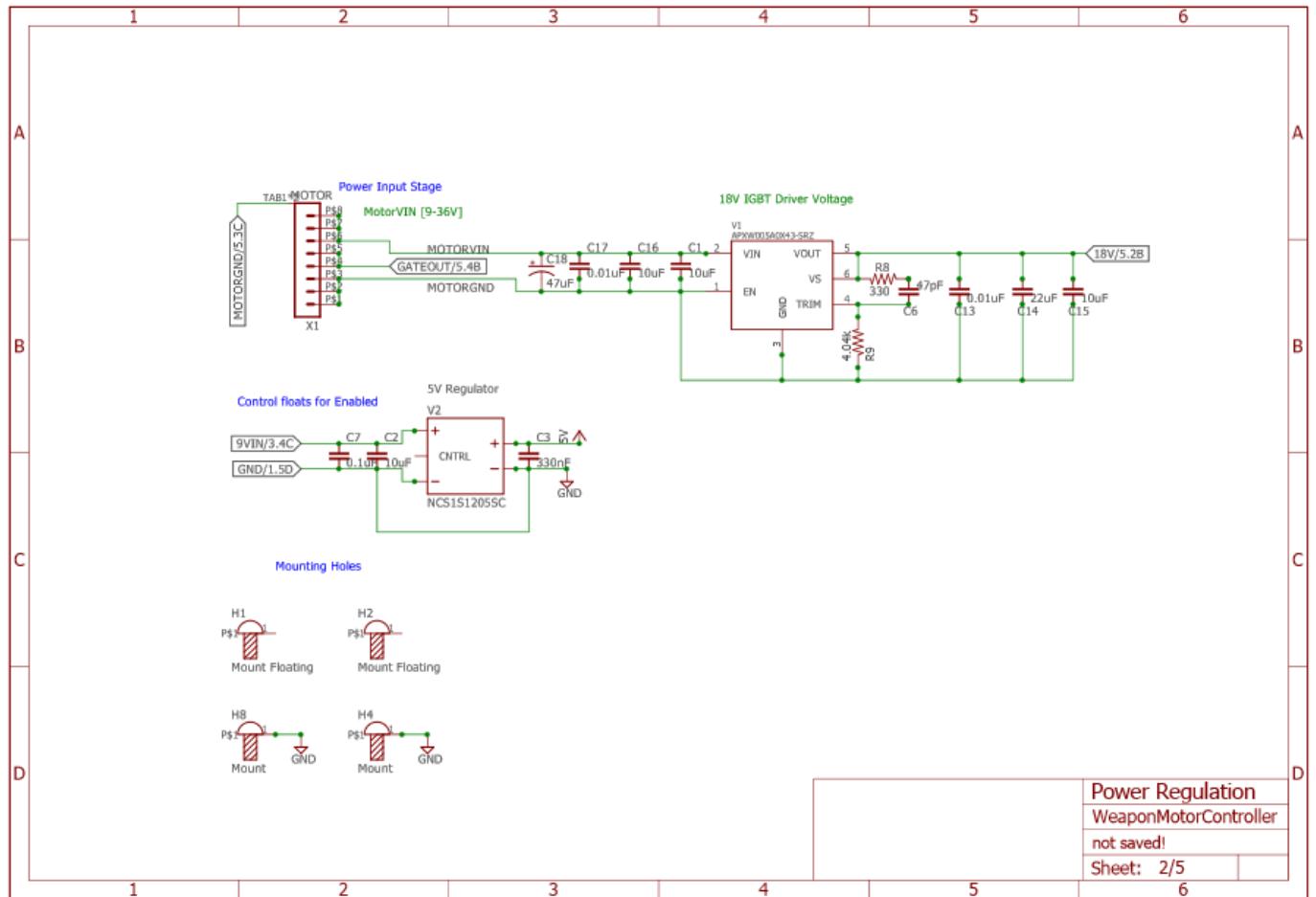


Figure 57: Weapon Motor Control Board Power Regulation Schematic

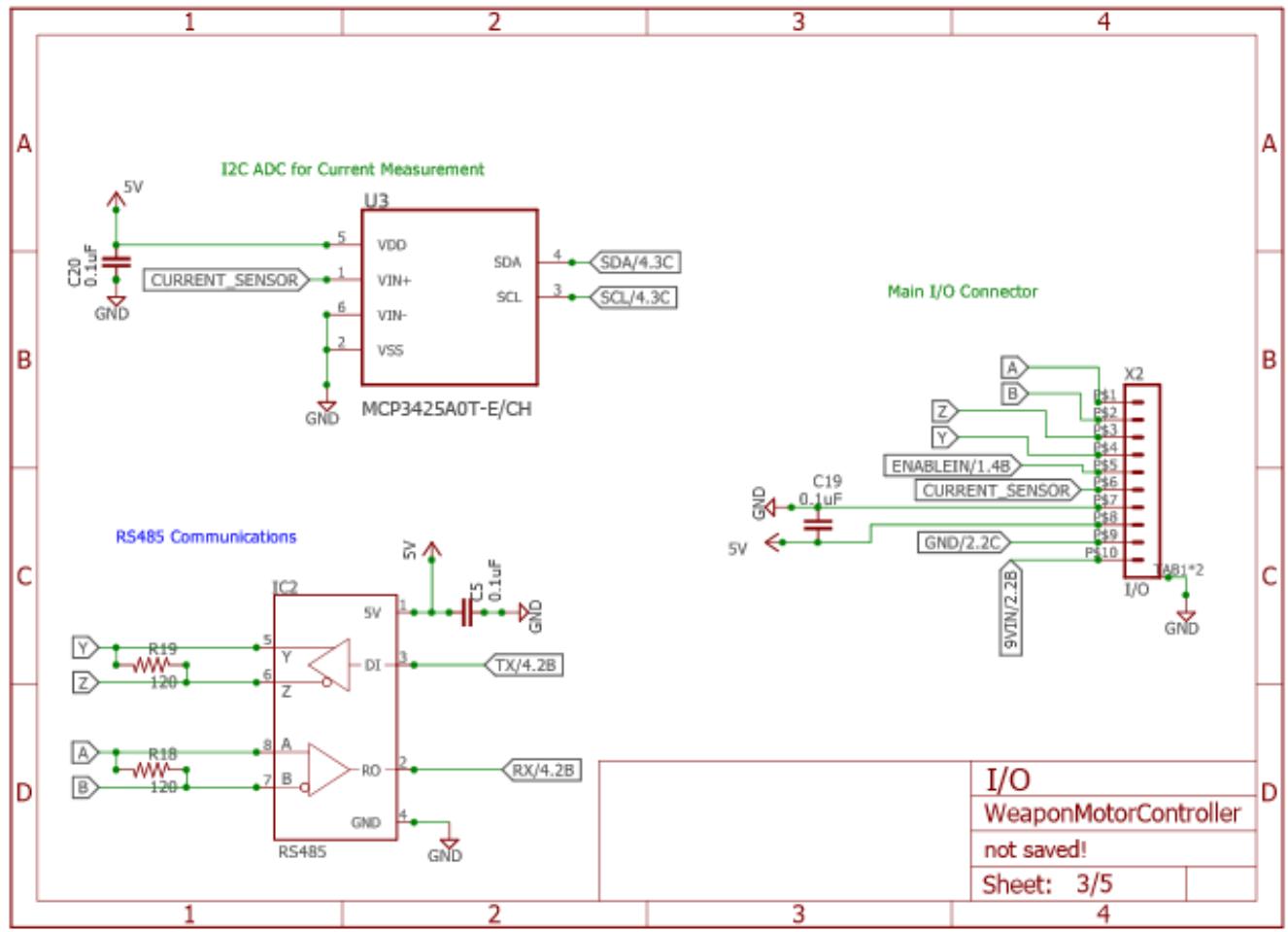


Figure 58: Weapon Motor Control Board I/O Schematic

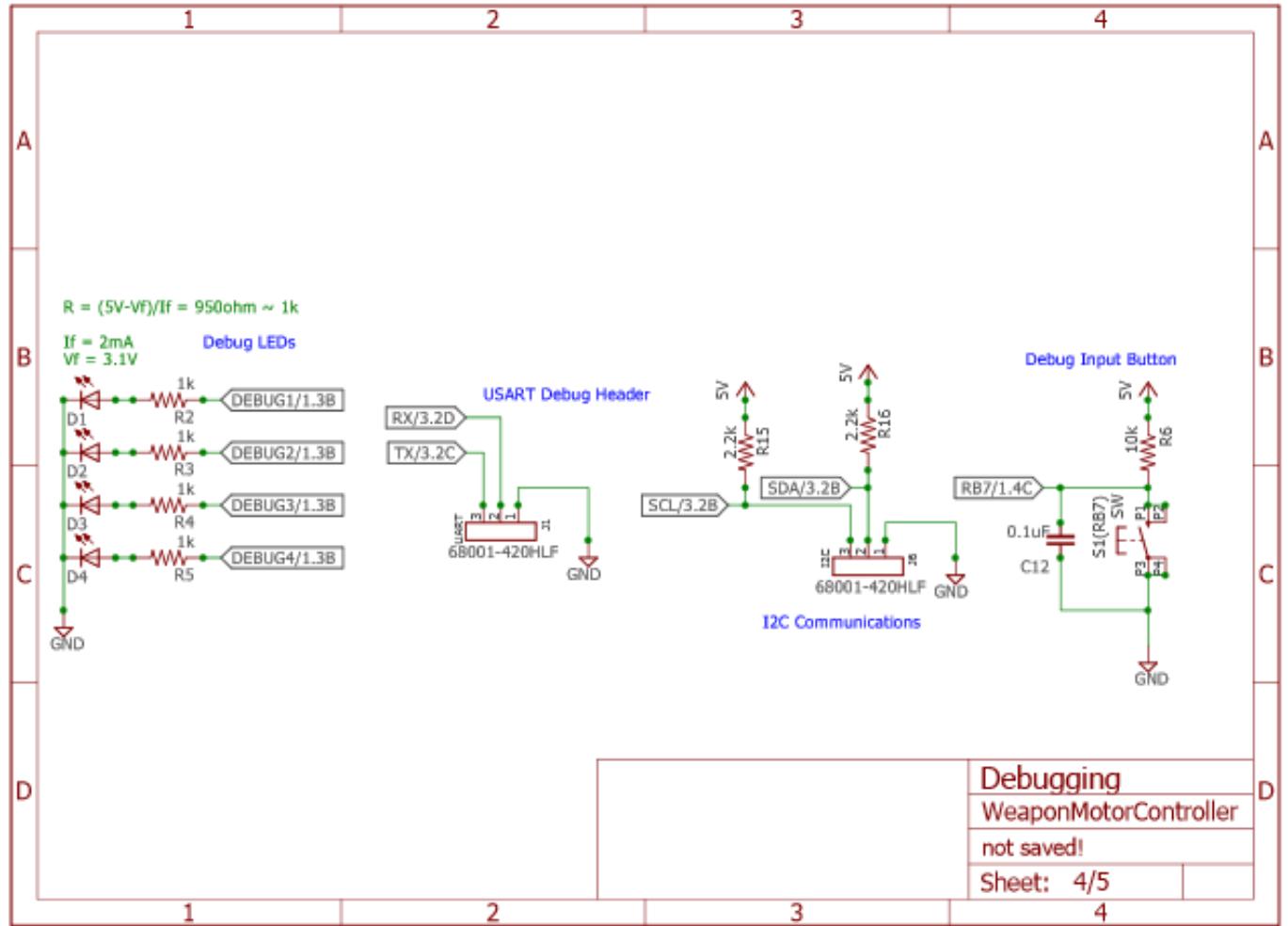


Figure 59: Weapon Motor Control Board Debugging Schematic

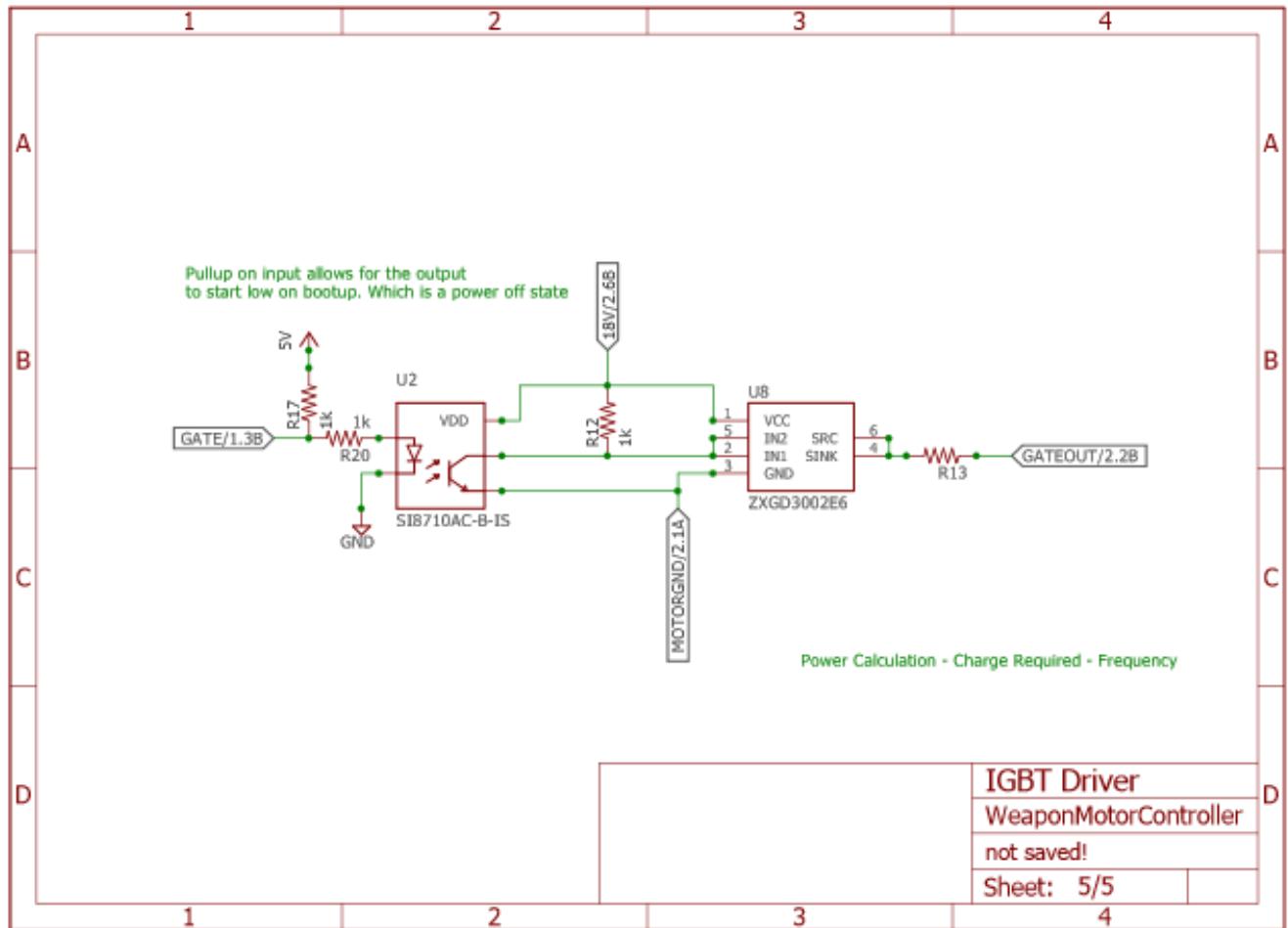


Figure 60: Weapon Motor Control Board IGBT Driver Schematic

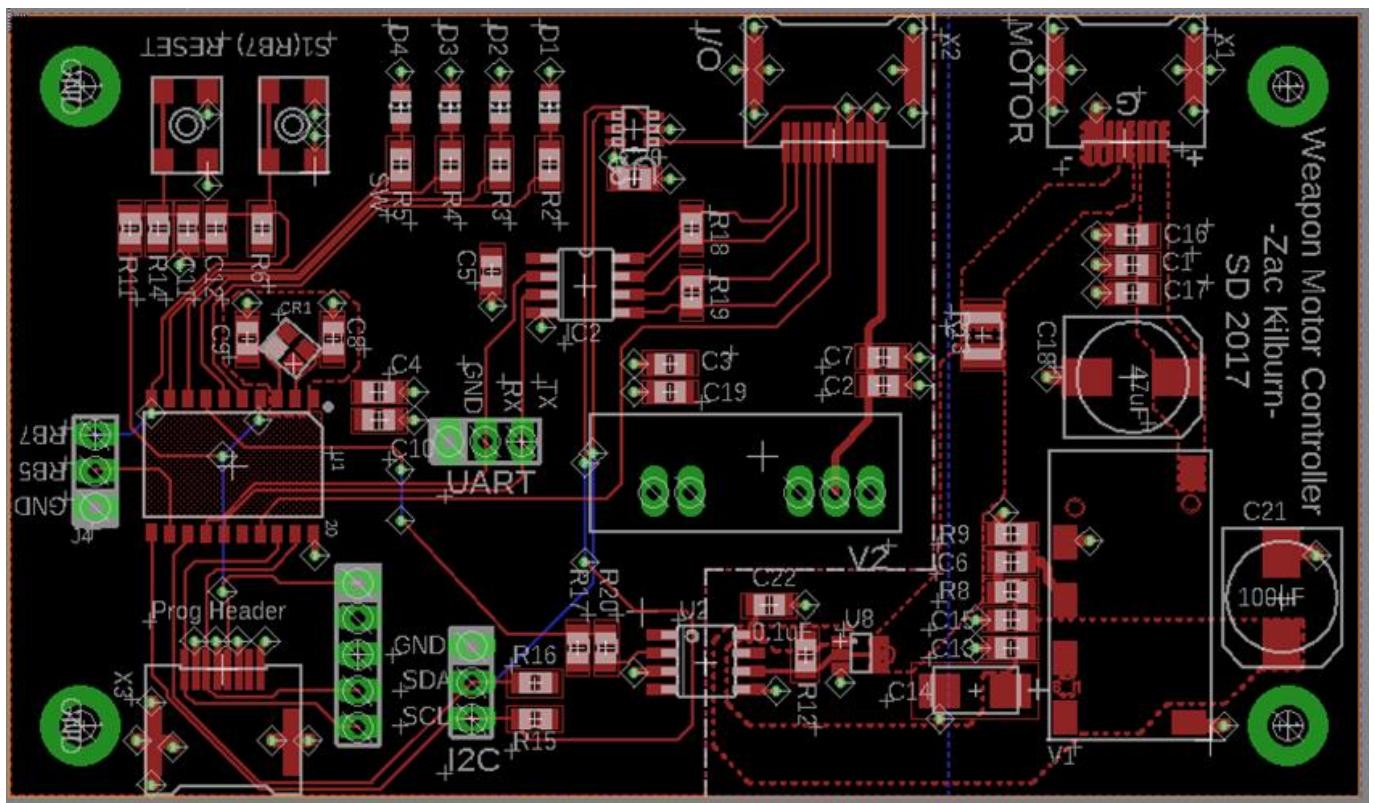


Figure 61: Weapon Motor Board PCB Layout

3.5.4 Weapon Motor Control Board Bill of Material

Refdes	Part Num.	Description
D1,D2,D3,D4	QBLP631-IB	LED BLUE CLEAR 0805 SMD
U1	PIC16F18345-E/SO	IC MCU 8BIT 14KB FLASH 20SOIC
CR1	NX2520SA-32.000000MHZ	CRYSTAL 32.0000MHZ 10PF SMD
J1,J4,J6,X4	68001-420HLF	CONN HEADER 20POS .100 STR TIN
RESET,S1	PTS645SM43SMTR92 LFS	SWITCH TACTILE SPST-NO 0.05A 12V
V1	APXW005A0X3-SRZ	DC/DC CONVERTER 3-18V 45W
V2	NCS1S1205SC	DC/DC CONVERTER 5V 1W
IC2	MAX490ESA+	IC TXRX RS485/RS422 8-SOIC
U2	SI8710AC-B-IS	DGTL ISO 3.75KV GEN PURP 8SOIC
U8	ZXGD3002E6TA	IC GATE DRVR IGBT/MOSFET SOT23-6
Current Sensor	HASS 200-S	SENSOR CURRENT HALL 200A AC/DC
IGBT	IXGN200N60B3	IGBT 300A 600V SOT-227B
X2	5031481090	CONN RCPT R/A DUAL 10CKT BEIGE
X2_CONN	5031491000	CONN PLUG DUAL 10CKT BEIGE
X1,X3	503148-0890	Headers & Wire Housings 1.5WB DUAL R/A EMBS TAPE PKG 8P
X1_CONN, X3_CONN	503149-0800	Headers & Wire Housings 1.5WB DUAL PLUG HSG 8CKT BE
R2,R3,R4,R5,R12, R17, R20	ASC0805-1KF1	RES 1 KOHM 1% 1/8W 0805
R6, R14	ASC0805-10KF1	RES 10 KOHM 1% 1/8W 0805
R8	TNPW0805330RDEEA	RES 330 OHM 0.5% 1/5W 0805
R9	TNPW08054K02BEEN	RES 4.02K OHM 0.1% 1/5W 0805
R11	RC0402FR-07470RL	RES SMD 470 OHM 1% 1/16W 0402
R13	RMCF1210FT2R00	RES SMD 2 OHM 1% 1/2W 1210
R15, R16	RC0805FR-072K2L	RES SMD 2.2K OHM 1% 0.4W 0805
R18,R19	ESR10EZPF1200	RES SMD 120 OHM 1% 0.4W 0805
C8,C9	08051A200FAT2A	CAP CER 20PF 100V NP0 0805
C6	885012007032	CAP CER 47PF 25V C0G/NP0 0805
C3	08055C333KAT2A	CAP CER 0.033UF 50V X7R 0805
C10,C17,C13	CC0805KRX7R9BB103	CAP CER 10000PF 50V X7R 0805
C4,C5,C7,C11,C12,C19,C20,C22	C0805C104K5RACTU	CAP CER 0.1UF 50V X7R 0805
C1,C2,C16,C15	GRM21BR61E106KA73L	CAP CER 10UF 25V X5R 0805
C14	T491C226K025AT	CAP TANT 22UF 25V 10% 2312
C18	EEE-FK1H470P	CAP ALUM 47UF 20% 50V SMD
C21	EEE-1EA101UP	CAP ALUM 100UF 20% 25V SMD
IGBT Connector Lugs	192210452	CONN RING FLAT 4AWG #10 CRIMP
IGBT Mounting Hardware	91280A132	Medium-Strength Class 8.8 Steel Hex Head Screw
IGBT Mounting Hardware	90591A255	Medium-Strength, Class 8, M4 x 0.7 mm Thread
Terminal Connectors (IGBT/ Motor)	192210422	CONN RING FLAT 1/0AWG #1/4 CRIMP
Wire		1/0 - 10ft Black/10ft Red
Battery Parallel Connector		InstallGear 0/2/4 AWG Gauge Power Distribution Block 1/0 Gauge In to (4) 4/8 Gauge Out

Table 81: Weapon Motor Control Board Bill Of Materials

3.5.5 Weapon Motor Control Board Implementation Revisions

One implementation that was revised was by adding a feed-back diode for the IGBT Driver, due the large current feeding the motor there was needed of a feedback diode that was connected parallel to the motor to allow the current slowly dissipate my looping through the motor and the diode when the driver was switched off. The allowed protection for the board or any other exterior components from being damaged by and feed back currents. One revision that was needed was to implementation of a pull-down transistor between the communication of the Weapon Motor Board and Main Control Board, in Figure 60: Weapon Motor Control Board IGBT Driver Schematic the current was being pulled up where when the weapon was in idle mode it would turn on to maximum speed, but with the pull-down transistor the board was able to inverse that occurrence allowing the weapon to be fully controlled. (HL)

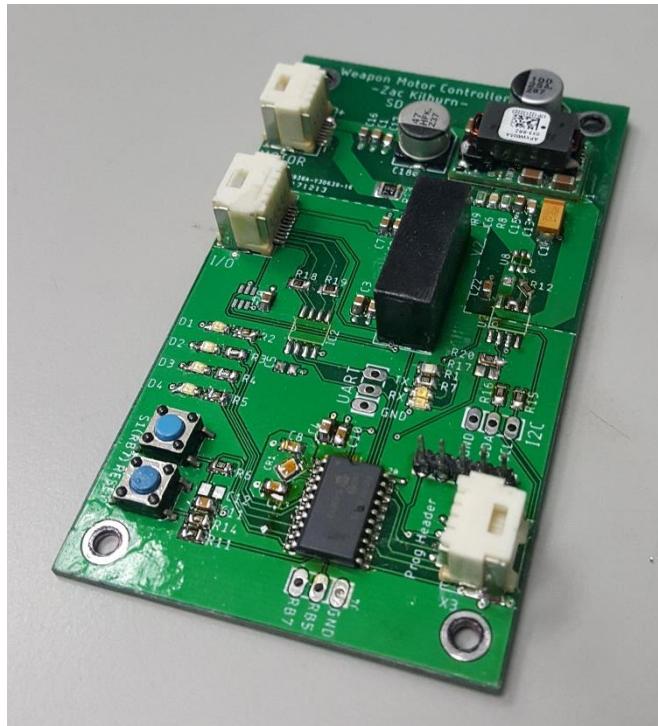


Figure 62: Completed WMCB

3.5.6 Weapon Motor Control Board Code

main.c

```
#define _XTAL_FREQ 20000000
#define SPEED_SETTING 1

#include "communications.h"
#include "FastTransfer.h"
#include "LED.h"
#include "MotorDrive.h"
#include "Timers.h"

#include <stdio.h>
#include <time.h>
#include <xc.h>
#include "config.h"
#include "mcc_generated_files/mcc.h"

/*
    Main application
*/

//-----
//--RUN ONCE---
//  TURN EVERYTHING ON
//  initMotorDrive();
//  initCommunications()
//--RUN ONCE---

//while(1)
//  updateComms();
//  REQUEST DATA FROM MCU
//  getReceiveArrayComms();
//
//  RECEIVE/PROCESS DATA FROM MCU
//  receiveArray();
//  getSpeedSetting();
//
//  //  ENABLE OR DIASBLE WEAPON BASED ON RECEIVED DATA
//  enableMotorDrive();
//  //      -or- (if/else statement?)
//  disableMotorDrive();
//
//  //  UPDATE WMB IF IT SHOULD STILL BE RUNNING OR NOT. IF NOT disableMotorDrive()
//  updateMotorDrive();
//  updateComms();
//  disableMotorDrive();
```

```

//-----
//-----  

// Main Code  

int RX_Buffer[50]; //Data Array  

bool motorDriveEnabled=false; //Start Motor Off  

int previousSpeedSetting=0; //Previous Speed Setting  

int receiveArray[15]; //Recieve Array  

bool tmr2_expired = true; //TMR2  

bool tmr0_expired = false; //TMR0  

void main(void)
{
    // initialize the device
    SYSTEM_Initialize();  

    // When using interrupts, you need to set the Global and Peripheral Interrupt Enable bits
    // Use the following macros to:  

    // Enable the Global Interrupts
    INTERRUPT_GlobalInterruptEnable();  

    // Enable the Peripheral Interrupts
    INTERRUPT_PeripheralInterruptEnable();  

    // Disable the Global Interrupts
    //INTERRUPT_GlobalInterruptDisable();  

    // Disable the Peripheral Interrupts
    //INTERRUPT_PeripheralInterruptDisable();  

    begin(RX_Buffer, sizeof(RX_Buffer), 3, false, EUSART_Write, EUSART_Read, EUSART_Bytes_Available,
EUSART_Peek);  

    initMotorDrive(); //Turn on Motor
    initCommunications(); //Start Comms
    int previousSpeedSetting; //Speed Setting
    int NewSpeed; //New Speed  

    // PWM/ Duty Ratio Stuff
    while(1)
    {
        // updateComms(); //Turn On Communications or Update Based on TMR2 =
25us
        getReceiveArrayComms(); //Get Array Comms
        NewSpeed=RX_Buffer[1]*10; //Set New Speed
        motorDriveEnabled=true; //Turn on Motor
    }
}

```

```

//If Timer0 is hit the speed setting will be ramped up or down
if(tmr0_expired=true)           //If Timer0 has reached 1us then enter loop
{
    if(previousSpeedSetting < NewSpeed)      //If Previous speed is less than Get speed
    {
        previousSpeedSetting++;          //Increase PWM
        PWM5_LoadDutyValue(previousSpeedSetting); //Set PWM based on speed setting
        tmr0_expired = false;           //Reset Timer TMRO
    }
    else if (previousSpeedSetting > NewSpeed) //If Previous speed is more than Get speed
    {
        previousSpeedSetting--;          //Decrease PWM
        PWM5_LoadDutyValue(previousSpeedSetting); //Set PWM based on speed setting
        tmr0_expired = false;
    }
    else                                //If Previous speed is same as Get speed
    {
        tmr0_expired = false;
    }

    if(previousSpeedSetting == 0 && NewSpeed == 0) //If Speed setting is 0
    {
        disableMotorDrive();           //Turn off Motor
        //break;                      //Exit while loop
    }
}

//-----
//Turn on LED D3 if Motor is supposed to turn on and Turn on LED D4 if motor is suppose to turn off

int x=1;
if (receiveData())
{
    setLED(3,0);
    //TRISBbits.TRISB7=1;
    if (RX_Buffer[x] > 0)
    {
        PORTBbits.RB7=1;           //Receiving Data
        toggleLED(1);
        setLED(2,0);
    }
    else
    {
        PORTBbits.RB6=1;
        toggleLED(2);           //Receiving Data No different
        setLED(1,0);
    }
}

```

```

        }
    else
    {
        PORTBbits.RB6=1;
        setLED(3,1); //Board Not Receive anything
    }
    setLED(4,1); //Board is Alive
//-----
}

}

// //Speed up spin of motor
// for(int i = 0; i <1023;i++)
// {
//     PWM5_LoadDutyValue(i);
//     updateMotorDrive();
//     __delay_ms(5);
// }
//
// //Slow down speed of motor
// for(int i=1023;i>0;i--)
// {
//     PWM5_LoadDutyValue(i);
//     updateMotorDrive();
//     __delay_ms(5);
// }

```

Communications.c

```

#include "communications.h"
#include "Timers.h"
#include "mcc_generated_files/tmr2.h"
#include "mcc_generated_files/tmr0.h"
#include <xc.h>

#define SPEED_SETTING 1

int receiveArray[15];
timer_t commsTimer;
extern bool tmr2_expired;

void initCommunications(void)
{
    setTimerInterval(&commsTimer, 5);

```

```

}

void updateComms(void)
{
    if(tmr2_expired)
    {
        tmr2_expired = false;
        if(receiveData())
        {

        }
    }
}

uint16_t getSpeedSetting()
{
    return receiveArray[SPEED_SETTING];
}

int * getReceiveArrayComms(void)
{
    return receiveArray;
}

FastTransfer.c

```

```

#include <xc.h>
#include <stdbool.h>
#include <stdlib.h>
#include "FastTransfer.h"

void wipeBuf(unsigned char * buf, uint16_t size);

circBuff_t ringBuff;

//Captures address of receive array, the max data address, the address of the module, true/false if
AKNAKs are wanted and the Serial address
void begin(int * ptr, unsigned char maxSize, unsigned char givenAddress, bool error, void
(*stufftosend)(unsigned char), unsigned char (*stufftoreceive)(void), int (*stuffavailable)(void), unsigned
char (*stuffpeek)(void))
{
    receiveArrayAddress = ptr;
    moduleAddress = givenAddress;
    serial_write = stufftosend;
    serial_available = stuffavailable;
    serial_peek = stuffpeek;
    serial_read = stufftoreceive;
    maxDataAddress = maxSize / 2;
    AKNAKsend = error;
    alignErrorCounter = 0;
    ringBuff.head=0;
    ringBuff.tail=0;
}

```

```

ringBuff.count=0;
}

//CRC Calculator
unsigned char CRC8(const unsigned char * data, unsigned char len)
{
    unsigned char crc = 0x00;
    while (len--)
    {
        unsigned char extract = *data++;
        unsigned char templ;
        for (templ = 8; templ; templ--)
        {
            unsigned char sum = (crc ^ extract) & 0x01;
            crc >>= 1;
            if (sum)
            {
                crc ^= polynomial;
            }
            extract >>= 1;
        }
    }
    return crc;
}

void testSendByteInternal(void)
{
    serial_write(0x06); //start address
}

//Sends out send buffer with a 2 start bytes, where the packet is going, where it came from, the size of
the data packet, the data and the crc.
bool sendData(unsigned char whereToSend)
{
    //calculate the crc
    unsigned char CS = CRC8(ringBuff.buf, ringBuff.count);

    serial_write(0x06); //start address
    serial_write(0x85); //start address
    serial_write(whereToSend);
    serial_write(moduleAddress);
    serial_write(ringBuff.count); //length of packet not including the crc

    //send the rest of the packet
    int i;
    for (i = 0; i < ringBuff.count; i++)

```

```

{
    serial_write(*(ringBuff.buf + i));
}

//send the crc
serial_write(CS);

//record the sent message data for aknak check later
crcBufS_put(&crc_buffer, whereToSend, CS, 0);

// clears the buffer after a sending
FastTransfer_buffer_flush(&ringBuff, 1);
return true;
}

bool receiveData()
{
    //start off by looking for the header bytes. If they were already found in a previous call, skip it.
    if (rx_len == 0)
    {
        //this size check may be redundant due to the size check below, but for now I'll leave it the way it is.
        if (serial_available() > 4)
        {
            //this will block until a 0x06 is found or buffer size becomes less then 3.
            while (serial_read() != 0x06)
            {
                //This will trash any preamble junk in the serial buffer
                //but we need to make sure there is enough in the buffer to process while we trash the rest
                //if the buffer becomes too empty, we will escape and try again on the next call
                alignErrorCounter++; //increments the counter whenever a byte is trashed
                if (serial_available() < 5)
                    return false;
            }
            if (serial_read() == 0x85)
            {
                rx_address = serial_read(); // pulls the address
                returnAddress = serial_read(); // pulls where the message came from
                rx_len = serial_read(); // pulls the length

                //make sure the address received is a match for this module if not throw the packet away

                if (rx_address != moduleAddress)
                {
                    addressErrorCounter++; // increments a counter whenever the wrong address is received
                    //if the address does not match the buffer is flushed for the size of
                    //the data packet plus one for the CRC
                    int u;
                    for (u = 0; u <= (rx_len + 1); u++)

```

```

    {
        serial_read();
    }
    rx_len = 0; // reset length
    return false;
}
}

}

}

//we get here if we already found the header bytes, the address matched what we know, and now we
are byte aligned.
if (rx_len != 0)
{
    //this check is preformed to see if the first data address is a 255, if it is then this packet is an AKNAK
    if (rx_array_inx == 0)
    {
        while (!(serial_available() >= 1));
        if (255 == serial_peek())
        {
            CRCcheck();
            rx_len = 0;
            rx_array_inx = 0;

            wipeBuf(rx_buffer,RX_BUFFER_LENGTH);
            return false;
        }
    }

    while ((serial_available() > 0 && ( rx_array_inx <= rx_len))//(serial_available() > 0) &&( rx_array_inx
<= rx_len))
    {
        rx_buffer[rx_array_inx++] = serial_read();
    }

    if (rx_len == (rx_array_inx - 1))
    {
        //seem to have got whole message
        //last uint8_t is CS
        calc_CS = CRC8(rx_buffer, rx_len);

        if (calc_CS == rx_buffer[rx_array_inx - 1])
        {//CS good

            //reassembles the data and places it into the receive array according to data address.
            int r;
            for (r = 0; r < rx_len; r = r + 3)
            {

```

```

        if (rx_buffer[r] < maxDataAddress)
        {
            group.parts[0] = rx_buffer[r + 1];
            group.parts[1] = rx_buffer[r + 2];

            receiveArrayAddress[(rx_buffer[r])] = group.integer;
        } else
        {
            dataAdressErrorCounter++;
        }
    }

    if (AKNAKsend)
    { // if enabled sends an AK
        unsigned char holder[3];
        holder[0] = 255;
        holder[1] = 1;
        holder[2] = rx_buffer[rx_array_inx - 1];
        unsigned char crcHolder = CRC8(holder, 3);
        serial_write(0x06);
        serial_write(0x85);
        serial_write(returnAddress);
        serial_write(moduleAddress);
        serial_write(3);
        serial_write(255);
        serial_write(1);
        serial_write(rx_buffer[rx_array_inx - 1]);
        serial_write(crcHolder);
    }

    rx_len = 0;
    rx_array_inx = 0;

    wipeBuf(rx_buffer,RX_BUFFER_LENGTH);
    return true;
}
else
{
    crcErrorCounter++; //increments the counter every time a crc fails

    if (AKNAKsend)
    { // if enabled sends NAK
        unsigned char holder[3];
        holder[0] = 255;
        holder[1] = 2;
        holder[2] = rx_buffer[rx_array_inx - 1];
        unsigned char crcHolder = CRC8(holder, 3);
        serial_write(0x06);
    }
}

```

```

        serial_write(0x85);
        serial_write(returnAddress);
        serial_write(moduleAddress);
        serial_write(3);
        serial_write(255);
        serial_write(2);
        serial_write(rx_buffer[rx_array_inx - 1]);
        serial_write(crcHolder);
    }

    //failed checksum, need to clear this out
    rx_len = 0;
    rx_array_inx = 0;

    wipeBuf(rx_buffer,RX_BUFFER_LENGTH);
    return false;
}
}

return false;
}

// populates to what data address and what info needs to be sent
void ToSend(unsigned char where, unsigned int what)
{
    FastTransfer_buffer_put(&ringBuff, where, what);
}

// disassembles the data and places it in a buffer to be sent
void FastTransfer_buffer_put(circBuff_t *_this, unsigned char towhere, unsigned int towhat)
{
    group.integer = towhat;

    if (_this->count < (BUFFER_SIZE - 3))
    {
        _this->buf[_this->head] = towhere;
        _this->head = FastTransfer_buffer_modulo_inc(_this->head, BUFFER_SIZE);
        ++_this->count;
        _this->buf[_this->head] = group.parts[0];
        _this->head = FastTransfer_buffer_modulo_inc(_this->head, BUFFER_SIZE);
        ++_this->count;
        _this->buf[_this->head] = group.parts[1];
        _this->head = FastTransfer_buffer_modulo_inc(_this->head, BUFFER_SIZE);
        ++_this->count;
    }
}

```

```

//pulls info out of the send buffer in a first in first out fashion
unsigned char FastTransfer_buffer_get(circBuff_t* _this)
{
    unsigned char c;
    if (_this->count > 0)
    {
        c = _this->buf[_this->tail];
        _this->tail = FastTransfer_buffer_modulo_inc(_this->tail, BUFFER_SIZE);
        --_this->count;
    }
    else
    {
        c = 0;
    }
    return (c);
}

//flushes the send buffer to get it ready for new data
void FastTransfer_buffer_flush(circBuff_t* _this, const int clearBuffer)
{
    _this->count = 0;
    _this->head = 0;
    _this->tail = 0;
    if (clearBuffer)
    {
        wipeBuf(_this->buf,sizeof(_this->buf));
    }
}

// increments counters for the buffer functions
unsigned int FastTransfer_buffer_modulo_inc(const unsigned int value, const unsigned int modulus)
{
    unsigned int my_value = value + 1;
    if (my_value >= modulus)
    {
        my_value = 0;
    }
    return (my_value);
}

//searches the buffer for the status of a message that was sent
unsigned char AKNAK(unsigned char module)
{
    int r;
    for (r = 0; r < CRC_COUNT; r++)
    {

```

```

        if (module == crcBufS_get(&crc_buffer, r, 0))
    {
        return crcBufS_get(&crc_buffer, r, 2);
    }
}
return 4;
}

//returns align error
unsigned int alignError(void)
{
    return alignErrorCounter;
}

//returns CRC error
unsigned int CRCError(void)
{
    return crcErrorCounter;
}

//returns address error
unsigned int addressError(void)
{
    return addressErrorCounter;
}

unsigned int dataAddressError(void)
{
    return dataAdressErrorCounter;
}

// after a packet is sent records the info of that packet
void crcBufS_put(struct crcBufS* _this, unsigned char address, unsigned char oldCRC, unsigned char status)
{
    _this->buf[_this->head] = address;
    _this->head++;
    _this->buf[_this->head] = oldCRC;
    _this->head++;
    _this->buf[_this->head] = status;
    _this->head++;
    if (_this->head >= CRC_BUFFER_SIZE)
    {
        _this->head = 0;
    }
}

```

```

// after a Ak or NAK is received that status is stored
void crcBufS_status_put(struct crcBufS* _this, unsigned char time, unsigned char status)
{
    if (time >= CRC_COUNT)
    {
        time = CRC_COUNT - 1;
    }
    time = time + 1;
    int wantedTime = time * 3;
    if (wantedTime > _this->head)
    {
        wantedTime = (CRC_BUFFER_SIZE) - (wantedTime - _this->head);
        _this->buf[(wantedTime + 2)] = status;
    }
    else
    {
        _this->buf[(_this->head - wantedTime) + 2] = status;
    }
}

// pulls data from the AKNAK buffer
unsigned char crcBufS_get(struct crcBufS* _this, unsigned char time, unsigned char space)
{
    if (time >= CRC_COUNT)
    {
        time = CRC_COUNT - 1;
    }
    if (space >= CRC_DEPTH)
    {
        space = CRC_DEPTH - 1;
    }
    time = time + 1;
    int wantedTime = time * 3;
    if (wantedTime > _this->head)
    {
        wantedTime = (CRC_BUFFER_SIZE) - (wantedTime - _this->head);
        return (_this->buf[(wantedTime + space)]);
    }
    else
    {
        return (_this->buf[(_this->head - wantedTime) + space]);
    }
}

//when an AK or NAK is received this compares it to the buffer and records the status
void CRCcheck(void)
{

```

```

while (!(serial_available() > 3)); // trap makes sure that there are enough bytes in the buffer for the
AKNAK check

unsigned char arrayHolder[3];
arrayHolder[0] = serial_read();
arrayHolder[1] = serial_read();
arrayHolder[2] = serial_read();
unsigned char SentCRC = serial_read();
unsigned char calculatedCRC = CRC8(arrayHolder, 3);

if (SentCRC == calculatedCRC)
{
    int rt;
    for (rt = 0; rt < CRC_COUNT; rt++)
    {
        if (returnAddress == crcBufS_get(&crc_buffer, rt, 0))
        {
            if (arrayHolder[2] == crcBufS_get(&crc_buffer, rt, 1))
            {
                if (arrayHolder[1] == 1)
                {
                    crcBufS_status_put(&crc_buffer, rt, 1);
                    break;
                }
                else if (arrayHolder[1] == 2)
                {
                    crcBufS_status_put(&crc_buffer, rt, 2);
                    break;
                }
            }
        }
    }
    else
    {
        crcErrorCounter++;
    } //increments the counter every time a crc fails
}

void wipeBuf(unsigned char * buf, uint16_t size)
{
    int i=0;
    for(i=0;i<size;i++)
    {
        buf[i]=0;
    }
}

```

LED.c

```
#include "LED.h"
#include "Timers.h"
timer_t blinky;
void initLEDs(void)
{
    setTimerInterval(&blinky,1000);
    LED1_TRIS = OUTPUT;
    LED2_TRIS = OUTPUT;
    LED3_TRIS = OUTPUT;
    LED4_TRIS = OUTPUT;

    LED1 = LED_OFF;
    LED2 = LED_OFF;
    LED3 = LED_OFF;
    LED4 = LED_OFF;

}

void setLED(int ledNum, int state)
{
    switch(ledNum)
    {
        case 1:
            LED1 = state;
            break;
        case 2:
            LED2 = state;
            break;
        case 3:
            LED3 = state;
            break;
        case 4:
            LED4 = state;
            break;
    }
}

void toggleLED(int ledNum)
{
    switch(ledNum)
    {
        case 1:
            LED1 ^= 1;
            break;
        case 2:
            LED2 ^= 1;
            break;
    }
}
```

```

        break;
    case 3:
        LED3 ^= 1;
        break;
    case 4:
        LED4 ^= 1;
        break;
    }
}
void blinkLED(int ledNum)
{
    if(timerDone(&blinky))
    {
        toggleLED(ledNum);
        resetTimer(&blinky);
    }
}
MotorDrive.c

```

```

#include "MotorDrive.h"
#include "mcc_generated_files/pwm5.h"
#include "communications.h"

bool motorDriveEnabled=false;
int previousSpeedSetting=0;

void initMotorDrive(void)
{
    motorDriveEnabled=false;
    PWM5_LoadDutyValue(0);
}

void updateMotorDrive(void)
{
    if(previousSpeedSetting!=getSpeedSetting())
    {
        PWM5_LoadDutyValue(getSpeedSetting());
        previousSpeedSetting=getSpeedSetting();
    }
}

void enableMotorDrive(void)
{
    motorDriveEnabled=true;
    PWM5_LoadDutyValue(getSpeedSetting());
}

void disableMotorDrive(void)

```

```
{
    motorDriveEnabled=false;
    PWM5_LoadDutyValue(0);
}
Timers.c
```

```
#include "Timers.h"
#include "mcc_generated_files/tmr0.h"
#include "mcc_generated_files/tmr2.h"

bool timerDone(timer_t * t)
{
//  TMR2_Initialize();
//  TMR2_StartTimer();
//  TMR2_ReadTimer();

    if(millis()>=(t->interval+t->prevTime) )
    {
        t->prevTime=millis();
        return true;
    }
    else
    {
        return false;
    }
}

void setTimerInterval(timer_t * t,unsigned int intervalSetting)
{
    t->interval=intervalSetting;
}

void resetTimer(timer_t * t)
{
    t->prevTime=millis();
}
adc.c
```

```
#include <xc.h>
#include "adc.h"
#include "mcc.h"

/**
Section: Macro Declarations
*/
#define ACQ_US_DELAY 5

/**
```

Section: ADC Module APIs

```

*/
void ADC_Initialize(void)
{
    // set the ADC to the options selected in the User Interface

    // ADGO stop; ADON enabled; CHS ANA0;
    ADCON0 = 0x01;

    // ADFM left; ADNREF VSS; ADPREF VDD; ADCS FOSC/2;
    ADCON1 = 0x00;

    // ADACT no_auto_trigger;
    ADACT = 0x00;

    // ADRESL 0;
    ADRESL = 0x00;

    // ADRESH 0;
    ADRESH = 0x00;

}

void ADC_SelectChannel(adc_channel_t channel)
{
    // select the A/D channel
    ADCON0bits.CHS = channel;
    // Turn on the ADC module
    ADCON0bits.ADON = 1;
}

void ADC_StartConversion()
{
    // Start the conversion
    ADCON0bits.ADGO = 1;
}

bool ADC_IsConversionDone()
{
    // Start the conversion
    return (!ADCON0bits.ADGO);
}

adc_result_t ADC_GetConversionResult(void)
{
    // Conversion finished, return the result
}

```

```

    return ((ADRESH << 8) + ADRESL);
}

adc_result_t ADC_GetConversion(adc_channel_t channel)
{
    // select the A/D channel
    ADCON0bits.CHS = channel;

    // Turn on the ADC module
    ADCON0bits.ADON = 1;
    // Acquisition time delay
    __delay_us(ACQ_US_DELAY);

    // Start the conversion
    ADCON0bits.ADGO = 1;

    // Wait for the conversion to finish
    while (ADCON0bits.ADGO)
    {
    }
    // Conversion finished, return the result
    return ((ADRESH << 8) + ADRESL);
}

/***
End of File
*/
euart.c

#include "eusart.h"
#include "stdint.h"
#include "../LED.h"
/***
Section: Macro Declarations
*/
#define EUSART_TX_BUFFER_SIZE 64
#define EUSART_RX_BUFFER_SIZE 64

/***
Section: Global Variables
*/
volatile uint8_t eusartTxHead = 0;
volatile uint8_t eusartTxTail = 0;
volatile uint8_t eusartTxBuffer[EUSART_TX_BUFFER_SIZE];
volatile uint8_t eusartTxBufferRemaining;

```

```

volatile uint8_t eusartRxHead = 0;
volatile uint8_t eusartRxTail = 0;
volatile uint8_t eusartRxBuffer[EUSART_RX_BUFFER_SIZE];
volatile uint8_t eusartRxCount;

/***
Section: EUSART APIs
***/

void EUSART_Initialize(void)
{
    // disable interrupts before changing states
    PIE1bits.RCIE = 0;
    PIE1bits.TXIE = 0;

    // Set the EUSART module to the options selected in the user interface.

    // ABDOVF no_overflow; SCKP Non-Inverted; BRG16 16bit_generator; WUE disabled; ABDEN disabled;
    BAUD1CON = 0x08;

    // SPEN enabled; RX9 8-bit; CREN enabled; ADDEN disabled; SREN disabled;
    RC1STA = 0x90;

    // TX9 8-bit; TX9D 0; SENDB sync_break_complete; TXEN enabled; SYNC asynchronous; BRGH
    hi_speed; CSRC slave;
    TX1STA = 0x24;

    // Baud Rate = 115200; SP1BRGL 68;
    SP1BRGL = 0x44;

    // Baud Rate = 115200; SP1BRGH 0;
    SP1BRGH = 0x00;

    // initializing the driver state
    eusartTxHead = 0;
    eusartTxTail = 0;
    eusartTxBufferRemaining = sizeof(eusartTxBuffer);

    eusartRxHead = 0;
    eusartRxTail = 0;
    eusartRxCount = 0;

    // enable receive interrupt
    PIE1bits.RCIE = 1;
}

uint8_t EUSART_Read(void)

```

```

{
    uint8_t readValue = 0;

    while(0 == eusartRxCount)
    {
    }

    readValue = eusartRxBuffer[eusartRxTail++];
    if(sizeof(eusartRxBuffer) <= eusartRxTail)
    {
        eusartRxTail = 0;
    }
    PIE1bits.RCIE = 0;
    eusartRxCount--;
    PIE1bits.RCIE = 1;

    return readValue;
}

unsigned char EUSART_Peek(void)
{
    uint8_t readValue = 0;

    readValue = eusartRxBuffer[eusartRxTail];

    return readValue;
}

int EUSART_Bytes_Available(void){

    return eusartRxCount;
}

void EUSART_Write(uint8_t txData)
{
    while(0 == eusartTxBufferRemaining)
    {

    }

    if(0 == PIE1bits.TXIE)
    {
        TX1REG = txData;
    }
    else
    {
        PIE1bits.TXIE = 0;
    }
}

```

```

eusartTxBuffer[eusartTxHead++] = txData;
if(sizeof(eusartTxBuffer) <= eusartTxHead)
{
    eusartTxHead = 1;
}
eusartTxBufferRemaining--;
}
PIE1bits.TXIE = 1;
}

char getch(void)
{
    return EUSART_Read();
}

void putch(char txData)
{
    EUSART_Write(txData);
}

void EUSART_Transmit_ISR(void)
{

// add your EUSART interrupt custom code
if(sizeof(eusartTxBuffer) > eusartTxBufferRemaining)
{
    TX1REG = eusartTxBuffer[eusartTxTail++];
    if(sizeof(eusartTxBuffer) <= eusartTxTail)
    {
        eusartTxTail = 0;
    }
    eusartTxBufferRemaining++;
}
else
{
    PIE1bits.TXIE = 0;
}
}

void EUSART_Receive_ISR(void)
{
//  toggleLED(1);

if(1 == RC1STAbits.OERR)
{
// EUSART error - restart

RC1STAbits.CREN = 0;
}

```

```

    RC1STAbits.CREN = 1;
}

// buffer overruns are ignored
eusartRxBuffer[eusartRxHead++] = RC1REG;
if(sizeof(eusartRxBuffer) <= eusartRxHead)
{
    eusartRxHead = 0;
}
eusartRxCount++;
}

ext_int.c

#include <xc.h>
#include "ext_int.h"
/**User Area Begin->code: Add External Interrupt handler specific headers

/**User Area End->code: Add External Interrupt handler specific headers

/**
Section: External Interrupt Handlers
*/
// INTn Dynamic Interrupt Handlers
void (*INT_InterruptHandler)(void);

/**
Interrupt Handler for EXT_INT - INT
*/
void INT_ISR(void)
{
    /**User Area Begin->code***

    /**User Area End->code***


    EXT_INT_InterruptFlagClear();

    // Callback function gets called everytime this ISR executes
    INT_CallBack();
}

/**
Callback function for EXT_INT - INT
*/
void INT_CallBack(void)
{
    // Add your custom callback code here
    if(INT_InterruptHandler)
    {

```

```

        INT_InterruptHandler();
    }
}

/***
Allows selecting an interrupt handler for EXT_INT - INT at application runtime
*/
void INT_SetInterruptHandler(void* InterruptHandler){
    INT_InterruptHandler = InterruptHandler;
}

/***
Default interrupt handler for EXT_INT - INT
*/
void INT_DefaultInterruptHandler(void){
    // add your INT interrupt custom code
    // or set custom function using INT_SetInterruptHandler()
}

/***
Section: External Interrupt Initializers
*/
/***
void EXT_INT_Initialize(void)

Initializer for the following external interrupts
INT
*/
void EXT_INT_Initialize(void)
{

*****
 * INT
 * Clear the interrupt flag
 * Set the external interrupt edge detect
 * Enable the interrupt, if enabled in the UI.
*****
EXT_INT_InterruptFlagClear();
EXT_INT_fallingEdgeSet();
// Set Default Interrupt Handler
INT_SetInterruptHandler(INT_DefaultInterruptHandler);
EXT_INT_InterruptEnable();

}
i2c1.c

#include "i2c1.h"

/***

```

I2C Driver Queue Status Type

@Summary

Defines the type used for the transaction queue status.

@Description

This defines type used to keep track of the queue status.

*/

```
typedef union
{
    struct
    {
        uint8_t full:1;
        uint8_t empty:1;
        uint8_t reserved:6;
    }s;
    uint8_t status;
}I2C_TR_QUEUE_STATUS;
```

/**

I2C Driver Queue Entry Type

@Summary

Defines the object used for an entry in the i2c queue items.

@Description

This defines the object in the i2c queue. Each entry is a composed of a list of TRBs, the number of the TRBs and the status of the currently processed TRB.

*/

```
typedef struct
{
    uint8_t           count;      // a count of trb's in the trb list
    I2C1_TRANSACTION_REQUEST_BLOCK *ptrb_list; // pointer to the trb list
    I2C1_MESSAGE_STATUS     *pTrFlag;    // set with the error of the last trb sent.
                                         // if all trb's are sent successfully,
                                         // then this is I2C1_MESSAGE_COMPLETE
} I2C_TR_QUEUE_ENTRY;
```

/**

I2C Master Driver Object Type

@Summary

Defines the object that manages the i2c master.

@Description

This defines the object that manages the sending and receiving of

```

i2c master transactions.
*/
typedef struct
{
    /* Read/Write Queue */
    I2C_TR_QUEUE_ENTRY      *pTrTail;    // tail of the queue
    I2C_TR_QUEUE_ENTRY      *pTrHead;    // head of the queue
    I2C_TR_QUEUE_STATUS     trStatus;    // status of the last transaction
    uint8_t                  i2cDoneFlag; // flag to indicate the current
                                         // transaction is done
    uint8_t                  i2cErrors;   // keeps track of errors

} I2C_OBJECT;

/***
I2C Master Driver State Enumeration

@Summary
Defines the different states of the i2c master.

@Description
This defines the different states that the i2c master
used to process transactions on the i2c bus.
*/
typedef enum
{
    S_MASTER_IDLE,
    S_MASTER_RESTART,
    S_MASTER_SEND_ADDR,
    S_MASTER_SEND_DATA,
    S_MASTER_SEND_STOP,
    S_MASTER_ACK_ADDR,
    S_MASTER_RCV_DATA,
    S_MASTER_RCV_STOP,
    S_MASTER_ACK_RCV_DATA,
    S_MASTER_NOACK_STOP,
    S_MASTER_SEND_ADDR_10BIT_LSB,
    S_MASTER_10BIT_RESTART,

} I2C_MASTER_STATES;

/***
Section: Macro Definitions
*/

```

```

/* defined for I2C1 */

#ifndef I2C1_CONFIG_TR_QUEUE_LENGTH
#define I2C1_CONFIG_TR_QUEUE_LENGTH 1
#endif

#define I2C1_TRANSMIT_REG           SSP1BUF      // Defines the transmit register used to
send data.
#define I2C1_RECEIVE_REG           SSP1BUF      // Defines the receive register used to
receive data.

// The following control bits are used in the I2C state machine to manage
// the I2C module and determine next states.
#define I2C1_WRITE_COLLISION_STATUS_BIT    SSP1CON1bits.WCOL  // Defines the write collision
status bit.
#define I2C1_MODE_SELECT_BITS          SSP1CON1bits.SSPM  // I2C Master Mode control bit.
#define I2C1_MASTER_ENABLE_CONTROL_BITS SSP1CON1bits.SSPEN // I2C port enable control
bit.

#define I2C1_START_CONDITION_ENABLE_BIT   SSP1CON2bits.SEN   // I2C START control bit.
#define I2C1_REPEAT_START_CONDITION_ENABLE_BIT SSP1CON2bits.RSEN // I2C Repeated START
control bit.
#define I2C1_RECEIVE_ENABLE_BIT         SSP1CON2bits.RCEN  // I2C Receive enable control bit.
#define I2C1_STOP_CONDITION_ENABLE_BIT   SSP1CON2bits.PEN   // I2C STOP control bit.
#define I2C1_ACKNOWLEDGE_ENABLE_BIT     SSP1CON2bits.ACKEN // I2C ACK start control bit.
#define I2C1_ACKNOWLEDGE_DATA_BIT      SSP1CON2bits.ACKDT // I2C ACK data control bit.
#define I2C1_ACKNOWLEDGE_STATUS_BIT    SSP1CON2bits.ACKSTAT // I2C ACK status bit.

#define I2C1_7bit  true
/***
Section: Local Functions
***/

void I2C1_FunctionComplete(void);
void I2C1_Stop(I2C1_MESSAGE_STATUS completion_code);

/***
Section: Local Variables
***/

static I2C_TR_QUEUE_ENTRY      i2c1_tr_queue[I2C1_CONFIG_TR_QUEUE_LENGTH];
static I2C_OBJECT              i2c1_object;
static I2C_MASTER_STATES       i2c1_state = S_MASTER_IDLE;
static uint8_t                  i2c1_trb_count = 0;

static I2C1_TRANSACTION_REQUEST_BLOCK *p_i2c1_trb_current = NULL;
static I2C_TR_QUEUE_ENTRY        *p_i2c1_current = NULL;

```

```

/**
Section: Driver Interface
*/
void I2C1_Initialize(void)
{
    i2c1_object.pTrHead = i2c1_tr_queue;
    i2c1_object.pTrTail = i2c1_tr_queue;
    i2c1_object.trStatus.s.empty = true;
    i2c1_object.trStatus.s.full = false;

    i2c1_object.i2cErrors = 0;

    // R_nW write_noTX; P stopbit_notdetected; S startbit_notdetected; BF RCinprocess_TXcomplete;
    // SMP High Speed; UA dontupdate; CKE disabled; D_nA lastbyte_address;
    SSP1STAT = 0x00;
    // SSPEN enabled; WCOL no_collision; CKP Idle:Low, Active:High; SSPM FOSC/4_SSPrxADD_I2C; SSPOV
    no_overflow;
    SSP1CON1 = 0x28;
    // ACKT1M ackseq; SBCDE disabled; BOEN disabled; SCIE disabled; PCIE disabled; DHEN disabled;
    SDAHT 100ns; AHEN disabled;
    SSP1CON3 = 0x00;
    // Baud Rate Generator Value: SSP1ADD 3;
    SSP1ADD = 0x03;

    // clear the master interrupt flag
    PIR1bits.SSP1IF = 0;
    // enable the master interrupt
    PIE1bits.SSP1IE = 1;

}

uint8_t I2C1_ErrorCountGet(void)
{
    uint8_t ret;

    ret = i2c1_object.i2cErrors;
    return ret;
}

void I2C1_ISR ( void )
{
    static uint8_t *pi2c_buf_ptr;

```

```

static uint16_t i2c_address      = 0;
static uint8_t i2c_bytes_left    = 0;
static uint8_t i2c_10bit_address_restart = 0;

PIR1bits.SSP1IF = 0;

// Check first if there was a collision.
// If we have a Write Collision, reset and go to idle state */
if(I2C1_WRITE_COLLISION_STATUS_BIT)
{
    // clear the Write colision
    I2C1_WRITE_COLLISION_STATUS_BIT = 0;
    i2c1_state = S_MASTER_IDLE;
    *(p_i2c1_current->pTrFlag) = I2C1_MESSAGE_FAIL;

    // reset the buffer pointer
    p_i2c1_current = NULL;

    return;
}

/* Handle the correct i2c state */
switch(i2c1_state)
{
    case S_MASTER_IDLE: /* In reset state, waiting for data to send */

        if(i2c1_object.trStatus.s.empty != true)
        {
            // grab the item pointed by the head
            p_i2c1_current = i2c1_object.pTrHead;
            i2c1_trb_count = i2c1_object.pTrHead->count;
            p_i2c1_trb_current = i2c1_object.pTrHead->ptrib_list;

            i2c1_object.pTrHead++;

            // check if the end of the array is reached
            if(i2c1_object.pTrHead == (i2c1_tr_queue + I2C1_CONFIG_TR_QUEUE_LENGTH))
            {
                // adjust to restart at the beginning of the array
                i2c1_object.pTrHead = i2c1_tr_queue;
            }

            // since we moved one item to be processed, we know
            // it is not full, so set the full status to false
            i2c1_object.trStatus.s.full = false;

            // check if the queue is empty
            if(i2c1_object.pTrHead == i2c1_object.pTrTail)

```

```

{
    // it is empty so set the empty status to true
    i2c1_object.trStatus.s.empty = true;
}

// send the start condition
I2C1_START_CONDITION_ENABLE_BIT = 1;

// start the i2c request
i2c1_state = S_MASTER_SEND_ADDR;
}

break;

case S_MASTER_RESTART:

/* check for pending i2c Request */

// ... trigger a REPEATED START
I2C1_REPEAT_START_CONDITION_ENABLE_BIT = 1;

// start the i2c request
i2c1_state = S_MASTER_SEND_ADDR;

break;

case S_MASTER_SEND_ADDR_10BIT_LSB:

if(I2C1_ACKNOWLEDGE_STATUS_BIT)
{
    i2c1_object.i2cErrors++;
    I2C1_Stop(I2C1_MESSAGE_ADDRESS_NO_ACK);
}
else
{
    // Remove bit 0 as R/W is never sent here
    I2C1_TRANSMIT_REG = (i2c_address >> 1) & 0x00FF;

    // determine the next state, check R/W
    if(i2c_address & 0x01)
    {
        // if this is a read we must repeat start
        // the bus to perform a read
        i2c1_state = S_MASTER_10BIT_RESTART;
    }
    else
    {
        // this is a write continue writing data

```

```

        i2c1_state = S_MASTER_SEND_DATA;
    }
}

break;

case S_MASTER_10BIT_RESTART:

if(I2C1_ACKNOWLEDGE_STATUS_BIT)
{
    i2c1_object.i2cErrors++;
    I2C1_Stop(I2C1_MESSAGE_ADDRESS_NO_ACK);
}
else
{
    // ACK Status is good
    // restart the bus
    I2C1_REPEAT_START_CONDITION_ENABLE_BIT = 1;

    // fudge the address so S_MASTER_SEND_ADDR works correctly
    // we only do this on a 10-bit address resend
    i2c_address = 0x00F0 | ((i2c_address >> 8) & 0x0006);

    // set the R/W flag
    i2c_address |= 0x0001;

    // set the address restart flag so we do not change the address
    i2c_10bit_address_restart = 1;

    // Resend the address as a read
    i2c1_state = S_MASTER_SEND_ADDR;
}

break;

case S_MASTER_SEND_ADDR:

/* Start has been sent, send the address byte */

/* Note:
On a 10-bit address resend (done only during a 10-bit
device read), the original i2c_address was modified in
S_MASTER_10BIT_RESTART state. So the check if this is
a 10-bit address will fail and a normal 7-bit address
is sent with the R/W bit set to read. The flag
i2c_10bit_address_restart prevents the address to
be re-written.
*/

```

```

if(i2c_10bit_address_restart != 1)
{
    // extract the information for this message
    i2c_address = p_i2c1_trb_current->address;
    pi2c_buf_ptr = p_i2c1_trb_current->pbuffer;
    i2c_bytes_left = p_i2c1_trb_current->length;
}

// check for 10-bit address
if(!I2C1_7bit && (0x0 != i2c_address))
{
    if (0 == i2c_10bit_address_restart)
    {
        // we have a 10 bit address
        // send bits<9:8>
        // mask bit 0 as this is always a write
        I2C1_TRANSMIT_REG = 0xF0 | ((i2c_address >> 8) & 0x0006);
        i2c1_state = S_MASTER_SEND_ADDR_10BIT_LSB;
    }
    else
    {
        // resending address bits<9:8> to trigger read
        I2C1_TRANSMIT_REG = i2c_address;
        i2c1_state = S_MASTER_ACK_ADDR;
        // reset the flag so the next access is ok
        i2c_10bit_address_restart = 0;
    }
}
else
{
    // Transmit the address
    I2C1_TRANSMIT_REG = i2c_address;
    if(i2c_address & 0x01)
    {
        // Next state is to wait for address to be acked
        i2c1_state = S_MASTER_ACK_ADDR;
    }
    else
    {
        // Next state is transmit
        i2c1_state = S_MASTER_SEND_DATA;
    }
}
break;

case S_MASTER_SEND_DATA:

    // Make sure the previous byte was acknowledged

```

```

if(I2C1_ACKNOWLEDGE_STATUS_BIT)
{
    // Transmission was not acknowledged
    i2c1_object.i2cErrors++;

    // Reset the Ack flag
    I2C1_ACKNOWLEDGE_STATUS_BIT = 0;

    // Send a stop flag and go back to idle
    I2C1_Stop(I2C1_DATA_NO_ACK);

}

else
{
    // Did we send them all ?
    if(i2c_bytes_left-- == 0U)
    {
        // yup sent them all!

        // update the trb pointer
        p_i2c1_trb_current++;

        // are we done with this string of requests?
        if(--i2c1_trb_count == 0)
        {
            I2C1_Stop(I2C1_MESSAGE_COMPLETE);
        }
        else
        {
            // no!, there are more TRB to be sent.
            //I2C1_START_CONDITION_ENABLE_BIT = 1;

            // In some cases, the slave may require
            // a restart instead of a start. So use this one
            // instead.
            I2C1_REPEAT_START_CONDITION_ENABLE_BIT = 1;

            // start the i2c request
            i2c1_state = S_MASTER_SEND_ADDR;

        }
    }
    else
    {
        // Grab the next data to transmit
        I2C1_TRANSMIT_REG = *pi2c_buf_ptr++;
    }
}

```

```

break;

case S_MASTER_ACK_ADDR:

/* Make sure the previous byte was acknowledged */
if(I2C1_ACKNOWLEDGE_STATUS_BIT)
{

    // Transmission was not acknowledged
    i2c1_object.i2cErrors++;

    // Send a stop flag and go back to idle
    I2C1_Stop(I2C1_MESSAGE_ADDRESS_NO_ACK);

    // Reset the Ack flag
    I2C1_ACKNOWLEDGE_STATUS_BIT = 0;
}
else
{
    I2C1_RECEIVE_ENABLE_BIT = 1;
    i2c1_state = S_MASTER_ACK_RCV_DATA;
}
break;

case S_MASTER_RCV_DATA:

/* Acknowledge is completed. Time for more data */

// Next thing is to ack the data
i2c1_state = S_MASTER_ACK_RCV_DATA;

// Set up to receive a byte of data
I2C1_RECEIVE_ENABLE_BIT = 1;

break;

case S_MASTER_ACK_RCV_DATA:

// Grab the byte of data received and acknowledge it
*pi2c_buf_ptr++ = I2C1_RECEIVE_REG;

// Check if we received them all?
if(--i2c_bytes_left)
{

    /* No, there's more to receive */

    // No, bit 7 is clear. Data is ok
}

```

```

// Set the flag to acknowledge the data
I2C1_ACKNOWLEDGE_DATA_BIT = 0;

// Wait for the acknowledge to complete, then get more
i2c1_state = S_MASTER_RCV_DATA;
}

else
{

    // Yes, it's the last byte. Don't ack it
    // Flag that we will nak the data
    I2C1_ACKNOWLEDGE_DATA_BIT = 1;

    I2C1_FunctionComplete();
}

// Initiate the acknowledge
I2C1_ACKNOWLEDGE_ENABLE_BIT = 1;
break;

case S_MASTER_RCV_STOP:
case S_MASTER_SEND_STOP:

    // Send the stop flag
    I2C1_Stop(I2C1_MESSAGE_COMPLETE);
    break;

default:

    // This case should not happen, if it does then
    // terminate the transfer
    i2c1_object.i2cErrors++;
    I2C1_Stop(I2C1_LOST_STATE);
    break;

}
}

void I2C1_FunctionComplete(void)
{
    // update the trb pointer
    p_i2c1_trb_current++;

    // are we done with this string of requests?
    if(--i2c1_trb_count == 0)
    {
        i2c1_state = S_MASTER_SEND_STOP;
}

```

```

    }

else
{
    i2c1_state = S_MASTER_RESTART;
}

}

void I2C1_Stop(I2C1_MESSAGE_STATUS completion_code)
{
    // then send a stop
    I2C1_STOP_CONDITION_ENABLE_BIT = 1;

    // make sure the flag pointer is not NULL
    if (p_i2c1_current->pTrFlag != NULL)
    {
        // update the flag with the completion code
        *(p_i2c1_current->pTrFlag) = completion_code;
    }

    // Done, back to idle
    i2c1_state = S_MASTER_IDLE;
}

void I2C1_MasterWrite(
    uint8_t *pdata,
    uint8_t length,
    uint16_t address,
    I2C1_MESSAGE_STATUS *pflag)
{
    static I2C1_TRANSACTION_REQUEST_BLOCK trBlock;

    // check if there is space in the queue
    if (i2c1_object.trStatus.s.full != true)
    {
        I2C1_MasterWriteTRBBuild(&trBlock, pdata, length, address);
        I2C1_MasterTRBInsert(1, &trBlock, pflag);
    }
    else
    {
        *pflag = I2C1_MESSAGE_FAIL;
    }
}

void I2C1_MasterRead(
    uint8_t *pdata,

```

```

        uint8_t length,
        uint16_t address,
        I2C1_MESSAGE_STATUS *pflag)
{
    static I2C1_TRANSACTION_REQUEST_BLOCK trBlock;

    // check if there is space in the queue
    if (i2c1_object.trStatus.s.full != true)
    {
        I2C1_MasterReadTRBBuild(&trBlock, pdata, length, address);
        I2C1_MasterTRBInsert(1, &trBlock, pflag);
    }
    else
    {
        *pflag = I2C1_MESSAGE_FAIL;
    }
}

void I2C1_MasterTRBInsert(
    uint8_t count,
    I2C1_TRANSACTION_REQUEST_BLOCK *ptrb_list,
    I2C1_MESSAGE_STATUS *pflag)
{
    // check if there is space in the queue
    if (i2c1_object.trStatus.s.full != true)
    {
        *pflag = I2C1_MESSAGE_PENDING;

        i2c1_object.pTrTail->ptrb_list = ptrb_list;
        i2c1_object.pTrTail->count = count;
        i2c1_object.pTrTail->pTrFlag = pflag;
        i2c1_object.pTrTail++;

        // check if the end of the array is reached
        if (i2c1_object.pTrTail == (i2c1_tr_queue + I2C1_CONFIG_TR_QUEUE_LENGTH))
        {
            // adjust to restart at the beginning of the array
            i2c1_object.pTrTail = i2c1_tr_queue;
        }

        // since we added one item to be processed, we know
        // it is not empty, so set the empty status to false
        i2c1_object.trStatus.s.empty = false;

        // check if full
    }
}

```

```

        if (i2c1_object.pTrHead == i2c1_object.pTrTail)
    {
        // it is full, set the full status to true
        i2c1_object.trStatus.s.full = true;
    }

}

else
{
    *pflag = I2C1_MESSAGE_FAIL;
}

// for interrupt based
if (*pflag == I2C1_MESSAGE_PENDING)
{
    while(i2c1_state != S_MASTER_IDLE);
    {
        // force the task to run since we know that the queue has
        // something that needs to be sent
        PIR1bits.SSP1IF = true;
    }
} // block until request is complete

}

void I2C1_MasterReadTRBBuild(
    I2C1_TRANSACTION_REQUEST_BLOCK *ptrb,
    uint8_t *pdata,
    uint8_t length,
    uint16_t address)
{
    ptrb->address = address << 1;
    // make this a read
    ptrb->address |= 0x01;
    ptrb->length = length;
    ptrb->pbuffer = pdata;
}

void I2C1_MasterWriteTRBBuild(
    I2C1_TRANSACTION_REQUEST_BLOCK *ptrb,
    uint8_t *pdata,
    uint8_t length,
    uint16_t address)
{
    ptrb->address = address << 1;
    ptrb->length = length;
    ptrb->pbuffer = pdata;
}

```

```

bool I2C1_MasterQueueIsEmpty(void)
{
    return(i2c1_object.trStatus.s.empty);
}

bool I2C1_MasterQueueIsFull(void)
{
    return(i2c1_object.trStatus.s.full);
}

void I2C1_BusCollisionISR( void )
{
    // enter bus collision handling code here
}

```

i2c2.c

```

#include "i2c2.h"

/** 
I2C Driver Queue Status Type

@Summary
Defines the type used for the transaction queue status.

@Description
This defines type used to keep track of the queue status.
*/

typedef union
{
    struct
    {
        uint8_t full:1;
        uint8_t empty:1;
        uint8_t reserved:6;
    }s;
    uint8_t status;
}I2C_TR_QUEUE_STATUS;

/** 
I2C Driver Queue Entry Type

@Summary
Defines the object used for an entry in the i2c queue items.

```

```

@Description
This defines the object in the i2c queue. Each entry is a composed
of a list of TRBs, the number of the TRBs and the status of the
currently processed TRB.
*/
typedef struct
{
    uint8_t           count;      // a count of trb's in the trb list
    I2C2_TRANSACTION_REQUEST_BLOCK *ptrb_list; // pointer to the trb list
    I2C2_MESSAGE_STATUS      *pTrFlag;   // set with the error of the last trb sent.
                                         // if all trb's are sent successfully,
                                         // then this is I2C2_MESSAGE_COMPLETE
} I2C_TR_QUEUE_ENTRY;

/***
I2C Master Driver Object Type

@Summary
Defines the object that manages the i2c master.

@Description
This defines the object that manages the sending and receiving of
i2c master transactions.
*/

typedef struct
{
    /* Read/Write Queue */
    I2C_TR_QUEUE_ENTRY     *pTrTail;    // tail of the queue
    I2C_TR_QUEUE_ENTRY     *pTrHead;    // head of the queue
    I2C_TR_QUEUE_STATUS    trStatus;    // status of the last transaction
    uint8_t                 i2cDoneFlag; // flag to indicate the current
                                         // transaction is done
    uint8_t                 i2cErrors;   // keeps track of errors

} I2C_OBJECT;

/***
I2C Master Driver State Enumeration

@Summary
Defines the different states of the i2c master.

@Description
This defines the different states that the i2c master
used to process transactions on the i2c bus.
*/

```

```

typedef enum
{
    S_MASTER_IDLE,
    S_MASTER_RESTART,
    S_MASTER_SEND_ADDR,
    S_MASTER_SEND_DATA,
    S_MASTER_SEND_STOP,
    S_MASTER_ACK_ADDR,
    S_MASTER_RCV_DATA,
    S_MASTER_RCV_STOP,
    S_MASTER_ACK_RCV_DATA,
    S_MASTER_NOACK_STOP,
    S_MASTER_SEND_ADDR_10BIT_LSB,
    S_MASTER_10BIT_RESTART,
}

} I2C_MASTER_STATES;

/***
Section: Macro Definitions
***/

/* defined for I2C2 */

#ifndef I2C2_CONFIG_TR_QUEUE_LENGTH
#define I2C2_CONFIG_TR_QUEUE_LENGTH 1
#endif

#define I2C2_TRANSMIT_REG           SSP2BUF      // Defines the transmit register used to
send data.
#define I2C2_RECEIVE_REG           SSP2BUF      // Defines the receive register used to
receive data.

// The following control bits are used in the I2C state machine to manage
// the I2C module and determine next states.
#define I2C2_WRITE_COLLISION_STATUS_BIT   SSP2CON1bits.WCOL  // Defines the write collision
status bit.
#define I2C2_MODE_SELECT_BITS          SSP2CON1bits.SSPM  // I2C Master Mode control bit.
#define I2C2_MASTER_ENABLE_CONTROL_BITS SSP2CON1bits.SSPEN // I2C port enable control
bit.

#define I2C2_START_CONDITION_ENABLE_BIT SSP2CON2bits.SEN  // I2C START control bit.
#define I2C2_REPEAT_START_CONDITION_ENABLE_BIT SSP2CON2bits.RSEN // I2C Repeated START
control bit.
#define I2C2_RECEIVE_ENABLE_BIT        SSP2CON2bits.RCEN // I2C Receive enable control bit.
#define I2C2_STOP_CONDITION_ENABLE_BIT SSP2CON2bits.PEN  // I2C STOP control bit.
#define I2C2_ACKNOWLEDGE_ENABLE_BIT   SSP2CON2bits.ACKEN // I2C ACK start control bit.
#define I2C2_ACKNOWLEDGE_DATA_BIT    SSP2CON2bits.ACKDT // I2C ACK data control bit.

```

```

#define I2C2_ACKNOWLEDGE_STATUS_BIT      SSP2CON2bits.ACKSTAT // I2C ACK status bit.

#define I2C2_7bit  true
/***
Section: Local Functions
***/

void I2C2_FunctionComplete(void);
void I2C2_Stop(I2C2_MESSAGE_STATUS completion_code);

/***
Section: Local Variables
***/

static I2C_TR_QUEUE_ENTRY      i2c2_tr_queue[I2C2_CONFIG_TR_QUEUE_LENGTH];
static I2C_OBJECT              i2c2_object;
static I2C_MASTER_STATES       i2c2_state = S_MASTER_IDLE;
static uint8_t                  i2c2_trb_count = 0;

static I2C2_TRANSACTION_REQUEST_BLOCK *p_i2c2_trb_current = NULL;
static I2C_TR_QUEUE_ENTRY          *p_i2c2_current = NULL;

/***
Section: Driver Interface
***/

void I2C2_Initialize(void)
{
    i2c2_object.pTrHead = i2c2_tr_queue;
    i2c2_object.pTrTail = i2c2_tr_queue;
    i2c2_object.trStatus.s.empty = true;
    i2c2_object.trStatus.s.full = false;

    i2c2_object.i2cErrors = 0;

    // R_nW write_noTX; P stopbit_notdetected; S startbit_notdetected; BF RCinprocess_TXcomplete;
    // SMP High Speed; UA dontupdate; CKE disabled; D_nA lastbyte_address;
    SSP2STAT = 0x00;
    // SSPEN enabled; WCOL no_collision; CKP Idle:Low, Active:High; SSPM FOSC/4_SSPrADD_I2C; SSPOV
    no_overflow;
    SSP2CON1 = 0x28;
    // ACKT1M ackseq; SBCDE disabled; BOEN disabled; SCIE disabled; PCIE disabled; DHEN disabled;
    SDAHT 100ns; AHEN disabled;
    SSP2CON3 = 0x00;
    // Baud Rate Generator Value: SSP2ADD 3;
    SSP2ADD = 0x03;
}

```

```

// clear the master interrupt flag
PIR2bits.SSP2IF = 0;
// enable the master interrupt
PIE2bits.SSP2IE = 1;

}

uint8_t I2C2_ErrorCount(void)
{
    uint8_t ret;

    ret = i2c2_object.i2cErrors;
    return ret;
}

void I2C2_ISR ( void )
{
    static uint8_t *pi2c_buf_ptr;
    static uint16_t i2c_address      = 0;
    static uint8_t i2c_bytes_left    = 0;
    static uint8_t i2c_10bit_address_restart = 0;

    PIR2bits.SSP2IF = 0;

    // Check first if there was a collision.
    // If we have a Write Collision, reset and go to idle state */
    if(I2C2_WRITE_COLLISION_STATUS_BIT)
    {
        // clear the Write colision
        I2C2_WRITE_COLLISION_STATUS_BIT = 0;
        i2c2_state = S_MASTER_IDLE;
        *(p_i2c2_current->pTrFlag) = I2C2_MESSAGE_FAIL;

        // reset the buffer pointer
        p_i2c2_current = NULL;

        return;
    }

    /* Handle the correct i2c state */
    switch(i2c2_state)
    {
        case S_MASTER_IDLE: /* In reset state, waiting for data to send */

```

```

if(i2c2_object.trStatus.s.empty != true)
{
    // grab the item pointed by the head
    p_i2c2_current = i2c2_object.pTrHead;
    i2c2_trb_count = i2c2_object.pTrHead->count;
    p_i2c2_trb_current = i2c2_object.pTrHead->ptrb_list;

    i2c2_object.pTrHead++;

    // check if the end of the array is reached
    if(i2c2_object.pTrHead == (i2c2_tr_queue + I2C2_CONFIG_TR_QUEUE_LENGTH))
    {
        // adjust to restart at the beginning of the array
        i2c2_object.pTrHead = i2c2_tr_queue;
    }

    // since we moved one item to be processed, we know
    // it is not full, so set the full status to false
    i2c2_object.trStatus.s.full = false;

    // check if the queue is empty
    if(i2c2_object.pTrHead == i2c2_object.pTrTail)
    {
        // it is empty so set the empty status to true
        i2c2_object.trStatus.s.empty = true;
    }

    // send the start condition
    I2C2_START_CONDITION_ENABLE_BIT = 1;

    // start the i2c request
    i2c2_state = S_MASTER_SEND_ADDR;
}

break;

case S_MASTER_RESTART:

/* check for pending i2c Request */

// ... trigger a REPEATED START
I2C2_REPEAT_START_CONDITION_ENABLE_BIT = 1;

// start the i2c request
i2c2_state = S_MASTER_SEND_ADDR;

break;

```

```

case S_MASTER_SEND_ADDR_10BIT_LSB:

    if(I2C2_ACKNOWLEDGE_STATUS_BIT)
    {
        i2c2_object.i2cErrors++;
        I2C2_Stop(I2C2_MESSAGE_ADDRESS_NO_ACK);
    }
    else
    {
        // Remove bit 0 as R/W is never sent here
        I2C2_TRANSMIT_REG = (i2c_address >> 1) & 0x00FF;

        // determine the next state, check R/W
        if(i2c_address & 0x01)
        {
            // if this is a read we must repeat start
            // the bus to perform a read
            i2c2_state = S_MASTER_10BIT_RESTART;
        }
        else
        {
            // this is a write continue writing data
            i2c2_state = S_MASTER_SEND_DATA;
        }
    }

    break;

case S_MASTER_10BIT_RESTART:

    if(I2C2_ACKNOWLEDGE_STATUS_BIT)
    {
        i2c2_object.i2cErrors++;
        I2C2_Stop(I2C2_MESSAGE_ADDRESS_NO_ACK);
    }
    else
    {
        // ACK Status is good
        // restart the bus
        I2C2_REPEAT_START_CONDITION_ENABLE_BIT = 1;

        // fudge the address so S_MASTER_SEND_ADDR works correctly
        // we only do this on a 10-bit address resend
        i2c_address = 0x00F0 | ((i2c_address >> 8) & 0x0006);

        // set the R/W flag
        i2c_address |= 0x0001;

```

```

// set the address restart flag so we do not change the address
i2c_10bit_address_restart = 1;

// Resend the address as a read
i2c2_state = S_MASTER_SEND_ADDR;
}

break;

case S_MASTER_SEND_ADDR:

/* Start has been sent, send the address byte */

/* Note:
On a 10-bit address resend (done only during a 10-bit
device read), the original i2c_address was modified in
S_MASTER_10BIT_RESTART state. So the check if this is
a 10-bit address will fail and a normal 7-bit address
is sent with the R/W bit set to read. The flag
i2c_10bit_address_restart prevents the address to
be re-written.
*/
if(i2c_10bit_address_restart != 1)
{
    // extract the information for this message
    i2c_address = p_i2c2_trb_current->address;
    pi2c_buf_ptr = p_i2c2_trb_current->pbuffer;
    i2c_bytes_left = p_i2c2_trb_current->length;
}

// check for 10-bit address
if(!I2C2_7bit && (0x0 != i2c_address))
{
    if (0 == i2c_10bit_address_restart)
    {
        // we have a 10 bit address
        // send bits<9:8>
        // mask bit 0 as this is always a write
        I2C2_TRANSMIT_REG = 0xF0 | ((i2c_address >> 8) & 0x0006);
        i2c2_state = S_MASTER_SEND_ADDR_10BIT_LSB;
    }
    else
    {
        // resending address bits<9:8> to trigger read
        I2C2_TRANSMIT_REG = i2c_address;
        i2c2_state = S_MASTER_ACK_ADDR;
        // reset the flag so the next access is ok
        i2c_10bit_address_restart = 0;
    }
}

```

```

        }

    }

else
{
    // Transmit the address
    I2C2_TRANSMIT_REG = i2c_address;
    if(i2c_address & 0x01)
    {
        // Next state is to wait for address to be acked
        i2c2_state = S_MASTER_ACK_ADDR;
    }
    else
    {
        // Next state is transmit
        i2c2_state = S_MASTER_SEND_DATA;
    }
}
break;

case S_MASTER_SEND_DATA:

// Make sure the previous byte was acknowledged
if(I2C2_ACKNOWLEDGE_STATUS_BIT)
{
    // Transmission was not acknowledged
    i2c_object.i2cErrors++;

    // Reset the Ack flag
    I2C2_ACKNOWLEDGE_STATUS_BIT = 0;

    // Send a stop flag and go back to idle
    I2C2_Stop(I2C2_DATA_NO_ACK);

}
else
{
    // Did we send them all ?
    if(i2c_bytes_left-- == 0U)
    {
        // yup sent them all!

        // update the trb pointer
        p_i2c2_trb_current++;

        // are we done with this string of requests?
        if(--i2c2_trb_count == 0)
        {
            I2C2_Stop(I2C2_MESSAGE_COMPLETE);
        }
    }
}

```

```

    }
else
{
    // no!, there are more TRB to be sent.
    //I2C2_START_CONDITION_ENABLE_BIT = 1;

    // In some cases, the slave may require
    // a restart instead of a start. So use this one
    // instead.
    I2C2_REPEAT_START_CONDITION_ENABLE_BIT = 1;

    // start the i2c request
    i2c2_state = S_MASTER_SEND_ADDR;

}

else
{
    // Grab the next data to transmit
    I2C2_TRANSMIT_REG = *pi2c_buf_ptr++;
}

break;
}

case S_MASTER_ACK_ADDR:

/* Make sure the previous byte was acknowledged */
if(I2C2_ACKNOWLEDGE_STATUS_BIT)
{

    // Transmission was not acknowledged
    i2c2_object.i2cErrors++;

    // Send a stop flag and go back to idle
    I2C2_Stop(I2C2_MESSAGE_ADDRESS_NO_ACK);

    // Reset the Ack flag
    I2C2_ACKNOWLEDGE_STATUS_BIT = 0;
}
else
{
    I2C2_RECEIVE_ENABLE_BIT = 1;
    i2c2_state = S_MASTER_ACK_RCV_DATA;
}

break;
}

case S_MASTER_RCV_DATA:

```

```

/* Acknowledge is completed. Time for more data */

// Next thing is to ack the data
i2c2_state = S_MASTER_ACK_RCV_DATA;

// Set up to receive a byte of data
I2C2_RECEIVE_ENABLE_BIT = 1;

break;

case S_MASTER_ACK_RCV_DATA:

    // Grab the byte of data received and acknowledge it
    *pi2c_buf_ptr++ = I2C2_RECEIVE_REG;

    // Check if we received them all?
    if(--i2c_bytes_left)
    {

        /* No, there's more to receive */

        // No, bit 7 is clear. Data is ok
        // Set the flag to acknowledge the data
        I2C2_ACKNOWLEDGE_DATA_BIT = 0;

        // Wait for the acknowledge to complete, then get more
        i2c2_state = S_MASTER_RCV_DATA;
    }
    else
    {

        // Yes, it's the last byte. Don't ack it
        // Flag that we will nak the data
        I2C2_ACKNOWLEDGE_DATA_BIT = 1;

        I2C2_FunctionComplete();
    }

    // Initiate the acknowledge
    I2C2_ACKNOWLEDGE_ENABLE_BIT = 1;
    break;

case S_MASTER_RCV_STOP:
case S_MASTER_SEND_STOP:

    // Send the stop flag
    I2C2_Stop(I2C2_MESSAGE_COMPLETE);
    break;

```

default:

```
// This case should not happen, if it does then  
// terminate the transfer  
i2c2_object.i2cErrors++;  
I2C2_Stop(I2C2_LOST_STATE);  
break;  
}  
}
```

```
void I2C2_FunctionComplete(void)  
{
```

```
// update the trb pointer  
p_i2c2_trb_current++;  
  
// are we done with this string of requests?  
if(-i2c2_trb_count == 0)  
{  
    i2c2_state = S_MASTER_SEND_STOP;  
}  
else  
{  
    i2c2_state = S_MASTER_RESTART;  
}
```

```
}
```

```
void I2C2_Stop(I2C2_MESSAGE_STATUS completion_code)  
{
```

```
// then send a stop  
I2C2_STOP_CONDITION_ENABLE_BIT = 1;  
  
// make sure the flag pointer is not NULL  
if (p_i2c2_current->pTrFlag != NULL)  
{  
    // update the flag with the completion code  
    *(p_i2c2_current->pTrFlag) = completion_code;  
}
```

```
// Done, back to idle  
i2c2_state = S_MASTER_IDLE;  
}
```

```
void I2C2_MasterWrite(
```

```

        uint8_t *pdata,
        uint8_t length,
        uint16_t address,
        I2C2_MESSAGE_STATUS *pflag)
{
    static I2C2_TRANSACTION_REQUEST_BLOCK trBlock;

    // check if there is space in the queue
    if (i2c2_object.trStatus.s.full != true)
    {
        I2C2_MasterWriteTRBBuild(&trBlock, pdata, length, address);
        I2C2_MasterTRBInsert(1, &trBlock, pflag);
    }
    else
    {
        *pflag = I2C2_MESSAGE_FAIL;
    }
}

void I2C2_MasterRead(
        uint8_t *pdata,
        uint8_t length,
        uint16_t address,
        I2C2_MESSAGE_STATUS *pflag)
{
    static I2C2_TRANSACTION_REQUEST_BLOCK trBlock;

    // check if there is space in the queue
    if (i2c2_object.trStatus.s.full != true)
    {
        I2C2_MasterReadTRBBuild(&trBlock, pdata, length, address);
        I2C2_MasterTRBInsert(1, &trBlock, pflag);
    }
    else
    {
        *pflag = I2C2_MESSAGE_FAIL;
    }
}

void I2C2_MasterTRBInsert(
        uint8_t count,
        I2C2_TRANSACTION_REQUEST_BLOCK *ptrb_list,
        I2C2_MESSAGE_STATUS *pflag)
{

```

```

// check if there is space in the queue
if (i2c2_object.trStatus.s.full != true)
{
    *pflag = I2C2_MESSAGE_PENDING;

    i2c2_object.pTrTail->ptrb_list = ptrb_list;
    i2c2_object.pTrTail->count = count;
    i2c2_object.pTrTail->pTrFlag = pflag;
    i2c2_object.pTrTail++;

    // check if the end of the array is reached
    if (i2c2_object.pTrTail == (i2c2_tr_queue + I2C2_CONFIG_TR_QUEUE_LENGTH))
    {
        // adjust to restart at the beginning of the array
        i2c2_object.pTrTail = i2c2_tr_queue;
    }

    // since we added one item to be processed, we know
    // it is not empty, so set the empty status to false
    i2c2_object.trStatus.s.empty = false;

    // check if full
    if (i2c2_object.pTrHead == i2c2_object.pTrTail)
    {
        // it is full, set the full status to true
        i2c2_object.trStatus.s.full = true;
    }
}

else
{
    *pflag = I2C2_MESSAGE_FAIL;
}

// for interrupt based
if (*pflag == I2C2_MESSAGE_PENDING)
{
    while(i2c2_state != S_MASTER_IDLE);
    {
        // force the task to run since we know that the queue has
        // something that needs to be sent
        PIR2bits.SSP2IF = true;
    }
} // block until request is complete
}

void I2C2_MasterReadTRBBuild(

```

```

I2C2_TRANSACTION_REQUEST_BLOCK *ptrb,
uint8_t *pdata,
uint8_t length,
uint16_t address)
{
    ptrb->address = address << 1;
    // make this a read
    ptrb->address |= 0x01;
    ptrb->length = length;
    ptrb->pbuffer = pdata;
}

void I2C2_MasterWriteTRBBuild(
    I2C2_TRANSACTION_REQUEST_BLOCK *ptrb,
    uint8_t *pdata,
    uint8_t length,
    uint16_t address)
{
    ptrb->address = address << 1;
    ptrb->length = length;
    ptrb->pbuffer = pdata;
}

bool I2C2_MasterQueueIsEmpty(void)
{
    return(i2c2_object.trStatus.s.empty);
}

bool I2C2_MasterQueueIsFull(void)
{
    return(i2c2_object.trStatus.s.full);
}

void I2C2_BusCollisionISR( void )
{
    // enter bus collision handling code here
}
interrupt_manager.c

#include "interrupt_manager.h"
#include "mcc.h"

void interrupt INTERRUPT_InterruptManager (void)
{
    // interrupt handler
    if(INTCONbits.PEIE == 1 && PIE1bits.BCL1IE == 1 && PIR1bits.BCL1IF == 1)
    {
        I2C1_BusCollisionISR();
    }
}

```

```

}

else if(INTCONbits.PEIE == 1 && PIE1bits.SSP1IE == 1 && PIR1bits.SSP1IF == 1)
{
    I2C1_ISR();
}

else if(INTCONbits.PEIE == 1 && PIE2bits.BCL2IE == 1 && PIR2bits.BCL2IF == 1)
{
    I2C2_BusCollisionISR();
}

else if(INTCONbits.PEIE == 1 && PIE2bits.SSP2IE == 1 && PIR2bits.SSP2IF == 1)
{
    I2C2_ISR();
}

else if(PIE0bits.TMROIE == 1 && PIR0bits.TMROIF == 1)
{
    TMRO_ISR();
}

else if(INTCONbits.PEIE == 1 && PIE1bits.TXIE == 1 && PIR1bits.TXIF == 1)
{
    EUSART_Transmit_ISR();
}

else if(INTCONbits.PEIE == 1 && PIE1bits.RCIE == 1 && PIR1bits.RCIF == 1)
{
    EUSART_Receive_ISR();
}

else if(INTCONbits.PEIE == 1 && PIE1bits.TMR2IE == 1 && PIR1bits.TMR2IF == 1)
{
    TMR2_ISR();
}

else
{
    //Unhandled Interrupt
}
}

mcc.c
#pragma config FEXTOSC = HS // FEXTOSC External Oscillator mode Selection bits->HS (crystal
oscillator) above 4 MHz
#pragma config RSTOSC = EXT1X // Power-up default value for COSC bits->EXTOSC operating per
FEXTOSC bits
#pragma config CLKOUTEN = OFF // Clock Out Enable bit->CLKOUT function is disabled; I/O or
oscillator function on OSC2
#pragma config CSWEN = ON // Clock Switch Enable bit->Writing to NOSC and NDIV is allowed
#pragma config FCMEN = ON // Fail-Safe Clock Monitor Enable->Fail-Safe Clock Monitor is enabled

// CONFIG2
#pragma config MCLRE = ON // Master Clear Enable bit->MCLR/VPP pin function is MCLR; Weak pull-
up enabled
#pragma config PWRTE = OFF // Power-up Timer Enable bit->PWRT disabled

```

```

#pragma config WDTE = OFF // Watchdog Timer Enable bits->WDT disabled; SWDTEN is ignored
#pragma config LPBOREN = OFF // Low-power BOR enable bit->ULPBOR disabled
#pragma config BOREN = ON // Brown-out Reset Enable bits->Brown-out Reset enabled, SBORN bit
ignored
#pragma config BORV = LOW // Brown-out Reset Voltage selection bit->Brown-out voltage (Vbor) set
to 2.45V
#pragma config PPS1WAY = ON // PPSLOCK bit One-Way Set Enable bit->The PPSLOCK bit can be
cleared and set only once; PPS registers remain locked after one clear/set cycle
#pragma config STVREN = ON // Stack Overflow/Underflow Reset Enable bit->Stack Overflow or
Underflow will cause a Reset
#pragma config DEBUG = OFF // Debugger enable bit->Background debugger disabled

// CONFIG3
#pragma config WRT = OFF // User NVM self-write protection bits->Write protection off
#pragma config LVP = ON // Low Voltage Programming Enable bit->Low Voltage programming enabled.
MCLR/VPP pin function is MCLR. MCLRE configuration bit is ignored.

// CONFIG4
#pragma config CP = OFF // User NVM Program Memory Code Protection bit->User NVM code
protection disabled
#pragma config CPD = OFF // Data NVM Memory Code Protection bit->Data NVM code protection
disabled

#include "mcc.h"

void SYSTEM_Initialize(void)
{
    PIN_MANAGER_Initialize();
    OSCILLATOR_Initialize();
    WDT_Initialize();
    I2C1_Initialize();
    I2C2_Initialize();
    ADC_Initialize();
    PWM5_Initialize();
    TMR2_Initialize();
    EXT_INT_Initialize();
    TMRO_Initialize();
    EUSART_Initialize();
}

void OSCILLATOR_Initialize(void)
{
    // NOSC EXTOSC; NDIV 1;
    OSCCON1 = 0x70;
    // CSWHOLD may proceed; SOSCPWR Low power; SOSCBE crystal oscillator;
    OSCCON3 = 0x00;
    // LFOEN disabled; ADOEN disabled; SOSCEN disabled; EXTOEN disabled; HFOEN disabled;
}

```

```

OSCEN = 0x00;
// HFFRQ 32_MHz;
OSCFRQ = 0x07;
// HFTUN 0;
OSCTUNE = 0x00;
}

void WDT_Initialize(void)
{
    // WDTPS 1:65536; SWDTEN OFF;
    WDTCON = 0x16;
}

```

pin_manager.c

```

#include <xc.h>
#include "pin_manager.h"
#include "stdbool.h"

void PIN_MANAGER_Initialize(void)
{
    /**
     * LATx registers
     */
    LATA = 0x00;
    LATB = 0x00;
    LATC = 0x00;

    /**
     * TRISx registers
     */
    TRISA = 0x37;
    TRISB = 0xF0;
    TRISC = 0x05;

    /**
     * ANSELx registers
     */
    ANSELC = 0xFB;
    ANSELB = 0x00;
    ANSELA = 0x33;

    /**
     * WPUX registers
     */
    WPUB = 0x00;
    WPUA = 0x00;
}

```

```

WPUC = 0x00;

< /**
ODx registers
*/
ODCONA = 0x00;
ODCONB = 0x00;
ODCONC = 0x00;

bool state = GIE;
GIE = 0;
PPSLOCK = 0x55;
PPSLOCK = 0xAA;
PPSLOCKbits.PPSLOCKED = 0x00; // unlock PPS

SSP1DATPPSbits.SSP1DATPPS = 0x0C; //RB4->MSSP1:SDA1;
SSP1CLKPPSbits.SSP1CLKPPS = 0x0E; //RB6->MSSP1:SCL1;
RXPPSbits.RXPPS = 0x12; //RC2->EUSART:RX;
RB4PPSbits.RB4PPS = 0x19; //RB4->MSSP1:SDA1;
RC3PPSbits.RC3PPS = 0x02; //RC3->PWM5:PWM5;
RB6PPSbits.RB6PPS = 0x18; //RB6->MSSP1:SCL1;
RB5PPSbits.RB5PPS = 0x1B; //RB5->MSSP2:SDA2;
RC1PPSbits.RC1PPS = 0x14; //RC1->EUSART:TX;
RB7PPSbits.RB7PPS = 0x1A; //RB7->MSSP2:SCL2;
INTPPSbits.INTPPS = 0x02; //RA2->EXT_INT:INT;
SSP2DATPPSbits.SSP2DATPPS = 0x0D; //RB5->MSSP2:SDA2;
SSP2CLKPPSbits.SSP2CLKPPS = 0x0F; //RB7->MSSP2:SCL2;

PPSLOCK = 0x55;
PPSLOCK = 0xAA;
PPSLOCKbits.PPSLOCKED = 0x01; // lock PPS

GIE = state;
}

void PIN_MANAGER_IOC(void)
{
}

pwm5.c
#include <xc.h>
#include "pwm5.h"

```

```

/**
Section: PWM Module APIs
*/
void PWM5_Initialize(void)
{
    // Set the PWM to the options selected in the MPLAB(c) Code Configurator.
    // PWM5POL active_lo; PWM5EN enabled;
    PWM5CON = 0x90;

    // PWM5DCH 39;
    PWM5DCH = 0x27;

    // PWM5DCL 3;
    PWM5DCL = 0xC0;

    // Select timer
    PWMTMRSbits.P5TSEL = 0x1;
}

void PWM5_LoadDutyValue(uint16_t dutyValue)
{
    // Writing to 8 MSBs of PWM duty cycle in PWMDCH register
    PWM5DCH = (dutyValue & 0x03FC)>>2;

    // Writing to 2 LSBs of PWM duty cycle in PWMDCL register
    PWM5DCL = (dutyValue & 0x0003)<<6;
}
tmr0.c

#include <xc.h>
#include "tmr0.h"

/**
Section: TMRO APIs
*/
extern bool tmr0_expired;

void (*TMRO_InterruptHandler)(void);

void TMRO_Initialize(void)
{
    // Set TMRO to the options selected in the User Interface

    // T0OUTPS 1:1; TOEN disabled; T016BIT 8-bit;

```

```

T0CON0 = 0x00;

// T0CS TOCKI_PIN; TOCKPS 1:1; T0ASYNC synchronised;
T0CON1 = 0x00;

// TMROH 19;
TMROH = 0x13;

// TMROL 0;
TMROL = 0x00;

// Clear Interrupt flag before enabling the interrupt
PIR0bits.TMROIF = 0;

// Enabling TMRO interrupt.
PIE0bits.TMROIE = 1;

// Set Default Interrupt Handler
TMRO_SetInterruptHandler(TMRO_DefaultInterruptHandler);

// Start TMRO
TMRO_StartTimer();
}

void TMRO_StartTimer(void)
{
    // Start the Timer by writing to TMROON bit
    T0CON0bits.TOEN = 1;
}

void TMRO_StopTimer(void)
{
    // Stop the Timer by writing to TMROON bit
    T0CON0bits.TOEN = 0;
}

uint8_t TMRO_Read8bitTimer(void)
{
    uint8_t readVal;

    // read Timer0, low register only
    readVal = TMROL;

    return readVal;
}

void TMRO_Write8bitTimer(uint8_t timerVal)
{

```

```

// Write to Timer0 registers, low register only
TMR0L = timerVal;
}

void TMRO_Load8bitPeriod(uint8_t periodVal)
{
    // Write to Timer0 registers, high register only
    TMR0H = periodVal;
}

void TMRO_ISR(void)
{
    // clear the TMRO interrupt flag
    PIR0bits.TMROIF = 0;
    // ticker function call;
    // ticker is 1 -> Callback function gets called every time this ISR executes
    TMRO_CallBack();

    // add your TMRO interrupt custom code
}

void TMRO_CallBack(void)
{
    // Add your custom callback code here
    tmr0_expired = true;
    if(TMRO_InterruptHandler)
    {
        TMRO_InterruptHandler();
    }
}

void TMRO_SetInterruptHandler(void* InterruptHandler){
    TMRO_InterruptHandler = InterruptHandler;
}

void TMRO_DefaultInterruptHandler(void){
    // add your TMRO interrupt custom code
    // or set custom function using TMRO_SetInterruptHandler()
}

tmr2.c

#include <xc.h>
#include "tmr2.h"

/**
 * Section: Global Variables Definitions
 */
extern bool tmr2_expired;

```

```

void (*TMR2_InterruptHandler)(void);

/**
Section: TMR2 APIs
*/
void TMR2_Initialize(void)
{
    // Set TMR2 to the options selected in the User Interface

    // T2CKPS 1:1; T2OUTPS 1:1; TMR2ON off;
    T2CON = 0x00;

    // PR2 199;
    PR2 = 0xC7;

    // TMR2 0;
    TMR2 = 0x00;

    // Clearing IF flag before enabling the interrupt.
    PIR1bits.TMR2IF = 0;

    // Enabling TMR2 interrupt.
    PIE1bits.TMR2IE = 1;

    // Set Default Interrupt Handler
    TMR2_SetInterruptHandler(TMR2_DefaultInterruptHandler);

    // Start TMR2
    TMR2_StartTimer();
}

void TMR2_StartTimer(void)
{
    // Start the Timer by writing to TMRxON bit
    T2CONbits.TMR2ON = 1;
}

void TMR2_StopTimer(void)
{
    // Stop the Timer by writing to TMRxON bit
    T2CONbits.TMR2ON = 0;
}

uint8_t TMR2_ReadTimer(void)
{
    uint8_t readVal;

```

```

readVal = TMR2;

    return readVal;
}

void TMR2_WriteTimer(uint8_t timerVal)
{
    // Write to the Timer2 register
    TMR2 = timerVal;
}

void TMR2_LoadPeriodRegister(uint8_t periodVal)
{
    PR2 = periodVal;
}

void TMR2_ISR(void)
{
    // clear the TMR2 interrupt flag
    PIR1bits.TMR2IF = 0;

    // ticker function call;
    // ticker is 1 -> Callback function gets called everytime this ISR executes
    TMR2_CallBack();
}

void TMR2_CallBack(void)
{
    // Add your custom callback code here
    // this code executes every TMR2_INTERRUPT_TICKER_FACTOR periods of TMR2
    tmr2_expired = true;
    if(TMR2_InterruptHandler)
    {
        TMR2_InterruptHandler();
    }
}

void TMR2_SetInterruptHandler(void* InterruptHandler){
    TMR2_InterruptHandler = InterruptHandler;
}

void TMR2_DefaultInterruptHandler(void){
    // add your TMR2 interrupt custom code
    // or set custom function using TMR2_SetInterruptHandler()
}

```

3.6.0 Mechanical Design

Figure 63 shows the chassis of the robot that all the other hardware will be mounted to.

The chassis assembly consists of two octagons, two horizontal support rings, a single vertical support ring, and internal cross supports. The octagons are constructed from 3"x1.5" 0.125" wall aluminum tubing such that the total outside width of the octagon is 24.3". Each section of the octagon has slots milled in to allow the suspension bearing blocks to be adjusted. Tapped holes are also on the exterior of the octagon sections so that the top and bottom cover of the robot can be mounted and secured. The horizontal support ring made from 0.25" aluminum plate has a diameter of 31" and the inside is cut such that the octagon sits flush to the cut edge. Slots are also cut into the horizontal support ring to allow the bearing suspension blocks to be adjusted. The vertical support ring is constructed from 2"x0.25" aluminum bar rolled into a 31" diameter circle. The spacing provided by the vertical support ring allows for sensors to be mounted between the horizontal support rings as well as clearance for the bearing suspension blocks mounting hardware. The internal cross supports are used to mount the gearbox of the locomotion system. The matching cross supports are cut such that the output shaft of the gearbox protrudes into the center of the trapezoidal openings of either side of the chassis. Holes are also cut to allow for mounting hardware clearance and gearbox bearing clearance. (MJH)

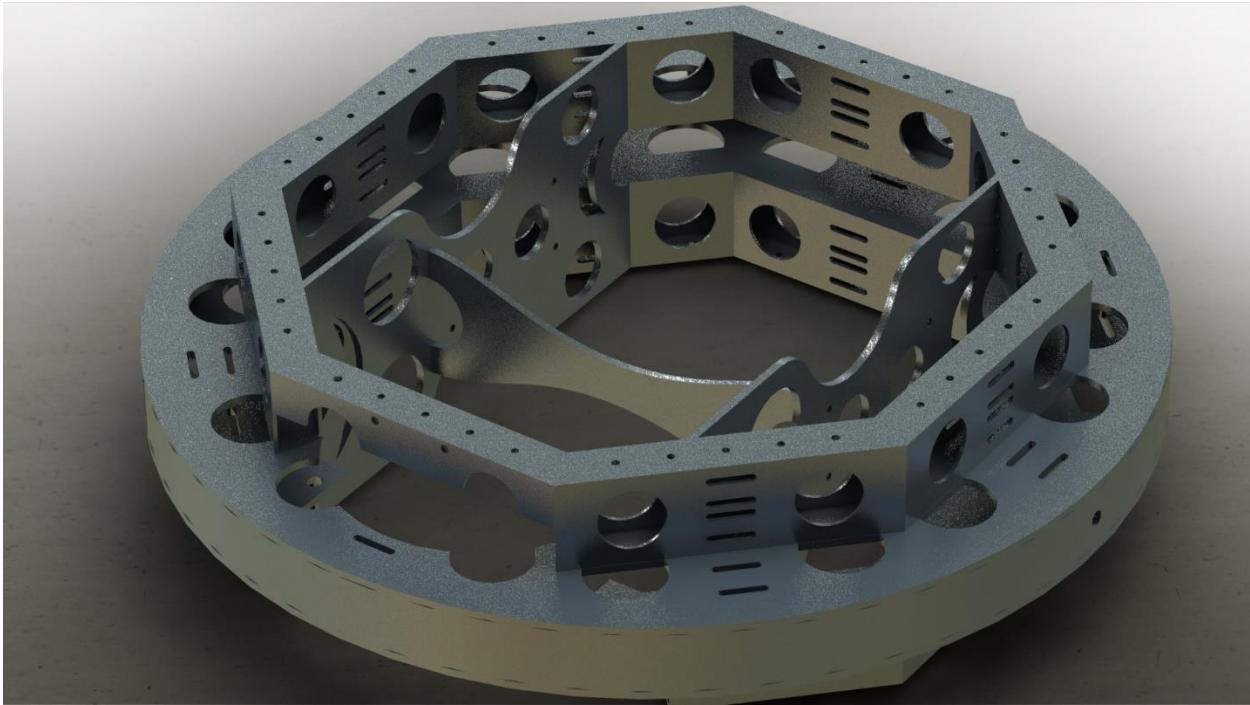


Figure 63: Robot Chassis

The weapon suspension bearing block can be seen in Figure 64. This assembly is used to suspend the weapon system on the robot chassis and to allow for the weapon to rotate around the stationary chassis. The guide rails of the weapon assembly will sit upon the bearing thus allowing the weapon assembly to rotate. The weapon suspension bearing blocks are to be mounted onto the chassis where the slots are cut allowing for adjustment of the resulting diameter from the suspension bearing blocks bearings to match that of the weapon assembly guide rail's diameter. The mounting of the suspension bearing block to a section of the chassis can be seen in Figure 65. (MJH)

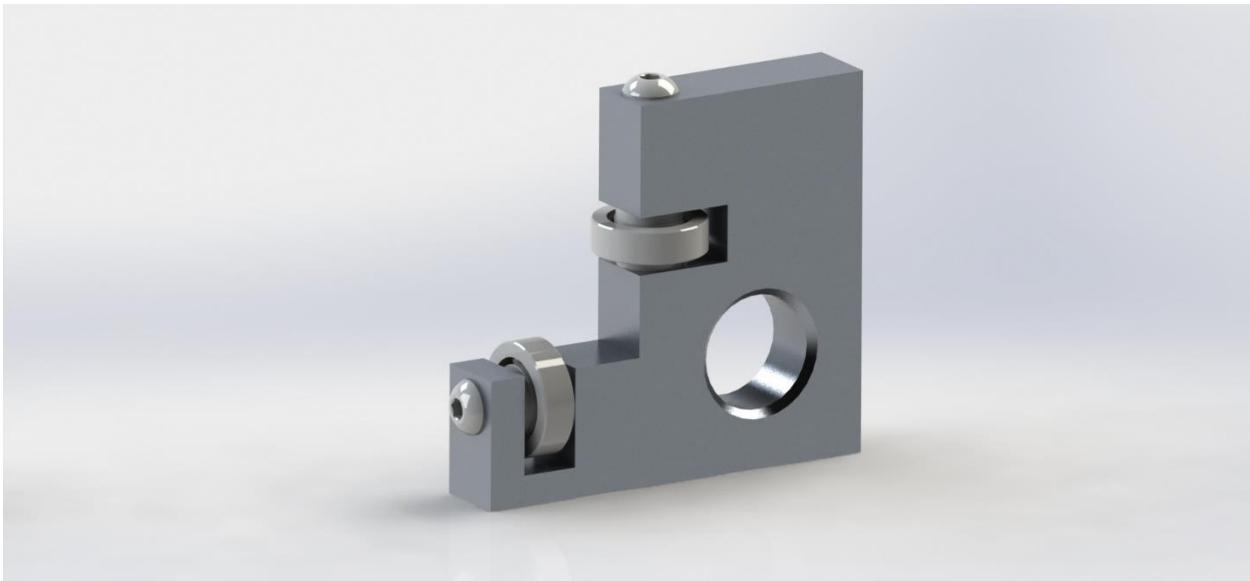


Figure 64: Weapon Suspension Bearing Block Model

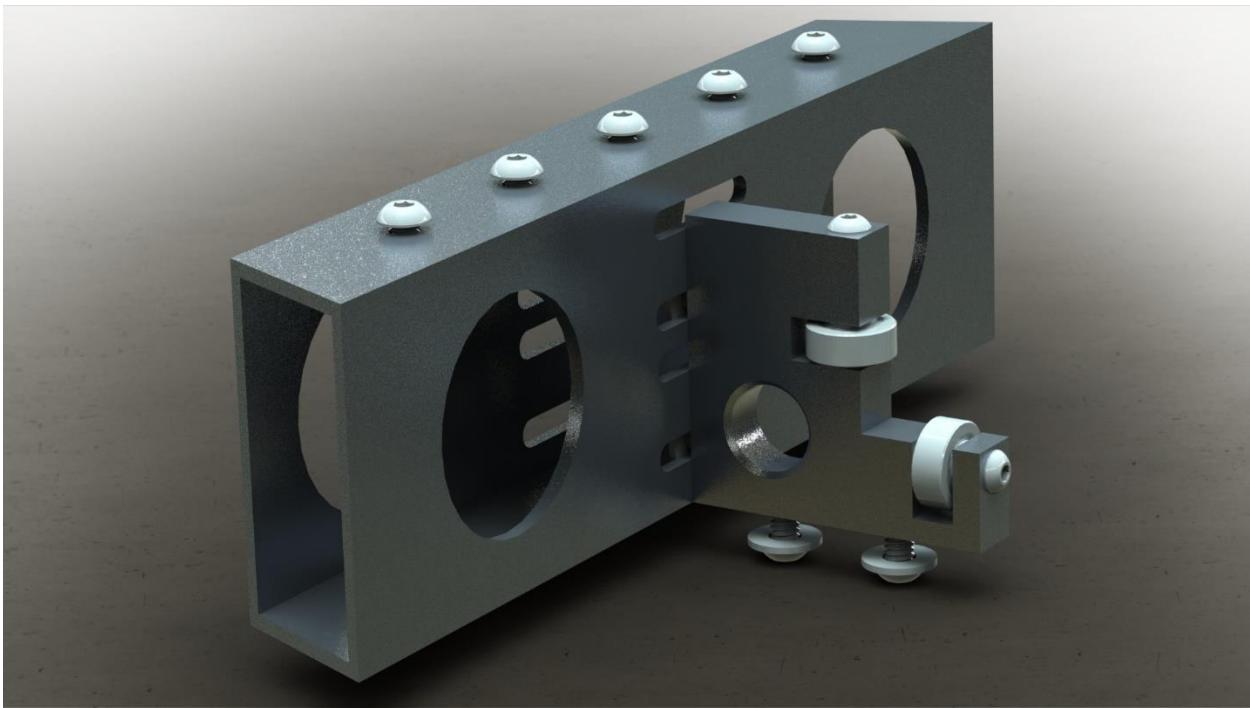


Figure 65: Weapon Suspension Bearing Block Octagon Section Assembly Model

The weapon assembly as seen in Figure 66 will revolve around the stationary chassis driven by a dedicated motor. This assembly is constructed from four main components. The outer drum with a diameter of 33", two inner guide rails with an outside diameter of 32" and

inside diameter of 28", the teardrop shaped wings, and the hammer faces. The guide rails ride upon the bearings of the bearing block assembly and thus suspend the weapon assembly on the chassis of the robot. The outer drum is used as the central mounting piece for the rest of the assembly. The wings that extend beyond the outer drum's outside diameter are used to mount the hammer face plates. The wings are stacked to 8" in total height thus allowing a 0.25" clearance between the top and bottom most parts of the weapon assembly. This is done because the floor of the arena the robot will be competing in can be uneven. This results in a decrease of likelihood that the corner of a wing will catch a section of the arena's floor. The hammer face plate is the component that will inflict damage onto the other competing robot. This face plate is made from steel rather than aluminum because of the extreme forces that the plate will endure during an impact. (MJH)

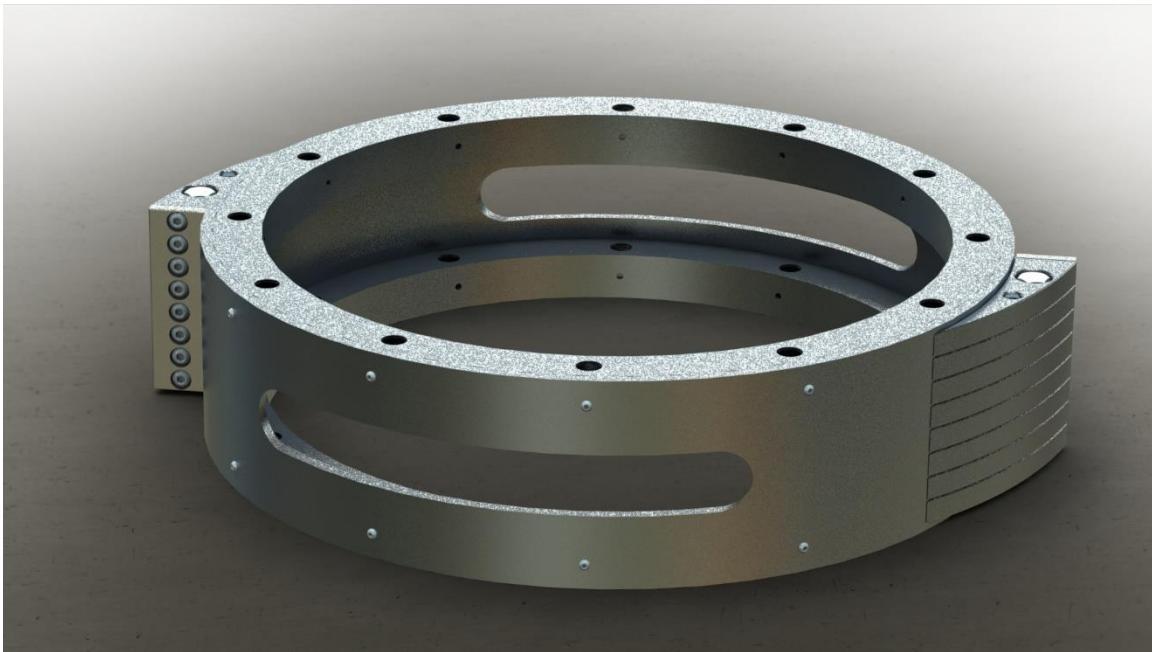


Figure 66: Weapon Model

To protect the batteries and to allow for easier removal for charging the custom battery enclosure is Figure 67 is used. The battery enclosure contains 8 8s lithium polymer battery

packs. Two of these packs are wired in parallel to provide power to the locomotion system and all the circuit boards within the robot. The remaining 6 packs are connected 3 in series and the dual 3 series backs in parallel. This configuration is known as a 24s2p pack because of the 24 individual cells in series and the 2 in parallel. The battery enclosure also contains foam to cushion the batteries from vibration and high g-force scenarios. A rubber gasket is used around the slot the wires protrude from to protect the wires from being cut on the sharp metal edge.

(MJH)

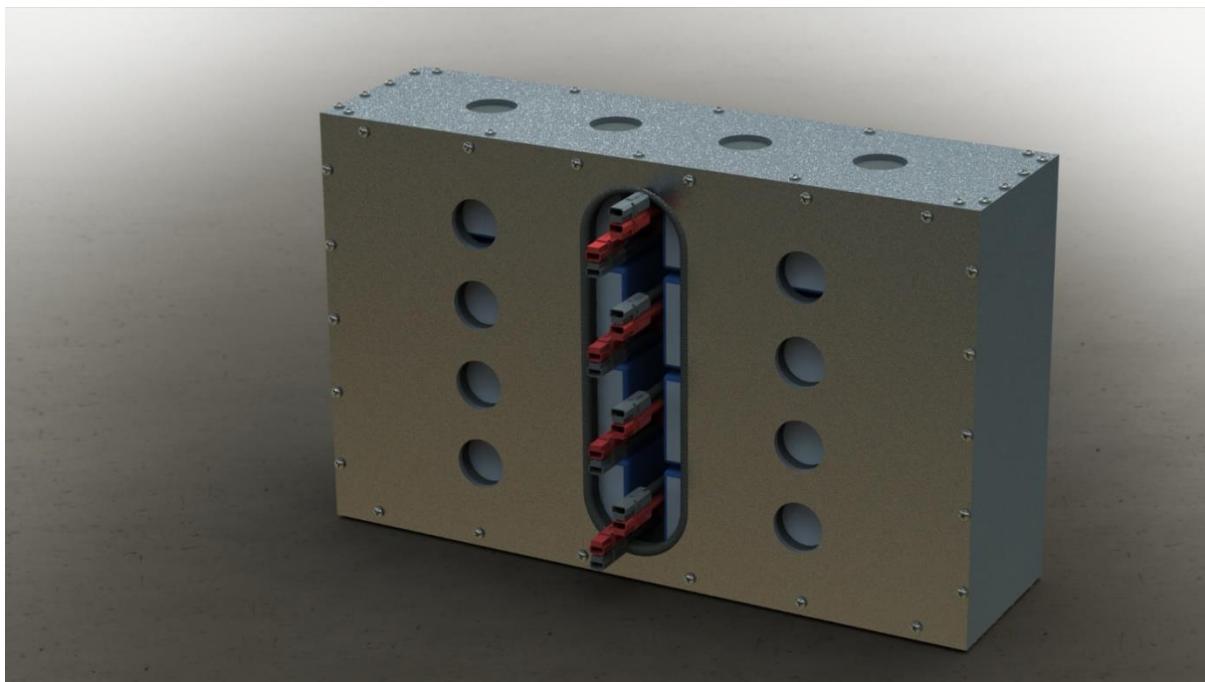


Figure 67: Battery Enclosure Model

Figure 68 shows the complete assembly of the robot. The battery enclosure module is designed so that it can be removed from the robot. This allows for safer and easier charging of the batteries. The locomotion wheels used are 10" diameter while the total height of the robot with cover and without wheels is 8.66". This results in a clearance of 0.67" between the ground and the covers. A steel skid bar is used to lower this clearance down to 0.3575" as well as

provide a harder material to be used as the point of contact between the ground and the robot.

(MJH)

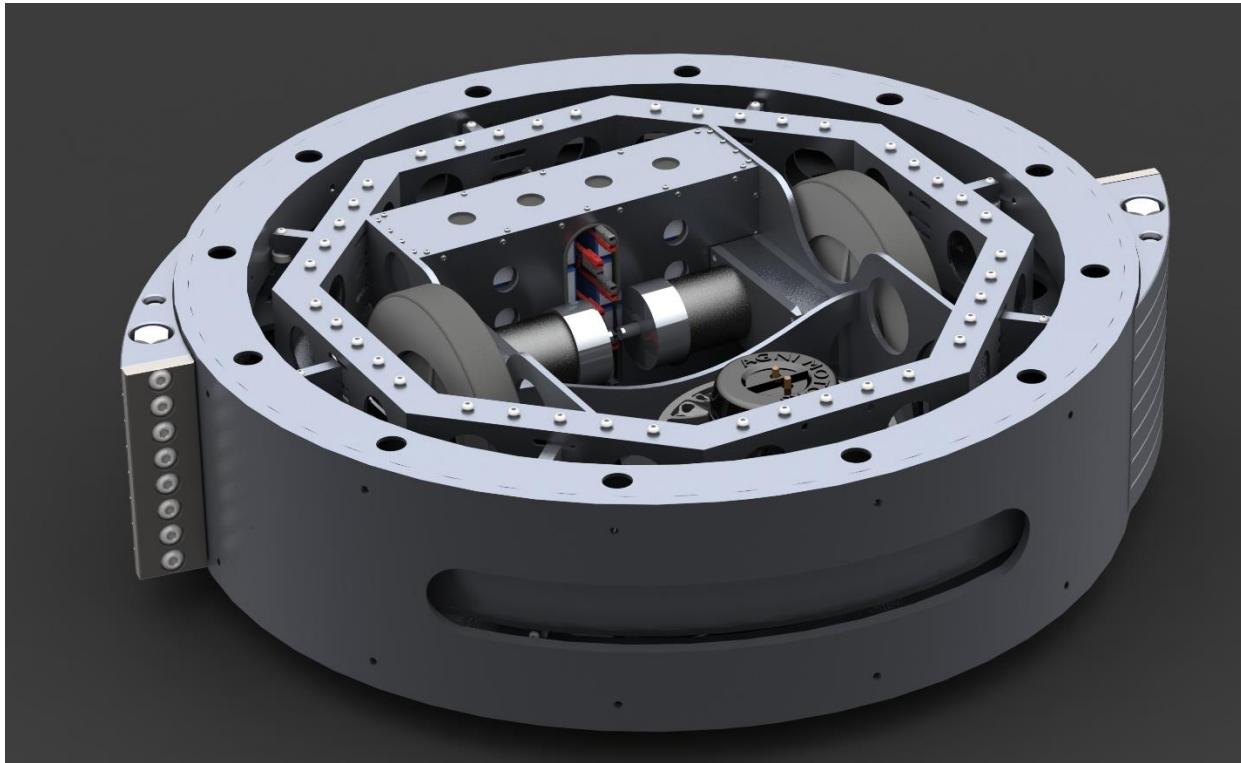


Figure 68: Robot Assembly Model

3.6.1 Mechanical Design Implementation and Revisions

Mounting of the power switches, distribution blocks, and some of the boards was not considered in the original design of the robot. Most of the mounting for this hardware was done with the installation of hinged panels within the chassis of the robot. The hinge action on the panels allowed for easier access to the electronics while still protecting them.

Figure 69 is a picture of the underside of the panel the power switches, distribution blocks, and DC-DC converter are mounting to. The position of the switches was space limited such that the terminals of the switches had to fit in between the bush assembly of the Ampflow

A28-150 locomotion motors. The position of the Anderson distribution block was selected such that the cross support of the chassis didn't interfere with the insertion and removal of Anderson connectors. The placement of the other three distribution blocks needed to avoid shorting across the cross support of the chassis. The DC-DC converter's position was selecting from the only remaining space on the panel. Figure 70 shows the panel mounted on the robot while still being opened because of the hinging action.

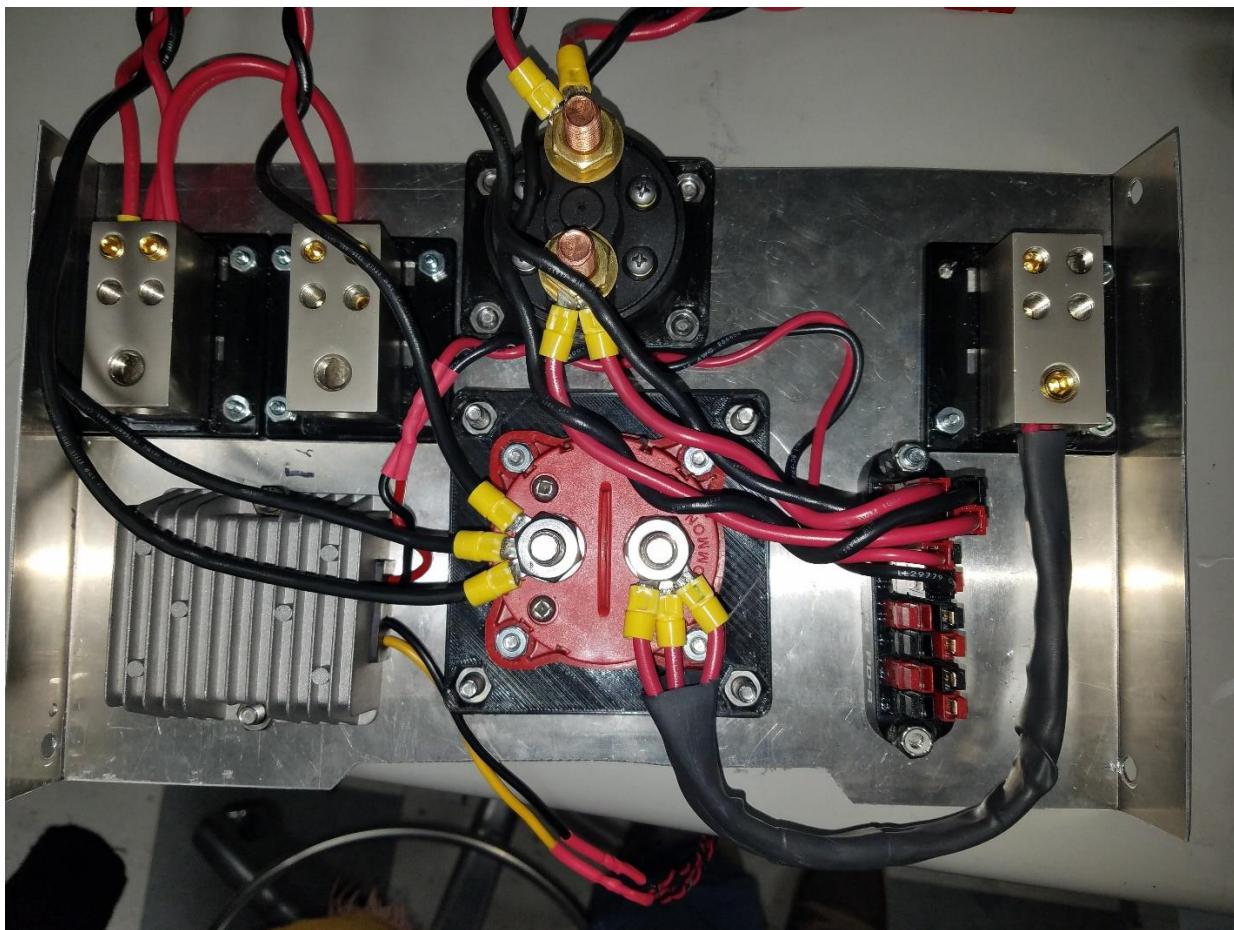


Figure 69: Power Panel Underside

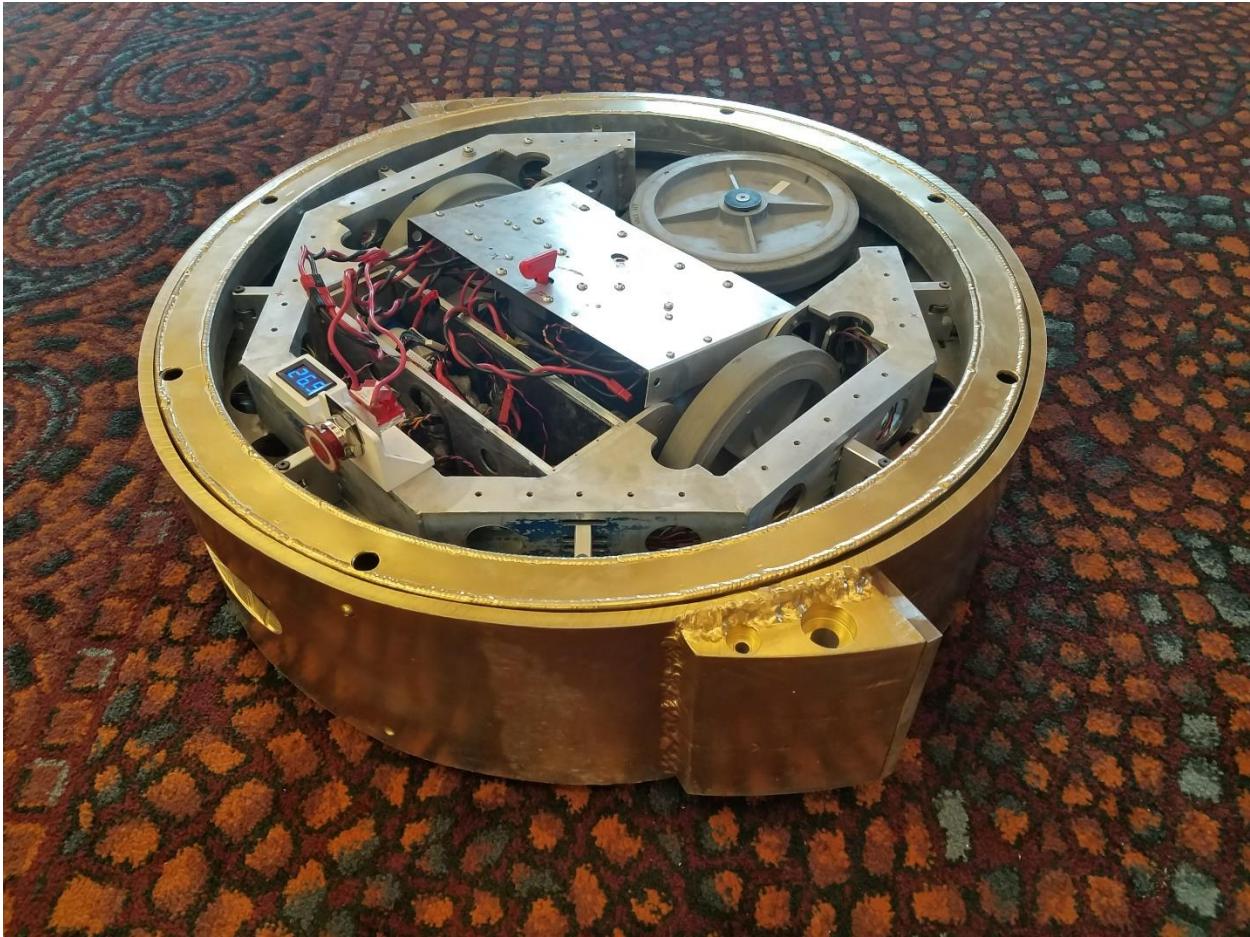


Figure 70: Power Distribution Panel Implementation

Figure 71 shows a picture of the underside of the panel the MDBs and the WMCB were mounted to. The panel has an omega shaped bracket on it that the enclosure that houses the WMCB in mounted to. The MDBs are secured to the panel with L-brackets on either side that are bolted to the panel and enclosure of the MDBs. Figure 72 shows the panel mounted onto the robot. Note the notch cut from the panel exposing some of the enclosure of the MDBs. This was done to allow for ease of access to the terminal blocks on the MDBs.



Figure 71: MDB and WMCB Panel Underside

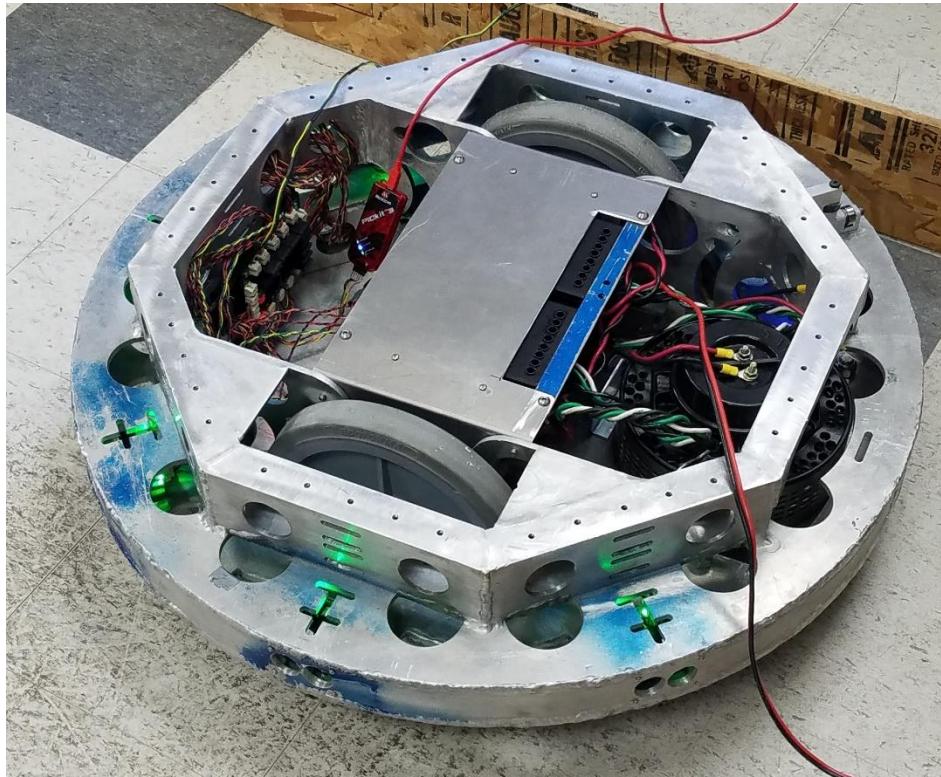


Figure 72: MDB & WMCB Panel Implementation

3.7.0 Design Calculations

All final design analysis and calculations will be shown in detail in this section. Design calculations have been done for the following circuit boards: main control board (MCB), motor driver board (MDB), weapon motor controller board (WMCB), and proximity sensor board (PSB). (MPH)

3.7.1 Main Control Board

The main control board has three important functions for the robot. One is to send the required regulated voltage to each of the other boards excluding the motor driver board, communicate to each board and identify objects and walls from the scanned data from the proximity sensors.

The main control board will send a regulated 9 volt to each of the proximity sensor boards and weapon motor controller board by a switch regulator and also use a switch 5 volt regulator to drop the unregulated 24 volt to 5 volt for the PIC32 micro controller and 3.3 volts for the IMU. Since there is many hardware components and voltage regulators, power dissipation must be calculated to determine if there would be a need for heat sinks. For linear voltage regulator (3.3 volts) the equation below to calculate the power dissipation:

$$P_{Diss} = (V_{in} - V_{out}) * \sum I_{total} \quad (1)$$

For the switch voltage regulators (9V and 5V) the equation below to calculate the power dissipation:

$$P_{Diss} = \frac{(V_{out}) * \sum I_{total}}{\% Efficiency} \quad (2)$$

There are a total of four voltage regulators on the main control board two 9 volt switch regulators, one 5 volt switch regulator and one linear 3.3 volt linear regulator. For hardware there are fifteen hardware components total on the main control board. The two voltage translators ($P_{Diss,vt}$) has a total power dissipation of 1000mW (given by data sheet). The IMU (I_{IMU}) has a current of 1.1mA for the gyroscope and 240uA for the accelerometer. The maximum supply current drawn, (I_{RS-485}), for the RS485 transceiver (U15 in PSB schematic) is 500 μ A according to the data sheet. The maximum current consumed by the PIC32MZ microcontroller (I_{PIC}), is 150mA. The two I/O expanders (I_{io}) total current consumes 100mA . The current drawn from the 7-segment display (P_{Disp}) has 75mW (given by data sheet) and the sum of the 6 dip-switches (I_{DIP}) of 1.98mA. The last maximum current consumed is the Buffer

Line Driver for the Micro-SD, the Buffer Line Driver (I_{Driver}) draws a maximum of 10mA.

Therefore, the following equation to determine the power dissipated in the 3.3V linear regulator:

$$P_{Diss} = (5 - 3.3)[V] * (I_{IMU} + I_{PIC} + I_{io} + I_{DIP} + I_{Driver} + I_{RS-485}) + P_{Dissi,Vt} + P_{Disp} \quad (3)$$

$$P_{Diss} = (1.7)[V] * ((1.1 + 150 + 100 + 1.98 + 10)[mA] + 500 \mu A) + (1000 + 75)[mW] \quad (4)$$

$$P_{Diss} = 1.52 [W] \quad (5)$$

For the switch regulators, the data sheet for the 9-volt regulator specifies % Efficiency of 96%.

Thus, following equation to determine the power dissipated in the 9V switch regulator:

$$P_{Diss} = \frac{(V_{out})(I_{IMU} + I_{PIC} + I_{io} + I_{DIP} + I_{Driver} + I_{RS-485}) + P_{Dissi,Vt} + P_{Disp}}{\% \text{Efficiency}} \quad (5)$$

$$P_{Diss} = \frac{(9)[V]*((1.1 + 150 + 100 + 1.98 + 10)[mA] + 500 \mu A) + (1000 + 75)[mW]}{.96} = 3.59 [W] \quad (6)$$

For the 5-volt regulator, the data sheet specifies % Efficiency of 87%. Thus, following equation to determine the power dissipated in the 5V switch regulator:

$$P_{Diss} = \frac{(5)[V]*((1.1 + 150 + 100 + 1.98 + 10)[mA] + 500 \mu A) + (1000 + 75)[mW]}{.87} = 2.75 [W] \quad (7)$$

Once the Power dissipated in found the temperature of the linear voltage regulator is determined by the following equation:

$$T_C = P_D * \theta_{JA} \quad (8)$$

The thermal junction to ambient resistance of the 3.3V linear regulator, $\theta_{JA}=62.5^{\circ}C/W$, the temperature of the linear voltage regulator is determined by the equation:

$$T_{\circ C} = 1.52W * 62.5 \frac{\circ C}{W} = 95 \circ C \quad (9)$$

Therefore, no heat sink will be required since the operating temperature range is $-40^{\circ}C - 125^{\circ}C$. For the Switch regulators the following equation will be used to find the temperature of the switch voltage regulator:

$$R_{Thermal Impedance} = \frac{T_{max} - T_{\circ C}}{P_{Diss}} \quad (10)$$

$$T_{\circ C} = T_{max} - R_{Thermal Impedance} * P_{Diss} \quad (11)$$

The 5V switch regulator operating temperature max, $T_{max} = 85^{\circ}C$, and the thermal impedance is $R_{Thermal Impedance} = 10^{\circ}C/W$. The temperature of the linear voltage regulator is determined by the equation:

$$T_{\circ C} = 85 - 10 * 2.75 = 57.5^{\circ}C \quad (12)$$

Therefore, no heat sink will be required since the operating temperature range is $-40^{\circ}C - 85^{\circ}C$. The 9V switch regulator operating temperature max, $T_{max} = 125^{\circ}C$, and the thermal impedance is $R_{Thermal Impedance} = 10^{\circ}C/W$. The temperature of the linear voltage regulator is determined by the equation:

$$T_{\circ C} = 125 - 10 * 3.59 = 85.5^{\circ}C \quad (13)$$

Therefore, a heat sink will be required since the operating temperature range is $-40^{\circ}C - 85^{\circ}C$. (HL)

When the robot is trying to identify a wall or an object the current ideology to performing this is to analyze the linear region of data that is scanned by the proximity sensors. When the

scanned area by the Lidar sensors is perpendicular towards a wall as shown in Figure 73. The blue circle represents the robot, the red represents the cone projection from the Lidar sensor, the purple is the object and the black dashed surface is the wall. The image shown on the left is the Lidar without the object interfering and the image on the right is with an object offsetting the sensors. When the robot turns, the sensor will scan in a sweeping motion and display a graph as shown in Figure 74. Since the robot is sweeping clockwise the farthest point of each end will have the longest distance and will decrease as it approaches the center. Since the sweep is in a clockwise motion the graph will have a similar shape as the function:

$$f(x) = y = x^2 \quad (1)$$

Where y is the distance measured and x is the time it was scanned. When the function is differentiated twice as shown:

$$\frac{d^2y}{dx^2} f(x) = y'' = 2 \quad (2)$$

The function is now a linear equation with a horizontal line as shown on the image to the left on Figure 75. However, if the scanned image did have some interference like an object, it will be shown as the image on the right on Figure 75. Since the wall is a linear region with a constant slope, the robot will indicate the scan region as a wall if the function does have a constant value for the whole-time duration when the Lidar sensor was active. Otherwise there will be outliers in the function as shown in right image of Figure 75, thus the robot will indicate there is an object present.(HL)

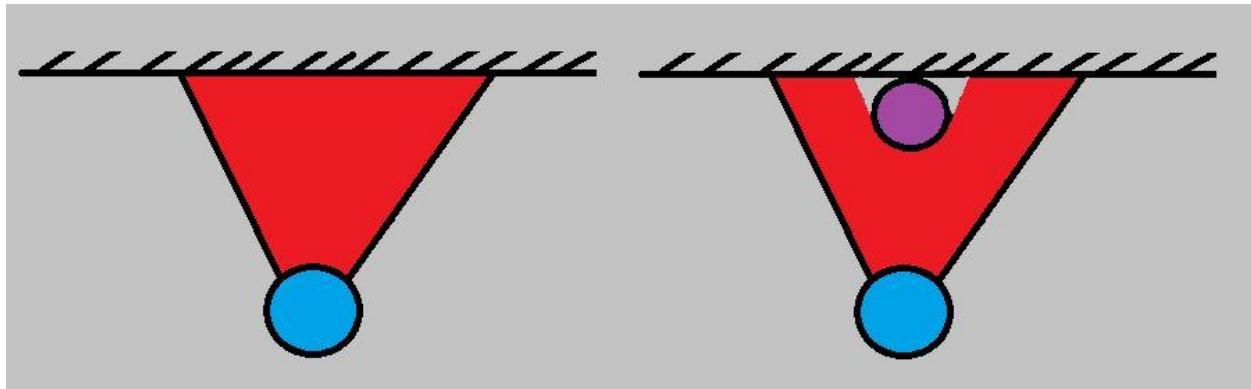


Figure 73: Robot Direct Scan

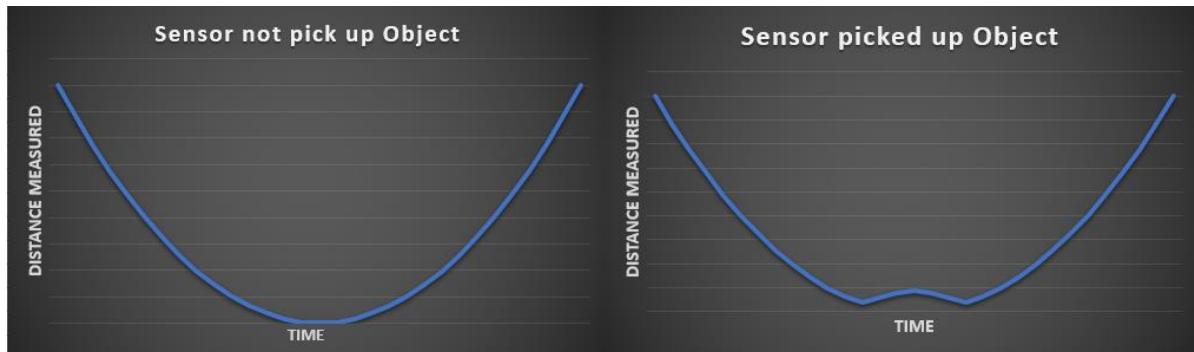


Figure 74: Graphical Interpretation of Direct scan

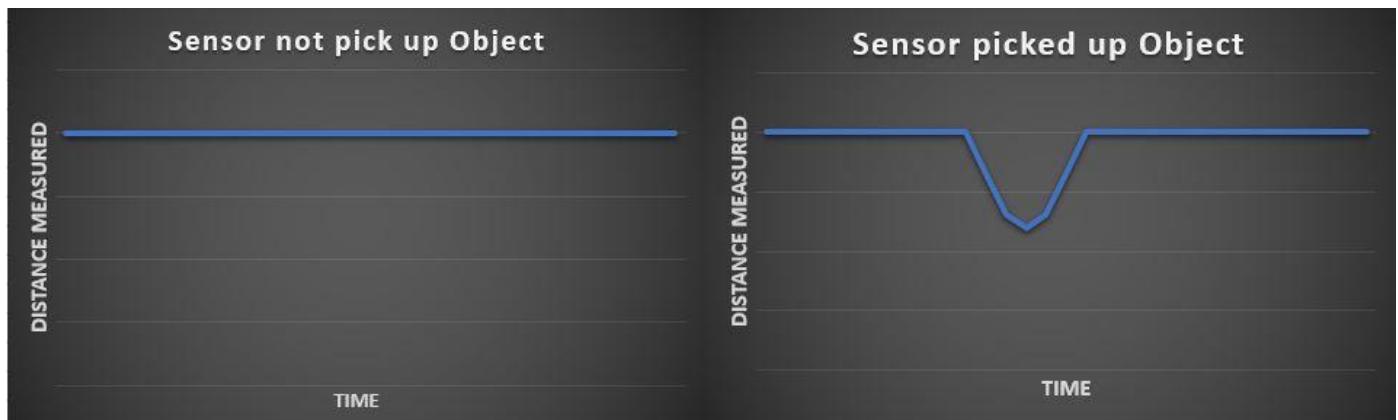


Figure 75: Graphical Interpretation of scanned information differentiated twice

The main control board will communicate to all other boards through RS485

communication. The RS485 transceiver chosen has a data rate, R_{RS485} , of 2.5 Mbps. The data

rate in bits per second will be defined as R_{BPS} . The following equation shows that 1 bit will be transmitted every 2.5 μ s:

$$R_{BPS} = \left(\frac{2.5 \text{ Mbits}}{\text{sec}} \right) \left(\frac{10^{-6} \text{ sec}}{2.5 \mu\text{s}} \right) = \frac{1 \text{ bit}}{2.5 \mu\text{s}} \quad (1)$$

Therefore, if one byte of information were to be transmitted to another board, it will take approximately 20 μ s excluding instruction execution time in the microcontroller. (MPH)

The main control board instruction execution time will be another main factor including the transceiver transmission rate. The exact amount of time it will take to transmit a message to one of the other boards cannot be determined at this point. The c program code for the main control board must be written in order to determine the amount of instructions the microcontroller needs to execute in assembly code. Once the amount of instructions are determined, the instruction clock frequency, F_{CY} , will be converted to the instruction clock period, τ_{CY} . According to the PIC32MZ2048EFG064-I/PT datasheet, the instruction clock period will be 4 ns. (MPH)

A total number of data bits will need to be transmitted to each board such as the parsed data to identify which board the data is for. The data itself must also be transmitted along with the parsed data. The total number of bits, N_{bits} , will be multiplied by 2.5 μ s since 1 bit can be transmitted every 2.5 μ s. The total number of instructions that need to be executed will be defined as N_{INST} . The amount of time it takes for the main control board to execute its code and transmit the data to another board will be defined by the following equation:

$$\tau_{total} = (N_{Data} * 2.5 \mu\text{s}) + (N_{INST} * \tau_{CY}) \quad (2)$$

Therefore, by using equations 1 and 2, the total amount of time it takes to transmit data to one of the boards can be calculated. (MPH)

3.7.2 Motor Driver Board

The motor driver board is intended to switch high power at a relatively high frequency to an inductive load. As a result, temperature of the switching devices must be considered. The switching device chosen was the IRFS3006-7PPbF because of its high current handling capability, low drain to source resistance, and a maximum drain source voltage that is greater than twice the intended operating voltage. The motor being controlled is capable of drawing 285 amps during a stall condition. To prevent damage to the motor driver board a current sensor is used to monitor the current being drawn by the motor. By monitoring the input current, the average current being drawn by the motor can be limited. Figure 76 shows a simplified circuit of the motor controller for the purpose of approximating thermal characteristics during a stall condition. The model of the MOSFET was acquired from Infineon's SPICE Model database. Duty cycles tested include $D=0.2, 0.5$, and 0.75 and load currents tested include $I=50, 100$, and 200 amps. Because the MOSFETS are in parallel the current through a single MOSFET is $1/3$ of the current through the load during all operating conditions. The resulting average power in a single MOSFET as a function of load current and duty ratio can be seen in Table 82 and the plotted results in Figure 77. (MJH)

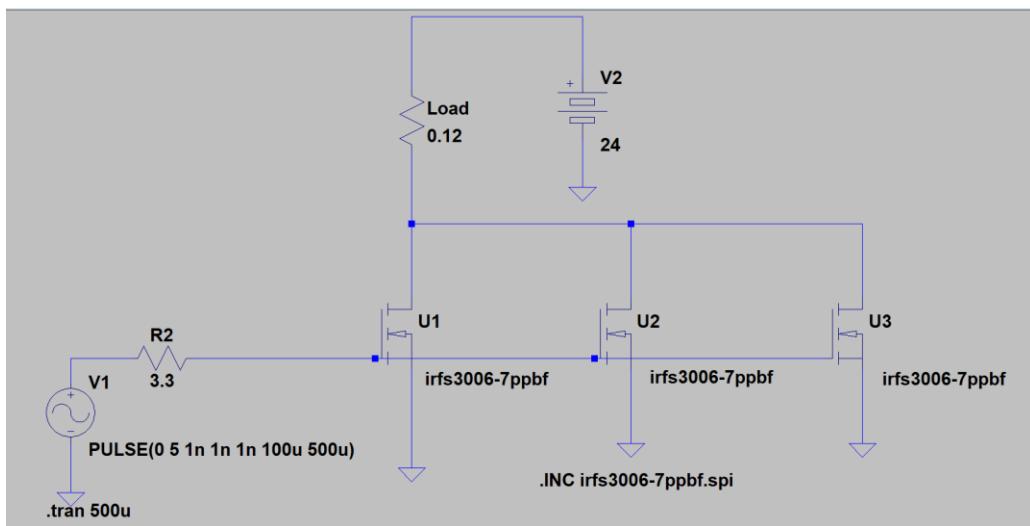


Figure 76: Motor Driver Board Thermal Characteristics Simulation

		Load Current [A]			
		50	100	150	200
Duty Cycle	D = 0.2	268.59	956.85	2080.8	3657.3
	D = 0.5	567.33	2165.9	4834.9	8619.3
	D = 0.75	816.39	3173.5	7129.9	12754
Resulting Power in single FET [mW]					

Table 82: MDB Single FET Power Table

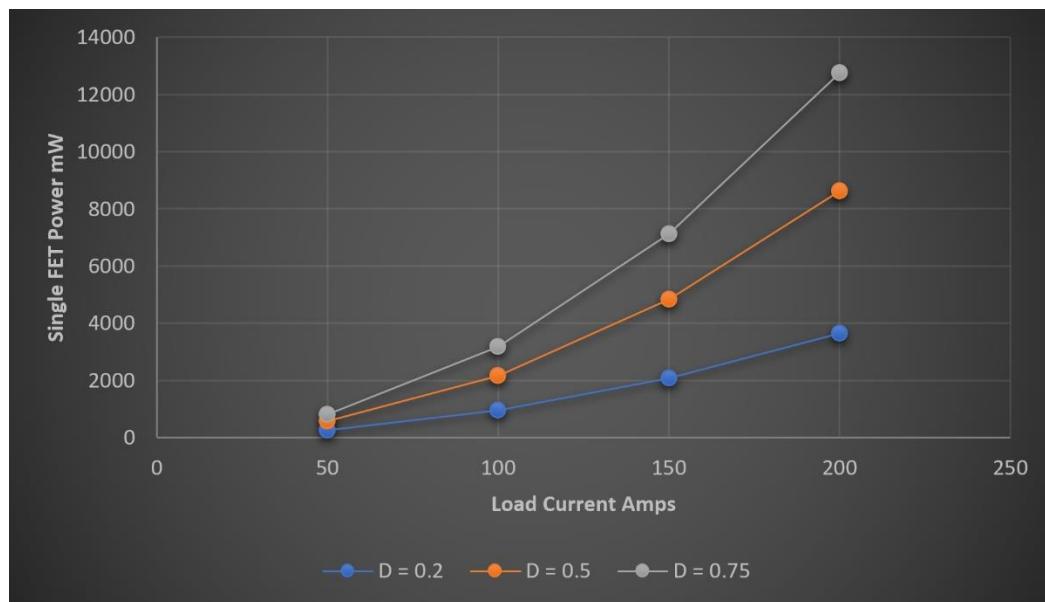


Figure 77: MDB Single FET Power Chart

To determine a sufficient thermal coefficient that a single MOSFET must have so that continuous operation can occur without overheating the power a single MOSFET dissipates needs calculated. The R_{DS} of the IRFS3006-7PPbF MOSFET being used is $1.5\text{m}\Omega$ to $2.1\text{m}\Omega$ when V_{GS} is 10V and I_D is 168 A. In the simulation V_{GS} is 5V and I_D is varied. This results in R_{DS} also varying as a function of I_D because V_{GS} is held constant at 5V. Under these operating conditions R_{DS} at load currents of 50, 100, and 150A can be seen in Table 83. To calculate the power a single MOSFET dissipates while the MOSFET is in the saturation region of operation the following equation is used. (MJH)

$$P = I^2 R_{DS}$$

Average power is calculated as the integral of power over a period divided by the time of one period. The equation is as follows.

$$\langle P \rangle = \frac{1}{T} \int_0^T P(t) dt$$

Modifying the equation to calculate the average power with a specific duty cycle is as follows.

$$\langle P \rangle = \frac{1}{T} \int_0^{DT} I^2 R_{DS} dt$$

The function being integrated is not a function of time. This means the average power can be expressed as follows.

$$\langle P \rangle = \frac{1}{T} I^2 R_{DS} DT = I^2 R_{DS} D$$

The resulting average power is not dependent on the length of time of the period. From this equation the power dissipated by a single MOSFET can be calculated. The results of the

theoretical power of a single MOSFET during varying load currents and duty ratios can be seen in Table 84. The theoretical results are also compared to the simulation results. The percent change from simulation to theoretical data can be seen in Table 85. When analyzing the trend in the data in Table 85 the percent difference decreases as load current increases and also when duty ratio increases. The difference is because the simulation is more accurate than the theoretical calculations. The theoretical calculations do not consider switching loss. As the duty ratio increases the percent of on time also increases. This results in the switching loss contributing less to the total power dissipated by a single MOSFET. The waveform of power in a single MOSFET can be seen in Figure 78. The waveforms of the spikes in power during switching can be seen in Figure 79 and Figure 80. As the duty ratio decreases the spikes during change in states of a MOSFET become more of a contributing factor to the average power. The power of a MOSFET also increases as load current increases which means that the contribution of the spikes is less significant as load current increases. This is consistent with the data seen in Table 84. As load current and duty ratio increases the simulation data approaches the same results of the theoretical data. Because the majority of heat will be generated during high duty ratio and high load current situations the power of a MOSFET can be approximated as $\langle P \rangle = I^2 R_{DS} D$ during high power conditions. (MJH)

	Load Current [A]			
	50	100	150	200
Rds [mOhm]	3.6	3.67	3.73	3.8

Table 83: MDB Rds at Load Currents Table

		Resulting Power in single FET [mW]			
	Load Current [A]	50	100	150	200
	Rds [mOhm]	3.6	3.67	3.73	3.8
Duty Cycle	0.2	200	815.5556	1865	3377.78
	0.5	500	2038.889	4662.5	8444.44
	0.75	750	3058.333	6993.75	12666.7

Table 84: MDB Theoretical Single MOSFET Power Table

		Resulting Percent Differnce			
	Load Current [A]	50	100	150	200
	Rds [mOhm]	3.6	3.67	3.73	3.8
Duty Cycle	0.2	-25.54%	-14.77%	-10.37%	-7.64%
	0.5	-11.87%	-5.86%	-3.57%	-2.03%
	0.75	-8.13%	-3.63%	-1.91%	-0.68%

Table 85: MDB Simulation VS Theoretical Difference Table

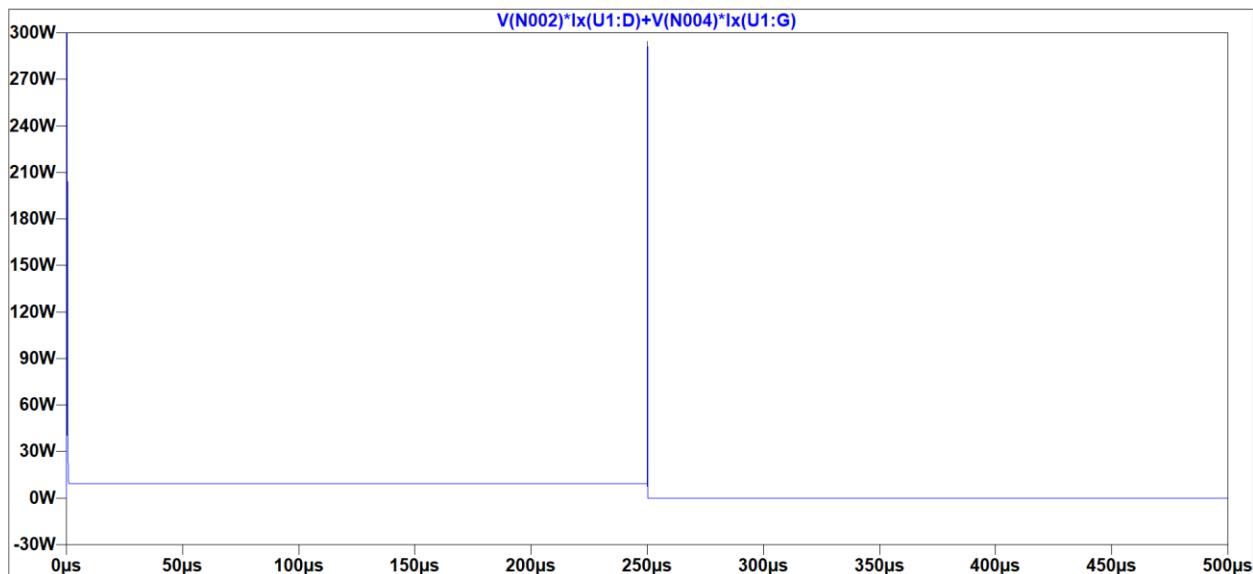


Figure 78: MDB Single FET Switching Power Waveform

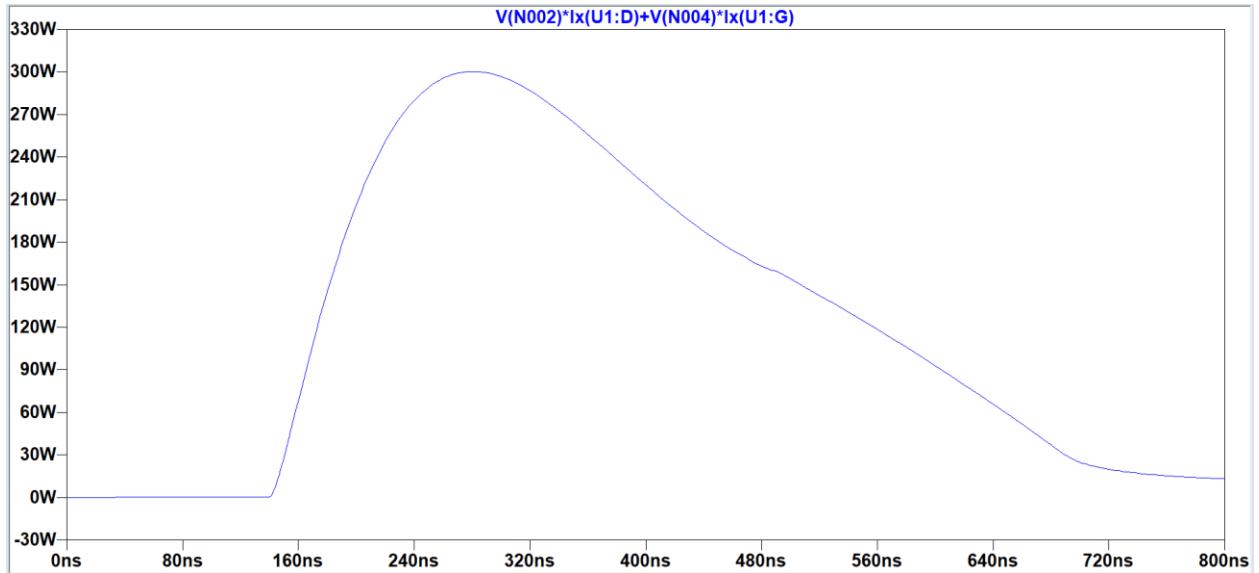


Figure 79: MDB FET Power Turn On Spike Waveform

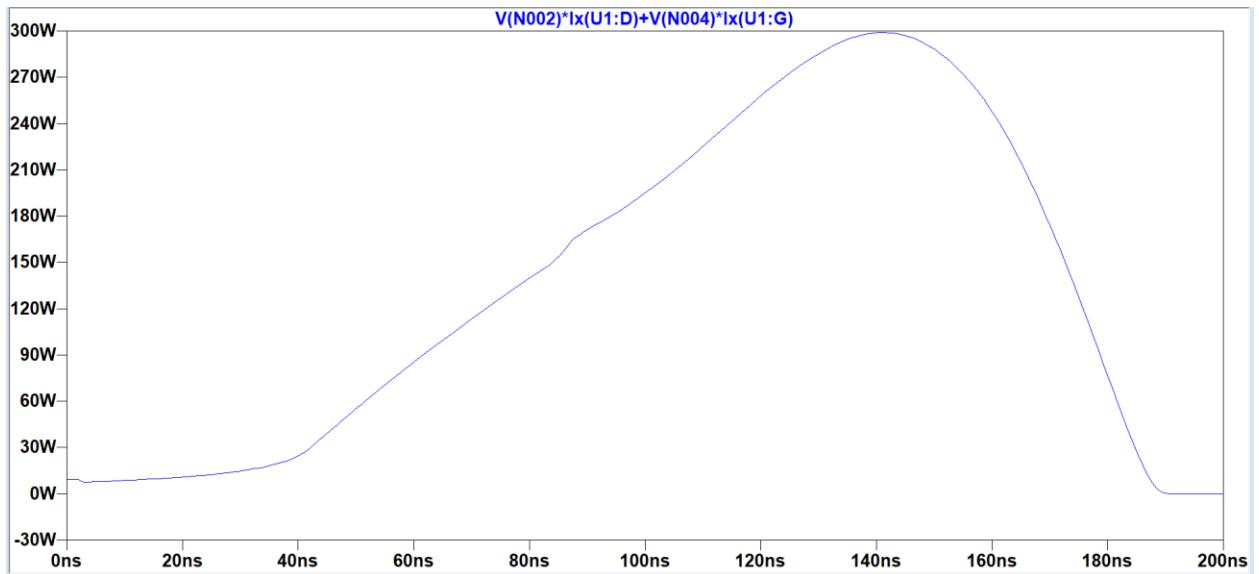


Figure 80: MDB FET Power Turn Off Spike Waveform

The thermal junction coefficient $R_{\theta JA}$ of the IRFS3006-7PPbF MOSFET being used is 40 $^{\circ}\text{C}/\text{W}$. This means that for every watt a MOSFET consumes the junction temperature will rise 40 degrees Celsius. Observing the values of Table 84 the power of a MOSFET can exceed 12 watts which will result in a junction temperature of 480 $^{\circ}\text{C}$ above ambient. The max operating junction temperature of the IRFS3006-7PPbF is 175 $^{\circ}\text{C}$ therefor a thermal management solution must be

implemented. This will be done with the addition of thermal mass on the bottom of the board. The increase in thermal mass and surface area will result in a longer time needed for the MOSFET to rise an equivalent change in temperature as well as a lower $^{\circ}\text{C}/\text{W}$. To ensure that the temperature of a single MOSFET does not exceed the maximum allowable junction temperature of 175 $^{\circ}\text{C}$ while being power limited to 12 watts dissipated in a single MOSFET the thermal coefficient of the MOSFET must be decreased to 14 $^{\circ}\text{C}/\text{W}$. This will be accomplished with the additional thermal mass and a heatsink applied to the case of each MOSFET. In addition to an increase in thermal mass and heatsinks the temperature of the board will be monitored near the MOSFETs. If the temperature begins to exceed a threshold the duty ratio will decrease. As seen before by decreasing the duty ratio the power dissipated by a single MOSFET will decrease. This will allow the power dissipated by a single MOSFET to be controlled. By limiting the power of a MOSFET by adjusting the duty ratio in turn the temperature of the MOSFET can be limited. Limiting the temperature of the MOSFET will prevent damage to the MOSFET allowing for safe operation. (MJH)

3.7.3 Weapon Motor Control Board

The gate resistance for the IGBT module (IXGN200N60B3-ND) that will switch the weapon motor power was selected based on the following calculations and simulations. The IGBT gate connection controls the conduction of the collector to emitter junctions. When the gate to emitter voltage reaches a threshold, current will begin to flow from the collector to the emitter. According to the datasheet, this threshold has a range of values at ambient temperature.

The range of threshold voltage is from 3 to 8.1 volts. (ZDK)

The gate terminal of the IGBT is modeled as a capacitance, whereas once charged, the terminal does not require current to maintain conduction of the collector and emitter junctions. The rate at which the gate terminal is charged depends on the total charge required to bring the gate voltage to the threshold. According to the datasheet, the amount of charge on the gate at 15V is given to be 750 nC. By using the equation 1 shown below, the gate capacitance is found to be 500 nF. (ZDK)

$$C = \frac{q}{V} = \frac{750\text{nC}}{15\text{V}} \approx 500\text{nF} \quad (1)$$

where C is capacitance, q is charge and V is voltage

Symbol ($T_J = 25^\circ\text{C}$, Unless Otherwise Specified)	Test Conditions	Characteristic Values		
		Min.	Typ.	Max.
$Q_{g(on)}$		750		nC
Q_{ge}		115		nC
Q_{gc}		245		nC

$I_C = 100\text{V}, V_{GE} = 15\text{V}, V_{CE} = 0.5 \cdot V_{CES}$

Table 86: IXGN200N60B3-ND Switching Characteristics

Simulation of the stated capacitance subjected to a step voltage of 10V (originally chosen as the gate drive voltage) resulted in the gate terminal reaching the maximum threshold voltage of 8.1 volts in approximately 2 uS. This time is far less than the selected 25 kHz switching period of 40uS. The gate resistance used in this simulation was 2 ohms, due to the assumption that when the capacitor is discharged, the voltage over the resistor will be 10V resulting in 5 amps of current (per ohms law). The 5 amps was determined to be a maximum according to the voltage regulator used to generate the 10V being rated at 5amps. (ZDK)

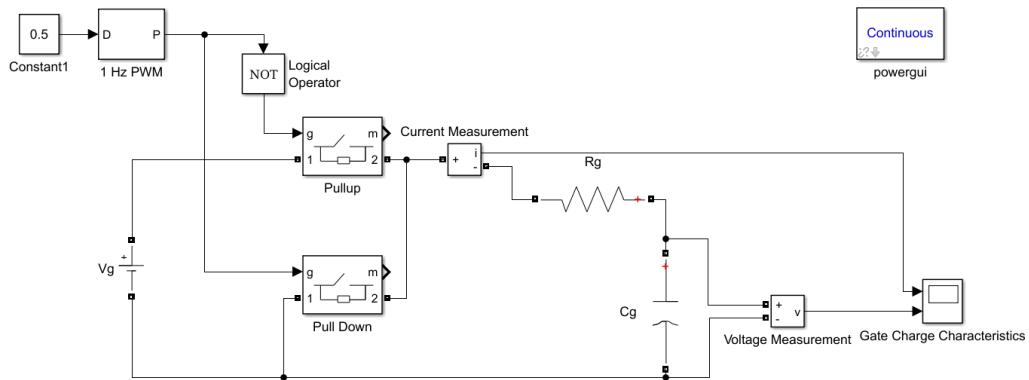


Figure 83: IGBT Gate Capacitance Charging Simulation Schematic

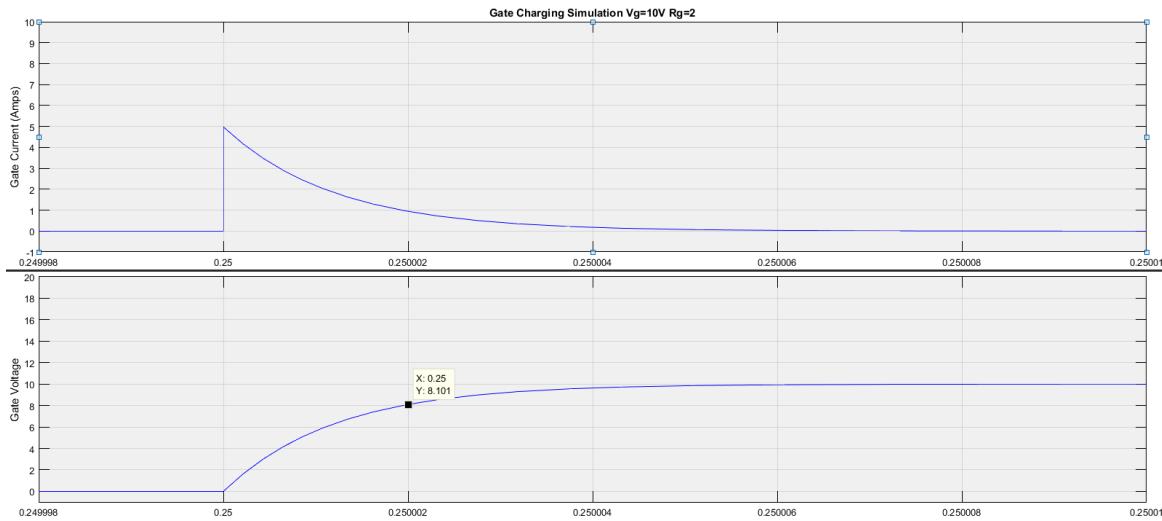


Figure 82: IGBT Gate Charging Simulation Results Vg=10V Rg=2ohms

Investigation of higher gate currents promoted simulating the presence of output capacitance on the voltage regulator providing the ability to source more than 5 amps of current to the gate. The driver chosen to source the gate current is rated at 9 amps, so the simulation was ran again at 9 amps of gate charge. At a resistance of 1.1 ohm to the gate, the time to reach the 8.1 volt threshold was approximately 1.1 uS shown in Figure 84, nearly half the time of the initial simulation. (ZDK)

Upon further review, an application note regarding the selection of R_g was provided by the manufacturer. The manufacturer provided suggestions to increase the gate voltage to ensure lower loss switching to ensure low V_{CE} . during conduction. A gate voltage of 18V was recommended when utilizing high gate currents to achieve a quick charge time. The simulation performed shown in Figure 85 displays a rise time to 8.1 volts in 725ns. In addition to the faster rise time, the higher gate voltage will allow for less conduction losses, as the higher voltage will provide a lower V_{CE} . (ZDK)

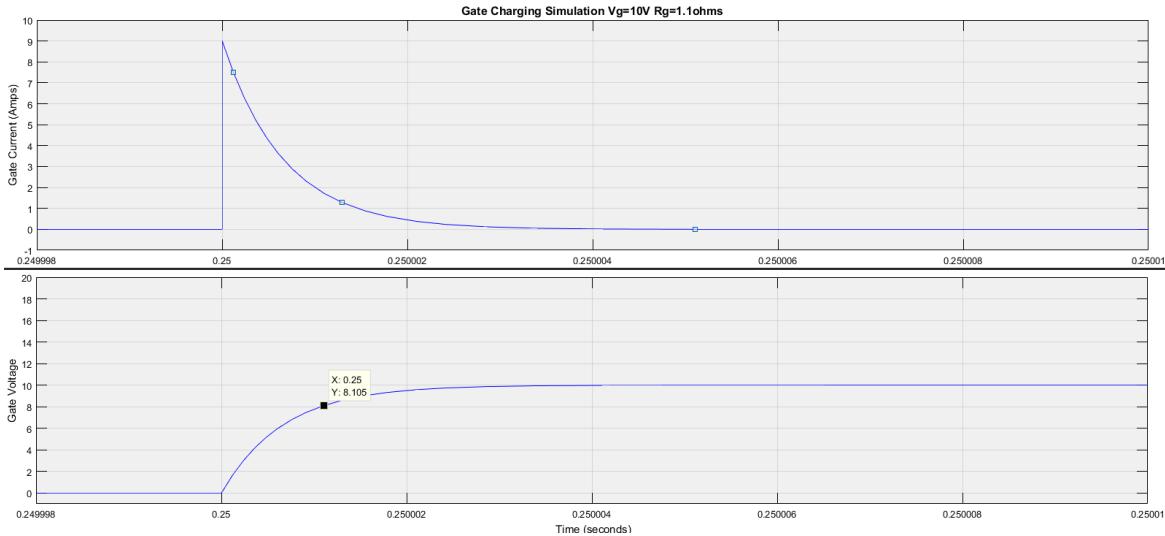


Figure 84: IGBT Gate Charging Simulation Results $V_g=10V$ $R_g=1.1\text{ohms}$

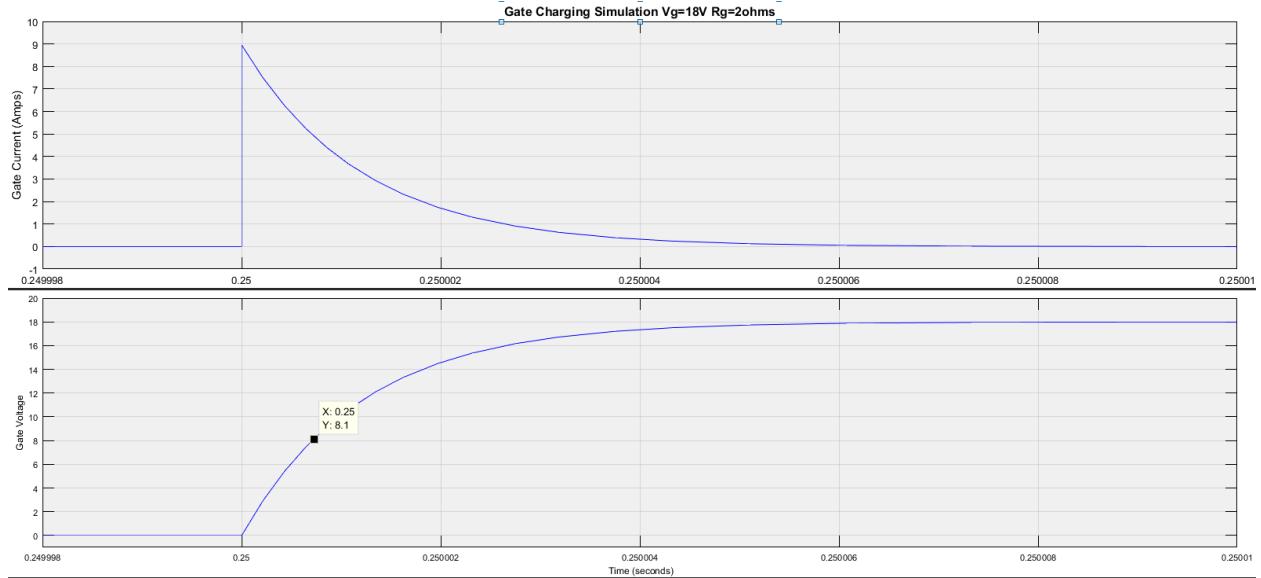


Figure 85: IGBT Gate Charging Simulation Results $V_g=18V$ $R_g=2\text{ohms}$

Further simulations were performed to ensure that the voltage regulator (APXW005A0X) would provide adequate current given a chosen output capacitance. The regulator is capable of providing the programmed voltage for currents less than 2.5 amps, but limits the output voltage at current exceeds this threshold in an attempt to provide short circuit and overcurrent protection. During these transient responses of gate current, the output capacitor on the regulator will provide current to satisfy the needs of the IGBT gate. (ZDK)

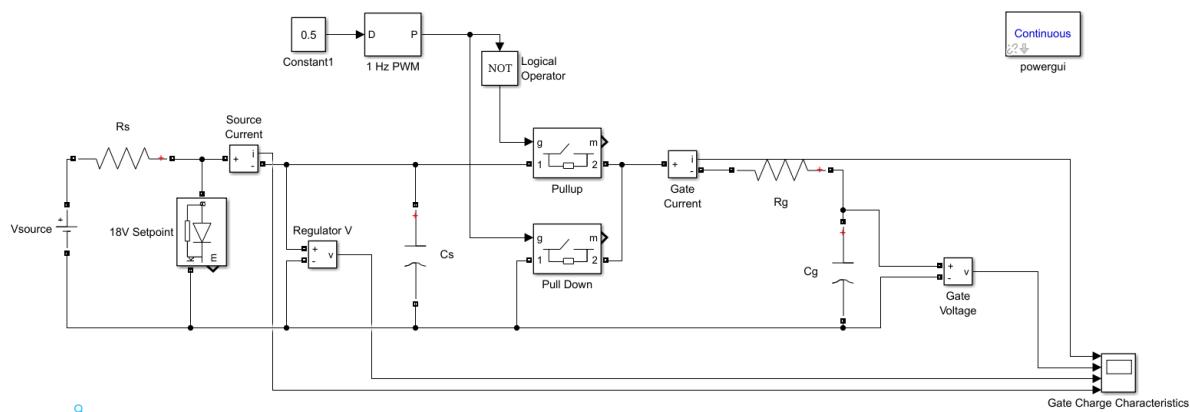


Figure 86: IGBT Voltage Regulator Output Capacitance Simulation Schematic

The regulator is modeled shown in Figure 86 as a 33V source with a source resistance of 6 ohms. These characteristics are determined from the data sheets voltage regulation vs. output current shown in Figure 88: APXW005A0X Output Current vs. Output Voltage. (ZDK)

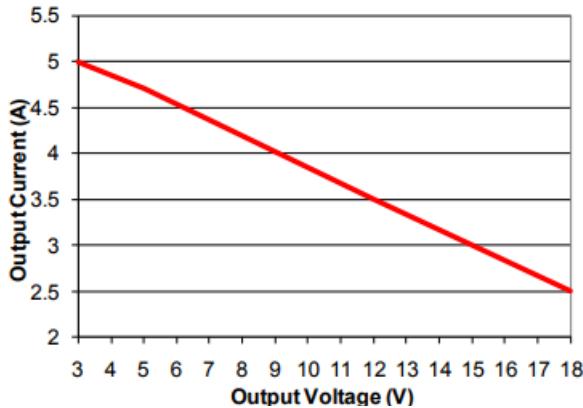


Figure 87: APXW005A0X Output Voltage vs Output Current

After swapping the axis, the source resistance that will allow the simulation to reproduce this relationship is 6 ohms, shown as the slope of the line in Figure 88. (ZDK)

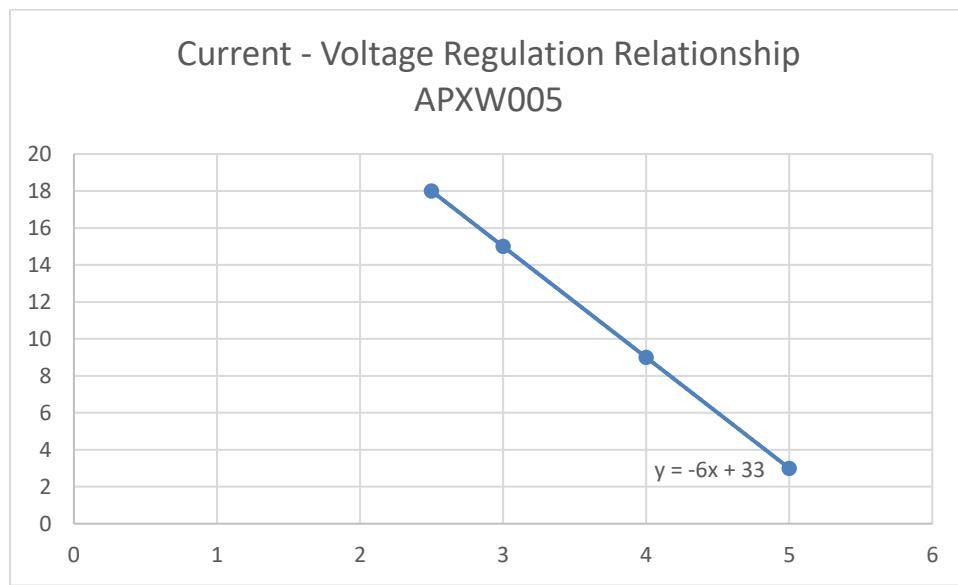


Figure 88: APXW005A0X Output Current vs. Output Voltage

By placing a diode after the 6ohm source resistance, in parallel with the source bulk capacitance, the voltage can be maintained at 18V when the load is not active. When the gate attempts to draw current, the supply will maintain 18V at the output capacitor until the regulator must push more than 2.5amps, at which point the voltage will drop according to the given figure from the regulators data sheet. (ZDK)

By varying the regulators output capacitor, the transient effect of the regulator voltage and the effects on the gate charging rate can be examined. Capacitance values of 10uF, 22uF, 100uF, 1000uF and 2000uF were simulated and results shown below. (ZDK)

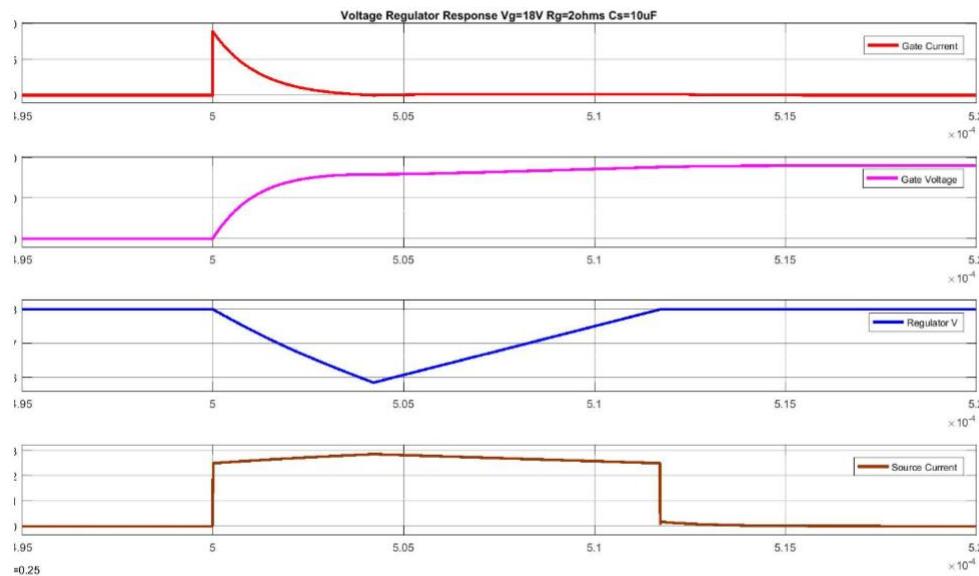


Figure 89: IGBT Voltage Regulator Response Cs=10uF

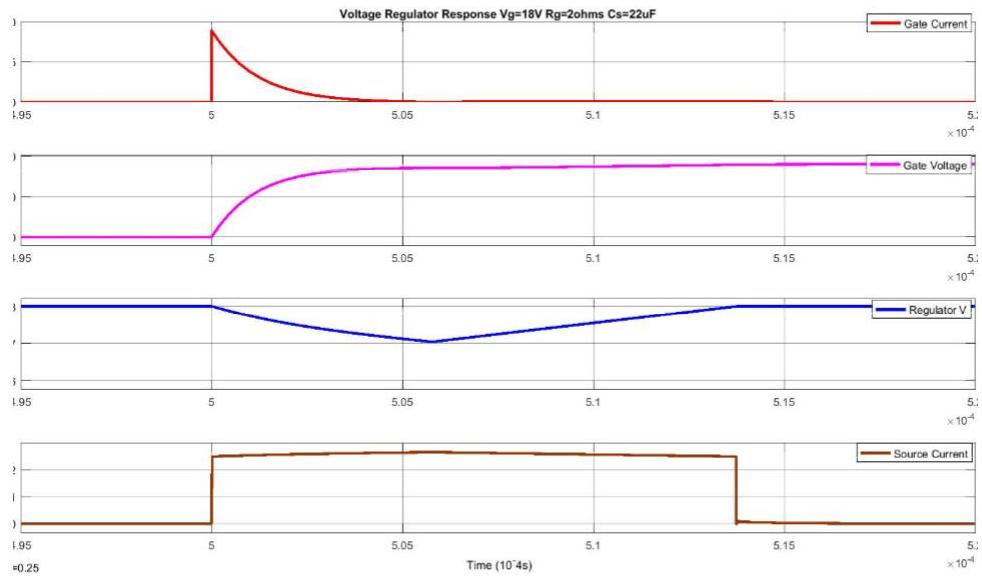


Figure 90: IGBT Voltage Regulator Response Cs=20uF

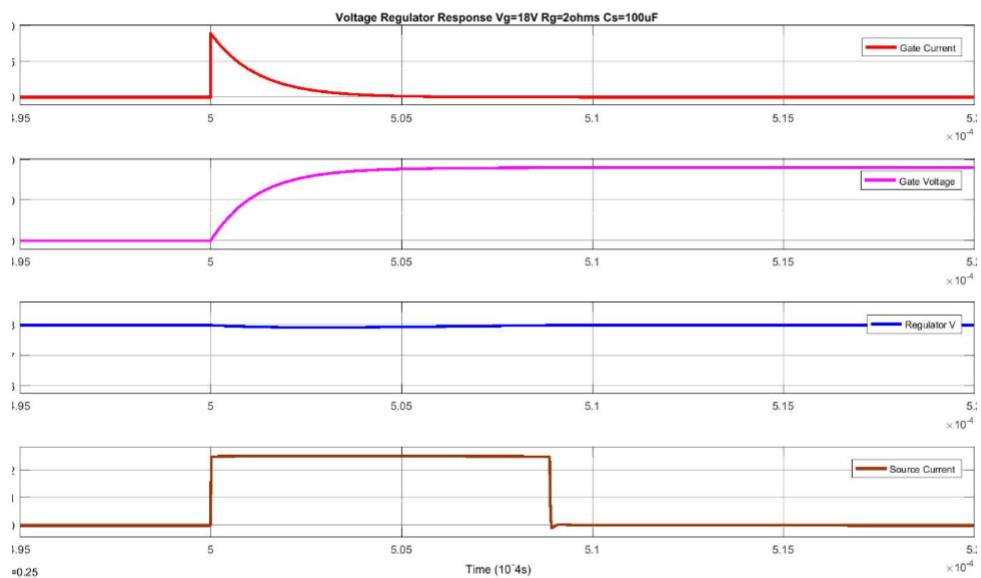


Figure 91: IGBT Voltage Regulator Response Cs=100uF

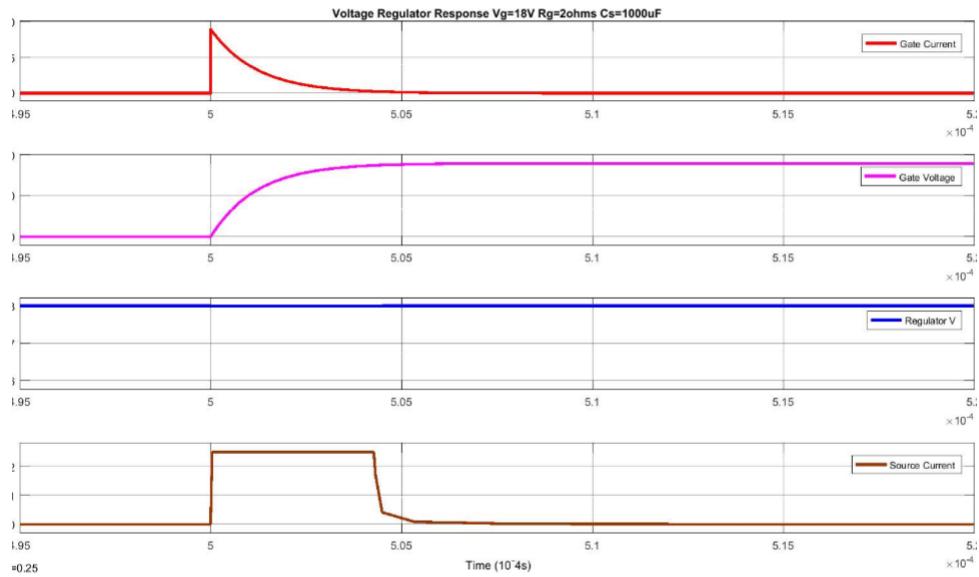


Figure 93: IGBT Voltage Regulator Response Cs=1000uF

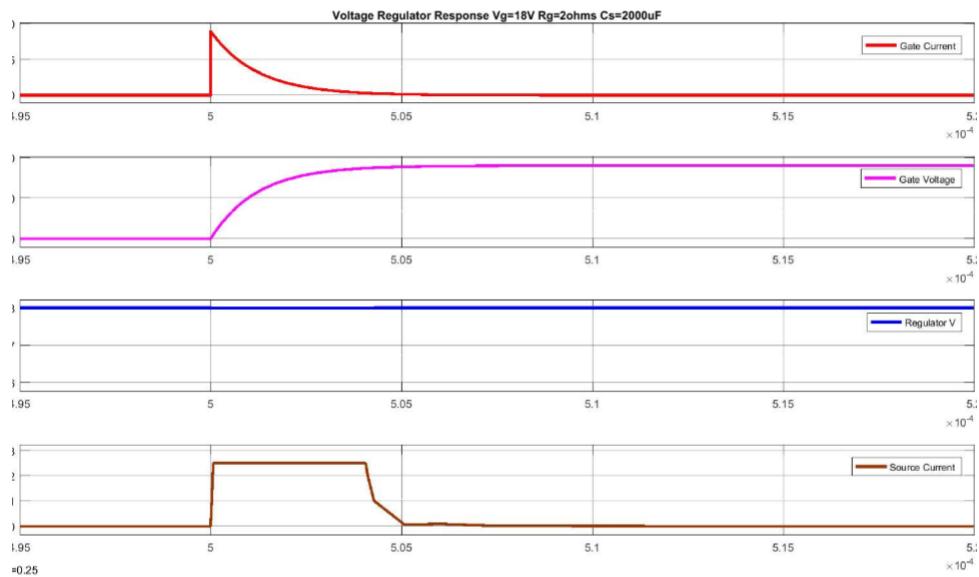


Figure 92: IGBT Voltage Regulator Response Cs=2000uF

The results of the simulations show that as the capacitance is increased, the amount of voltage drop due to current draw on the regulator decreases. At relatively low capacitances the voltage drops as much as 2 volts, which is not favorable as it induces abnormal voltage transients into the gate voltage due to deprived current capabilities. There also appears to be a local minimum of threshold rate achieved near 100uF. It is generally a bad idea to simply add the maximum amount of capacitance that a regulator is capable of. A value for Cs of 100uF was chosen as a result of the simulations. Numerical analysis of the results from the simulations are shown in the table below. (ZDK)

Capacitance	Gate Threshold Period (8.1V attained on gate)	Minimum Regulator Voltage During Transient
10uF	735nS	15.9V
22uF	726nS	17.04V
100uF	723nS	17.93
1000uF	724nS	17.996
2000uF	726nS	17.998

Table 87: IGBT Voltage Regulator Output Capacitance Simulation Results

The Weapon Motor Board will send a PWM signal that will control the Motor through an IGBT Driver (IXGN200N60B3) in the robot that spins the Weapon. To control the charge up speed of the spin, the PWM waveform will increase or decrease by the Duty Cycle controlled by the microcontroller. To calculate the Duty Cycle, the following equation will be used:

$$Period = \frac{1}{Frequency} \quad (2)$$

$$Duty\ Cycle = \frac{Pulse\ Width}{Period} \quad (3)$$

The highest frequency capability of the IGBT Driver is 40kHz thus, by equation (1) the period will be 25us. Since the Duty Cycle is controllable from the software, the equation (2) can be algebraically modified to:

$$\text{Duty Cycle} = \frac{\text{Pulse Width}}{\text{Period}} \leftrightarrow \text{Pulse Width} = \text{Duty Cycle} * \text{Period} \quad (3)$$

The Pulse Width represents the duration of the period when the weapon motor is enabled to be spinning. For example if the Duty Cycle is set as 50% the pulse width will be 12.5us. By this the user is allowed full control of the activation/deactivation of the weapon.

3.7.4 Proximity Sensor Board

Design calculations were done for the angular velocity in order to determine the arc length of the weapon shell window. The first parameter taken into account was the sample rate of the LIDAR sensor which was determined to be 500 Hz (2ms period). The weapon shell radius was measured to be 1.375 ft. The angular velocity of the weapon motor can reach a maximum of 1200 rpm. First, the angular velocity in radians per second for the weapon motor was determined to be:

$$\omega_M = 1200 \left(\frac{\text{rev}}{\text{min}} \right) * \left(\frac{2\pi \text{ rad}}{\text{rev}} \right) * \left(\frac{\text{min}}{60 \text{ sec}} \right) = 44.44\pi \frac{\text{rad}}{\text{s}} \quad (1)$$

Taking into consideration the period of the LIDAR sensor, which was determined to be 2ms, the degrees swept in one period was determined to be:

$$\theta_s = 44.44\pi \frac{\text{rad}}{\text{s}} * (2 * 10^{-3} \text{ s}) \left(\frac{180^\circ}{\pi} \right) = 16^\circ \quad (2)$$

Last, the arc length equation was used to determine the minimum arc length of the window to ensure proper LIDAR sensor reads:

$$L = 2\pi R \left(\frac{\theta_s}{360^\circ} \right) \quad (3)$$

Using the radius of the weapon shell ($R = 1.375 \text{ ft.}$) the arc length (L) was calculated to be:

$$L = 2\pi(1.375 \text{ ft.}) \left(\frac{16^\circ}{360^\circ} \right) = 4.6077 \text{ in.} \quad (4)$$

Design calculations were done for power dissipation in all voltage regulators to determine if heat sinks were necessary. The following equation will be used to determine power dissipation:

$$P_D = (V_{IN} - V_{OUT}) \sum I \quad (1)$$

Each linear regulator supplies 5 volts to all sensors and microcontrollers in each satellite location. One linear regulator is assigned for each of the 7 satellite locations which contain one hall effect sensor, one LIDAR sensor, and one microcontroller all supplied by the 5 volts from the linear regulator. The maximum current consumed by the LIDAR sensor, I_L , is 135 mA. The maximum current consumed by the hall effect sensor, I_H , is 5.2 mA. The maximum supply current drawn, I_{RS485} , for the RS485 transceiver (U15 in PSB schematic) is 500 μ A according to the data sheet. The total current, I_{DIP} , consumed by the DIP switch is 2 mA. Last, the maximum current consumed by the microcontroller, I_μ , is 160 mA. Therefore, the following equation determines the power dissipated in each linear regulator:

$$P_D = (9V - 5V) * (135mA + 5.2mA + 500\mu A + 2mA + 160mA) = 1.2108 W \quad (2)$$

The thermal junction to ambient resistance, $\theta_{JA} = 62.5^\circ C/W$, the temperature of the linear voltage regulator was determined by the following equations:

$$T_{^{\circ}C} = P_D * \theta_{JA} \quad (3)$$

$$T_{^{\circ}C} = * 1.2108W * 62.5 \frac{^{\circ}C}{W} = 75.675 ^{\circ}C \quad (4)$$

Therefore, no heat sink will be required since the operating temperature range is $0^\circ C - 125^\circ C$.

Design calculations were done for the LIDAR measurement read time and also the computing instruction time to read and store data. According to the LIDAR Lite v3 specifications, the measurement time, τ_{LIDAR} , takes approximately 2 ms. (MPH)

I2C communication will be used to communicate with the LIDAR sensor and 7 bit addressing will be used. When data is written to the LIDAR sensor or read from the sensor, the address must first be transmitted with 7 bits used for the address, 1 bit used for read or write, and

the last bit used for acknowledge from the slave device. After the sensor address is transmitted, each subsequent packet will be a size of 1 byte each plus 1 bit for acknowledgement (9 bits).

(MPH)

The sequence of events for enabling the LIDAR to take a measurement and then get the data back from the sensor are as follows:

1. I2C communication starts by pulling SDA low while SCL is high.
2. The sensor is first written to with the 7 bit address, 1 bit for write, and 1 bit for acknowledgment (9 bits total).
3. Register address to be written to is transmitted to the sensor (9 bits).
4. Data is written to the register specified (9 bits).
5. I2C communication is stopped by pulling SDA high while SCL is high.
6. I2C communication is initialized.
7. The sensor address is transmitted again (9 bits).
8. Register address to be read from is transmitted to the sensor (9 bits).
9. The program halts until the busy flag is cleared (clears once LIDAR measurement is complete).
10. I2C communication stops.
11. I2C communication is initialized.
12. The address is transmitted again (9 bits).
13. The measurement data is read (18 bits).
14. I2C communication stops.

According to the sequence of events just mentioned, a total of 8 bytes will be transmitted or received. The speed of I2C transmission, τ_{I2C} , will be $\frac{400 \text{ kbits}}{\text{sec}}$. Therefore, the rate of 1 bit being transferred, τ_{1bit} , will be $2.5 \mu\text{s}$. Since there are 72 bits to be transmitted or received the rate of data, τ_{Data} , will be $180 \mu\text{s}$. There will be a delay of $30 \mu\text{s}$ between each start and stop condition, therefore, τ_{Delay} , will be $30 \mu\text{s}$. The total number of delays will be the total number of instances where I2C transmission stops then starts again which occur 2 times. (MPH)

After the LIDAR takes a measurement and the data is transmitted to the PSB through I2C communication, the microcontroller will store the 2 bytes of measurement data in an array. The instruction clock frequency, F_{CY} , for the PIC16F18345, will be 8 MHz. Therefore, the instruction clock period, τ_{CY} , will be 125 ns. (MPH)

Figure 94 below shows a disassembled C program that stores 2 bytes of data in an integer array which will be done for the LIDAR measurement data. (MPH)

```

14      !     for(i = 0; i < 2; i++) {
15 0x2DA: CLR i
16 0x2DC: BRA 0x2F0
17 0x2EA: MOV i, W0
18 0x2EC: INC W0, W0
19 0x2EE: MOV W0, i
20 0x2F0: MOV i, W0
21 0x2F2: SUB W0, #0x1, [W15]
22 0x2F4: BRA LE, 0x2DE
23 !         distance[i] = LIDAR_data;
24 0x2DE: MOV i, W0
25 0x2E0: MOV LIDAR_data, W1
26 0x2E2: ADD W0, W0, W2
27 0x2E4: MOV #0x800, W0
28 0x2E6: ADD W2, W0, W0
29 0x2E8: MOV W1, [W0]
30 !

```

Figure 94: Disassembled LIDAR Storage Code

According to Figure 94, it takes a total of 14 instructions to store the LIDAR measurement data into the integer array. (MPH)

The following equation determines the total time to take a measurement, transmit the data, and then store the data in the microcontroller:

$$\tau_{total} = \tau_{LIDAR} + \tau_{Data} + (2 * \tau_{Delay}) + (14 * \tau_{CY}) \quad (1)$$

Using equation 1, the total time, τ_{total} , is determined to be 2.24175 ms. (MPH)

4.0.0 Parts List

A comprehensive parts list of the components needed to construct all the systems in the project is as follows. The tables below include the parts lists to construct a single main control board, two motor driver boards, 7 proximity sensor boards, and a single weapon motor controller board. (MJH)

Main Control Board:

Qty.	Refdes	Part Num.	Description	Vendor Part Num.	
4	R1, R2, R7, R8	RC0805JR-070RL	0 Ohms Jumper 0.125W, 1/8W Chip Resistor 0805 (2012 Metric) Moisture Resistant Thick Film	311-0.0ARCT-ND	https://
1	C19	C0805C104K5RACTU	0.1µF ±10% 50V Ceramic Capacitor X7R 0805 (2012 Metric)	399-1170-1-ND	https://
2	R13, R34	RC0805FR-071K65L	1.65 kOhms ±1% 0.125W, 1/8W Chip Resistor 0805 (2012 Metric) Moisture Resistant Thick Film	311-1.65KCRCT-ND	https://
2	C11, C12	C0805C104K5RACTU	0.1µF ±10% 50V Ceramic Capacitor X7R 0805 (2012 Metric)	399-1170-1-ND	https://
2	C9, C31	C0805C101J5GACTU	100pF ±5% 50V Ceramic Capacitor C0G, NP0 0805 (2012 Metric)	399-1122-1-ND	https://
1	X2	1040310811	10 (8 + 2) Position Card Connector Secure Digital - microSD™ Surface Mount, Right Angle Gold	WM6357CT-ND	https://
24	R16, R21, R22, R23, R24, R25, R26, R27, R28, R29, R30, R31, R32, R35, R40, R41, R53, R54, R55, R56, R3, R5, R6, R9	RC0805FR-0710KL	10 kOhms ±1% 0.125W, 1/8W Chip Resistor 0805 (2012 Metric) Moisture Resistant Thick Film	311-10.0KCRCT-ND	https://
2	C22, C23	CL21C130JBANNNC	13pF ±5% 50V Ceramic Capacitor C0G, NP0 0805 (2012 Metric)	1276-2582-1-ND	https://
2	R37, R38	RMCF0805JT120R	120 Ohms ±5% 0.125W, 1/8W Chip Resistor 0805 (2012 Metric)	RMCF0805JT120RCT-ND	https:// ND/19

			Metric) Automotive AEC-Q200 Thick Film		
1	X18	2401548E4#2A	USB - C USB 3.1 (USB 3.1 Gen 2, Superspeed+) Receptacle Connector 24 Position Surface Mount, Right Angle; Through Hole	12401548E4#2ACT-ND	https://
1	R17	RC0805JR-071KL	1 kOhms ±5% 0.125W, 1/8W Chip Resistor 0805 (2012 Metric) Moisture Resistant Thick Film	311-1.0KARCT-ND	https://
25	C1, C2, C3, C4, C5, C7, C8, C10, C13, C14, C15, C16, C17, C18, C24, C25, C26, C27, C28, C29, C30, C32, C33, C34, C35	UMK212BJ105KG-T	1µF ±10% 50V Ceramic Capacitor X5R 0805 (2012 Metric)	587-2229-1-ND	https://
1	S2	218-6LPST	Dip Switch SPST 6 Position Surface Mount Slide (Standard) Actuator 25mA 24VDC	CT2186LPST-ND	https://
3	R4, R10, R11	RMCF0805JT2K00	2 kOhms ±5% 0.125W, 1/8W Chip Resistor 0805 (2012 Metric) Automotive AEC-Q200 Thick Film	RMCF0805JT2K00CT-ND	https://ND/19
2	R12, R15	RC0805FR-073K65L	3.65 kOhms ±1% 0.125W, 1/8W Chip Resistor 0805 (2012 Metric) Moisture Resistant Thick Film	311-3.65KCRCT-ND	https://
12	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12	3557-2	Fuse Block 30A 500V 1 Circuit Blade PCB	36-3557-2-ND	https://
4	R19, R33,R18, R20	RC0805FR-074K7L	4.7 kOhms ±1% 0.125W, 1/8W Chip Resistor 0805 (2012 Metric) Moisture Resistant Thick Film	311-4.70KCRCT-ND	https://
4	X1, X15, X16, X17	WM14471CT-ND	8 Position Receptacle Connector 0.059" (1.50mm) Surface Mount, Right Angle Tin	5025850870	https://
13	X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14,X9	5031481090	10 Position Receptacle Connector 0.059" (1.50mm) Surface Mount, Right Angle Tin	5031481090	https://
8	R57, R58, R59, R60, R61, R62, R63, R64	RC0805FR-071KL	1 kOhms ±1% 0.125W, 1/8W Chip Resistor 0805 (2012 Metric) Moisture Resistant Thick Film	311-1.00KCRCT-ND	https://

1	L1	7447715330	33 μ H Shielded Wirewound Inductor 2.3A 85 mOhm Max Nonstandard	732-2196-1-ND	https://
1	U10	74LVC4245APW,118	Voltage Level Translator Bidirectional 1 Circuit 8 Channel 24-TSSOP	1727-4308-1-ND	https://
1	R14	RC0805FR-07768RL	768 Ohms \pm 1% 0.125W, 1/8W Chip Resistor 0805 (2012 Metric) Moisture Resistant Thick FilmResistant Thin Film	311-768CRCT-ND	https://
2	C20, C21	CL21CR82BBANNNC	0.82pF \pm 0.1pF 50V Ceramic Capacitor COG, NP0 0805 (2012 Metric)	1276-2703-1-ND	https://
1	U12	ACSA03-41EWA-F01	Character LED Display Module Red 7-Segment 1 Character Common Anode 2V 20mA 0.394" H x 0.287" W x 0.148" D (10.00mm x 7.30mm x 3.75mm) 10-SMD, No Lead	754-1023-1-ND	https://
5	D2, D3, D4, D5, D6	APTD2012LSURCK	Red 630nm LED Indication - Discrete 1.75V 0805 (2012 Metric)	754-2044-1-ND	https://
1	D1	SD0805S040S0R5	Diode Schottky 40V 500mA (DC) Surface Mount 0805 (2012 Metric)	478-7802-1-ND	https://
1	U17	CD4050BD	CMOS Hex Buffer/Converters	296-32894-5-ND	https://
1	C6	EEE-1VA101XP	100 μ F 35V Aluminum Electrolytic Capacitors Radial, Can - SMD 2000 Hrs @ 85°C	PCE3951CT-ND	https://
1	Y1	XC2122CT-ND	32.768kHz \pm 20ppm Crystal 9pF 60 kOhms 0805 (2012 Metric)	ECS-.327-CDX-1074	https://
1	U2	LD1086DT33TR	Linear Voltage Regulator IC Positive Fixed 1 Output 3.3V 1.5A D-Pak	497-3446-1-ND	https://
1	IC1	LD39100PU33RY	LDO Voltage Regulators 1 A low quiescent current low noise voltage regulator	497-16700-1-ND	https://
1	U5	LSM6DS3HTR	IMU ACCEL/GYRO/TEMP I2C/SPI LGA	497-16367-1-ND	https://
1	U15	MAX490ESA+	RS485 TRANSEIVER	MAX490ESA+-ND	https://

13	Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12, Q13	MMBT3904LT3G	TRANS NPN 40V 0.2A SOT23	MMBT3904LT3GOSCT-ND	https://
1	Y2	NX2520SA-24.000000MHZ	24MHz 710ppm Crystal 10pF 60 Ohms 4-SMD, No Lead	644-1061-1-ND	https://
2	U4, U9	PCA9534APWR	I/O Expander 8 I ² C, SMBus 400kHz 16-TSSOP	296-21760-1-ND	https://
1	U6	PIC32MZ2048EFG064-I/PT	IC MCU 32BIT 2MB FLASH 64TQFP	PIC32MZ2048EFG064-I/PT	https://
1	S1	PTS645SM43SMTR92 LFS	Tactile Switch SPST-NO Top Actuated Surface Mount	CKN9112CT-ND	https://ND/11
1	U3	R-629.0P	Non-Isolated PoL Module DC DC Converter 1 Output 9V 2A 11V - 32V Input	945-1604-5-ND	https://ND/23
1	U1	ROF-78E5.0-0.5SMD-R	Non-Isolated PoL Module DC DC Converter 1 Output 5V 500mA 9V - 36V Input	945-1690-1-ND	https://ND/35
1	D16	SS23	Diode Schottky 30V 2A Surface Mount DO-214AA (SMB)	SS23CT-ND	https://
2	U7, U8	SN74LVC1T45DCKR	Voltage Level Translator Bidirectional 1 Circuit 1 Channel 420Mbps SC-70-6	296-16844-1-ND	https://
1		MS-C6-8G	Memory Card microSD™ 8GB Class 6	768-1288-ND	

Motor Driver Board:

Qty.		Description	Unit	Total
Qty.	Part Num.	Description	Cost	Cost
2	CC2220KKX7R9BB105	CAP CER 1UF 50V X7R 2220	\$0.76	\$1.52
8	ECA-2AHG102	CAP ALUM 1000UF 20% 100V RADIAL	\$1.36	\$10.88
2	CC0805KRX7R9BB103	CAP CER 10000PF 50V X7R 0805	\$0.02	\$0.04
4	GRM319R61E106KA12D	CAP CER 10UF 25V X5R 1206	\$0.17	\$0.69
4	CC1210MKX5R7BB107	CAP CER 100UF 16V X5R 1210	\$1.82	\$7.28
2	C0805C474K5RACTU	CAP CER 0.47UF 50V X7R 0805	\$0.05	\$0.10
10	08055C104KAT2A	CAP CER 0.1UF 50V X7R 0805	\$0.01	\$0.09
50	CL21B105KAFNNNE	CAP CER 1UF 25V X7R 0805	\$0.04	\$1.81
6	C0805C101J5GACTU	CAP CER 100PF 50V C0G/NP0 0805	\$0.05	\$0.30
4	CL21C180JBANNNC	CAP CER 18PF 50V C0G/NP0 0805	\$0.05	\$0.19
100	502579-1000	1.5 WB CONN. PLUG TERMINAL 24-28	\$0.11	\$10.88
22	APT2012LZGCK	LED GREEN CLEAR 0805 SMD	\$0.31	\$6.91
2	SL23-E3/52T	DIODE SCHOTTKY 30V 2A DO214AA	\$0.48	\$0.96
8	VB30100S-E3/8W	DIODE SCHOTTKY 100V 30A TO263AB	\$1.05	\$8.42
2	CD1206-S01575	DIODE GEN PURP 100V 150MA 1206	\$0.03	\$0.06
4	APT2012LSYCK/J3-PRV	LED YELLOW CLEAR 0805 SMD	\$0.49	\$1.96
6	APTD2012LSURCK	LED RED CLEAR SMD	\$0.21	\$1.23
2	1455N1601BK	BOX ALUM BLACK 6.3"L X 4.06"W	\$25.14	\$50.28

2	A 6Z25M028DF0906	5 mm (HTD) Pitch,28 Teeth, 6mm Bore, Aluminum Insert, Polycarbonate Timing Pulley for 9mm Wide Belt	\$8.26	\$16.52
2	I010	3.500" WIDE EXTRUDED ALUMINUM HEATSINK 10" Length	\$11.50	\$23.00
2	LP2985-33DBVR	IC REG LINEAR 3.3V 150MA SOT23-5	\$0.60	\$1.20
2	MH2029-300Y	SMD EMI Suppression Ferrite Beads	\$0.06	\$0.12
2	A 6Z25-040DF0916	5 mm (HTD) Pitch,40 Teeth, 0.5" Bore, Aluminum Insert, Polycarbonate Timing Pulley for .354(9mm)" Wide Belt	\$11.14	\$22.28
2	Motor Controller V4	PCB of the Motor Controller Board 5x	\$43.84	\$87.68
2	SI2338DS-T1-GE3	MOSFET N-CH 30V 6A SOT23	\$0.35	\$0.70
24	IRFS3006-7PPBF	MOSFET N-CH 60V 240A D2PAK-7	\$3.01	\$72.24
4	RC0805JR-076K2L	RES SMD 6.2K OHM 5% 1/8W 0805	\$0.01	\$0.05
10	RC0805FR-074K7L	RES SMD 4.7K OHM 1% 1/8W 0805	\$0.02	\$0.16
8	RC0805JR-070RL	RES SMD 0 OHM JUMPER 1/8W 0805	\$0.01	\$0.10
6	RC0805JR-070RL	RES SMD 0 OHM JUMPER 1/8W 0805	\$0.01	\$0.08
4	RMCF0805JT22R0	RES SMD 22 OHM 5% 1/8W 0805	\$0.02	\$0.07
2	RMCF0805JT3K00	RES SMD 3K OHM 5% 1/8W 0805	\$0.02	\$0.03
4	RMCF0805JT2K00	RES SMD 2K OHM 5% 1/8W 0805	\$0.02	\$0.07
4	RC0805JR-071K6L	RES SMD 1.6K OHM 5% 1/8W 0805	\$0.01	\$0.05
2	RC0805FR-07620KL	RES SMD 620K OHM 1% 1/8W 0805	\$0.02	\$0.03

2	RC0805FR-07976KL	RES SMD 976K OHM 1% 1/8W 0805	\$0.02	\$0.03
2	RC0805FR-07102KL	RES SMD 102K OHM 1% 1/8W 0805	\$0.02	\$0.03
4	RMCF0805JT120R	RES SMD 120 OHM 5% 1/8W 0805	\$0.02	\$0.07
22	RMCF0805JT1K20	RES SMD 1.2K OHM 5% 1/8W 0805	\$0.02	\$0.37
2	RC0805JR-0712KL	RES SMD 12K OHM 5% 1/8W 0805	\$0.01	\$0.03
4	RC0805JR-071KL	RES SMD 1K OHM 5% 1/8W 0805	\$0.01	\$0.05
2	RMCF0805JT3K90	RES SMD 3.9K OHM 5% 1/8W 0805	\$0.02	\$0.03
2	RC0805JR-0730KL	RES SMD 30K OHM 5% 1/8W 0805	\$0.01	\$0.03
2	ERJ-14YJ103U	RES SMD 10K OHM 5% 1/2W 1210	\$0.02	\$0.04
12	RMCF0805JT150R	RES SMD 150 OHM 5% 1/8W 0805	\$0.02	\$0.20
2	RC0805FR-0711K5L	RES SMD 11.5K OHM 1% 1/8W 0805	\$0.02	\$0.03
8	CRCW20103R30FKEFHP	RES SMD 3.3 OHM 1% 1W 2010	\$0.58	\$4.65
18	RC0805JR-0710KL	RES SMD 10K OHM 5% 1/8W 0805	\$0.01	\$0.23
2	N.A.	Signswise 600p/r Incremental Rotary Encoder Dc5-24v Wide Voltage Power Supply 6mm Shaft	\$17.99	\$35.98
2	PTS645SM43SMTR92LFS	SWITCH TACTILE SPST-NO 0.05A 12V	\$0.08	\$0.15
2	218-6LPST	SWITCH SLIDE DIP SPST 25MA 24V	\$1.35	\$2.70
24	TG6050-10-10-2	THERMAL PAD 10X10X2MM	\$0.59	\$14.23
2	A 6R25M065090	5 mm (HTD) Pitch, 65 Teeth, 9mm wide Single Sided Neoprene Belt with Fiberglass Cords	\$8.53	\$17.06
14	1625854-2	0805 PROBE PAD	\$0.20	\$2.80
4	DAC5571IDBVT	DAC	\$1.52	\$6.08
2	LTC4360CSC8-1#TRMPBF	Ovvoltage Protection Controller	\$2.87	\$5.74

4	LM75BD	Digital temperature sensor and thermal watchdog	\$0.69	\$2.75
2	ACSA03-41EWA-F01	DISPLAY 0.3" SGL 627NM RED SMD	\$1.94	\$3.88
2	LTC6992CS6-3#TRMPBF	Voltage-Controlled Pulse Width Modulator (PWM)	\$3.83	\$7.66
2	ATMEGA32U4-AU	IC MCU 8BIT 32KB FLASH 44TQFP	\$4.12	\$8.24
2	MAX490ESA+	RS485 TRANSEIVER	\$2.52	\$5.04
2	CD4050BDR	IC BUFF/CONVERTER HEX 16SOIC	\$0.42	\$0.84
2	ACS758ECB-200U-PFF-T	SENSOR CURRENT HALL 200A DC	\$7.98	\$15.96
2	PCA9534APWR	IC I/O EXPANDER I2C 8B 16TSSOP	\$1.04	\$2.08
2	R-78HB5.0-0.5	CONV DC/DC 0.5A 5V OUT SIP VERT	\$12.31	\$24.62
4	ADC081C021CIMM/NOPB	ADC081C021CIMM/NOPB	\$1.64	\$6.56
2	A3921KL PTR-T	IC FULL BRIDGE CTLR 28TSSOP	\$4.01	\$8.02
2	MAX232ID	DUAL EIA-232 DRIVERS/ RECEIVERS	\$0.89	\$1.79
2	SN74LVC2G17DBVR	DUAL SCHMITT-TRIGGER BUFFER	\$0.17	\$0.34
2	CLVC1G175MDCKREP	SINGLE D-TYPE FLIP-FLOP WITH ASYNCHRONOUS CLEAR	\$1.11	\$2.22
2	SN74LVC1G04DCKR	SINGLE INVERTER GATE	\$0.09	\$0.17
4	TCA9535PWR	I/O EXPANDER	\$1.45	\$5.80
2	SN74LV393ADR	IC DUAL 4-BIT BIN CNTRS 14-SOIC	\$0.91	\$1.82
2	1040310811	CONN MICRO SD CARD PUSH-PULL R/A	\$1.92	\$3.84
2	503148-2090	Headers & Wire Housings 1.5 W/B Dual RA Rec 20Ckt EmbsTpPkgBeige	\$2.06	\$4.12
2	503149-2000	CONN PLUG DUAL 20CKT BEIGE	\$0.49	\$0.98

2	503148-0890	Headers & Wire Housings 1.5WB DUAL R/A EMBS TAPE PKG 8P	\$1.54	\$3.08
2	503149-0800	CONN PLUG DUAL 8POS BEIGE	\$0.31	\$0.62
2	503148-1490	Headers & Wire Housings 1.5WB DUAL R/A EMBS TAPE PKG 14P	\$1.98	\$3.96
2	503149-1400	CONN PLUG DUAL 14POS BEIGE	\$0.48	\$0.96
16	B6A-PCB-HEX	6 AWG High AMP PCB Wire Lug 6- 16 AWG	\$0.50	\$8.00
2	503148-1090	Headers & Wire Housings 1.5 W/B Dual RA Rec 10Ckt EmbsTpPkgBeige	\$1.27	\$2.54
2	503148-1290	Headers & Wire Housings 1.5 W/B Dual RA Rec 12Ckt EmbsTpPkgBeige	\$1.70	\$3.40
2	FA-238 16.0000MB-C3	CRYSTAL 16.0000MHZ 18PF SMD	\$0.43	\$0.86
				Total \$548.68

Table 88: MDB Revised Material Cost

Proximity Sensor Board:

			Unit	Total
Qty.	Part Num.	Description	Cost	Cost
10	NX2520SA	CRYSTAL 32.0000MHZ 10PF SMD	\$1.01	\$10.12
8	B3U-1000P(M)	SWITCH TACTILE SPST-NO 0.05A 12V	1.00	8.00
1	PG164130	PROGRAMMER MCU PICKIT 3	50.35	50.35
10	L7805ABD2T-TR	IC REG LINEAR 5V 1.5A D2PAK	0.62	6.22
8	MAX490ESA+	IC TXRX RS485/RS422 8-SOIC	2.66	21.28
35	APT2012LZGCK	LED GREEN CLEAR 0805 SMD	0.40	13.89
16	865080257014	CAP 1000 UF 20% 10 V	0.93	14.88
10	GHR-06V-S	CONN GH HOUSING 6POS 1.25MM	0.16	1.63
8	219-4MSTR	SWITCH SLIDE DIP SPST 100MA 20V	0.92	7.36
25	5025780800	CONN PLUG HSG 8CKT BEIGE	0.21	5.30
21	5025840860	CONN RCPT 8CKT VERT BEIGE	1.38	29.02
28	1894	HEX STANDOFF 4-40 ALUMINUM 5/8"	0.49	13.78
28	2207	HEX STANDOFF 4-40 ALUMINUM 2"	1.00	28.08
50	C0805C105K3RACTU	Multilayer Ceramic Capacitors MLCC - SMD/SMT 25volts 1uF X7R 10%	0.08	4.20
16	C0805C200G5GACTU	Multilayer Ceramic Capacitors MLCC - SMD/SMT 50volts 20pF C0G 2%	0.44	6.98
25	C0805C104J5RACAUTO	Multilayer Ceramic Capacitors MLCC - SMD/SMT 50volts 0.1uF 5% X7R AUTO	0.09	2.23
10	C0805C334K3RACAUTO	Multilayer Ceramic Capacitors MLCC - SMD/SMT 25volts 0.33uF 10% X7R	0.13	1.26
250	502579-1000	Headers & Wire Housings CLIKMate Plg Term Gld 24-28AWG	0.09	23.50
2	22-23-2061	Headers & Wire Housings VERT PCB HDR 6P TIN FRICTION LOCK	0.41	0.82
125	3349	Screws & Fasteners #6 NYLON FLAT WASHER	0.04	5.13
10	502584-1460	Headers & Wire Housings CLIKMATE PCB RECPT 14P VT TIN PNP TAPE	2.22	22.20
10	502578-1400	Headers & Wire Housings CLIKMATE PLUG HSG 14P SR BEIGE	0.30	2.99
10	55100-3M-02-A	Industrial Hall Effect / Magnetic Sensors 55100 3M02A HALL SENSOR	5.98	59.80
40	RC0805FR-0710KL	Thick Film Resistors - SMD 10K OHM 1%	0.02	0.64
25	RC0805FR-074K7L	Thick Film Resistors - SMD 4.7K OHM 1%	0.02	0.40
20	RC0805FR-07120RL	Thick Film Resistors - SMD 120 OHM 1%	0.02	0.32
21	RC0805FR-071K2L	Thick Film Resistors - SMD 1.2K OHM 1%	0.02	0.34
14	RC0805JR-073K6L	Thick Film Resistors - SMD 3.6K OHM 5%	0.01	0.18
7	PIC16LF18345-I/SS	Microcontroller for sensor boards	1.06	7.42
1	NB034-0	Actuator for hall effect sensors	1.28	1.28
3	RB-Pli-06	LIDAR Lite v3 GARMIN	149.99	449.97
			Total	\$799.55

Table 89: PSB Revised Material Cost

Weapon Motor Control Board:

Qty.	Part Num.	Description	Unit Cost	Total Cost
4	QBLP631-IB	LED BLUE CLEAR 0805 SMD	\$0.39	\$1.56
1	PIC16F18345-E/SO	IC MCU 8BIT 14KB FLASH 20SOIC	1.38	1.38
1	NX2520SA-32.000000MHZ	CRYSTAL 32.0000MHZ 10PF SMD	1.14	1.14
1	68001-420HLF	CONN HEADER 20POS .100 STR TIN	0.64	0.64
2	PTS645SM43SMTR92 LFS	SWITCH TACTILE SPST-NO 0.05A 12V	0.15	0.30
1	APXW005A0X3-SRZ	DC/DC CONVERTER 3-18V 45W	15.12	15.12
1	NCS1S1205SC	DC/DC CONVERTER 5V 1W	8.46	8.46
1	MAX490ESA+	IC TXRX RS485/RS422 8-SOIC	2.66	2.66
1	SI8710AC-B-IS	DGTL ISO 3.75KV GEN PURP 8SOIC	0.92	0.92
1	ZXGD3002E6TA	IC GATE DRVR IGBT/MOSFET SOT23-6	0.78	0.78
1	HASS 200-S	SENSOR CURRENT HALL 200A AC/DC	22.80	22.80
1	IXGN200N60B3	IGBT 300A 600V SOT-227B	35.45	35.45
1	5031481090	CONN RCPT R/A DUAL 10CKT BEIGE	1.81	1.81
1	5031491000	CONN PLUG DUAL 10CKT BEIGE	0.35	0.35
2	503148-0890	Headers & Wire Housings 1.5WB DUAL R/A EMBS TAPE PKG 8P	1.54	3.08
2	503149-0800	Headers & Wire Housings 1.5WB DUAL PLUG HSG 8CKT BE	0.35	0.70
7	ASC0805-1KF1	RES 1 KOHM 1% 1/8W 0805	0.17	1.19
2	ASC0805-10KF1	RES 10 KOHM 1% 1/8W 0805	0.17	0.34
1	TNPW0805330RDEEA	RES 330 OHM 0.5% 1/5W 0805	0.42	0.42
1	TNPW08054K02BEEN	RES 4.02K OHM 0.1% 1/5W 0805	0.93	0.93
1	RC0805JR-07470RL	RES SMD 470 OHM 5% 1/8W 0805	0.10	0.10
1	RMCF1210FT2R00	RES SMD 2 OHM 1% 1/2W 1210	0.16	0.16
2	RC0805FR-072K2L	RES SMD 2.2K OHM 1% 0.4W 0805	0.10	0.20
2	ESR10EZPF1200	RES SMD 120 OHM 1% 0.4W 0805	0.17	0.34
2	08051A200FAT2A	CAP CER 20PF 100V NP0 0805	0.51	1.02
1	885012007032	CAP CER 47PF 25V C0G/NP0 0805	0.10	0.10
1	08055C333KAT2A	CAP CER 0.033UF 50V X7R 0805	0.10	0.10
3	CC0805KRX7R9BB103	CAP CER 10000PF 50V X7R 0805	0.10	0.30
8	C0805C104K5RACTU	CAP CER 0.1UF 50V X7R 0805	0.10	0.80
4	GRM21BR61E106KA73 L	CAP CER 10UF 25V X5R 0805	0.21	0.84
1	T491C226K025AT	CAP TANT 22UF 25V 10% 2312	0.86	0.86
1	EEE-FK1H470P	CAP ALUM 47UF 20% 50V SMD	0.67	0.67
1	EEE-1EA101UP	CAP ALUM 100UF 20% 25V SMD	0.53	0.53
1	xxxx	3 Boards From OSH Park	40.50	40.50
Total				\$146.55

Table 90: WMCB Revised Material Cost

Mechanical:

			Unit	Total
Qty.	Part Num.	Description	Cost	Cost
2	2617310133	10 Inch AmpFlow Drive Wheel	\$34.00	\$68.00
1	EMS-AGNI-B-95R	Motenergy ME1003 PMDC Motor	\$680.00	\$680.00
28	60355K44	Double Shielded, for 1/4" Shaft Diameter, 3/4" OD	\$6.90	\$193.20
1	91116A160	18-8 Stainless Steel Oversized Washer	\$5.92	\$5.92
28	91125A531	Female Threaded Round Standoff	\$1.39	\$38.92
1	91287A153	18-8 Stainless Steel Hex Head Screw	\$10.50	\$10.50
20	91831A029	18-8 Stainless Steel Nylon-Insert Locknut	\$4.42	\$88.40
2	91845A031	18-8 Stainless Steel Hex Nut	\$7.23	\$14.46
2	91845A330	18-8 Stainless Steel Hex Nut	\$5.61	\$11.22
8	92095A227	18-8 Stainless Steel Hex Drive Rounded Head Screw	\$10.25	\$82.00
112	92141A011	18-8 Stainless Steel Washer	\$2.33	\$260.96
76	92141A029	18-8 Stainless Steel Washer	\$3.37	\$256.12
2	92198A651	18-8 Stainless Steel Hex Head Screw	\$2.03	\$4.06
2	92198A862	18-8 Stainless Steel Hex Head Screw	\$6.40	\$12.80
30	92210A306	Passivated 18-8 Stainless Steel Hex Drive Flat Head Screw	\$5.72	\$171.60
14	92949A271	18-8 Stainless Steel Hex Drive Rounded Head Screw	\$6.16	\$86.24
14	92949A273	18-8 Stainless Steel Hex Drive Rounded Head Screw	\$7.05	\$98.70
28	92949A537	18-8 Stainless Steel Hex Drive Rounded Head Screw	\$4.76	\$133.28

28	92949A539	18-8 Stainless Steel Hex Drive Rounded Head Screw	\$6.12	\$171.36
16	92949A714	18-8 Stainless Steel Hex Drive Rounded Head Screw	\$5.74	\$91.84
73	92949A832	18-8 Stainless Steel Hex Drive Rounded Head Screw	\$7.03	\$513.19
20	92949A835	18-8 Stainless Steel Hex Drive Rounded Head Screw	\$12.04	\$240.80
4	98370A036	18-8 Stainless Steel Washer	\$2.80	\$11.20
1	8770A591	High-Performance Carbide 3 Flute End Mill	\$62.98	\$62.98
2	7972A506	Gas Welding Rod for Aluminum, TIG, 3/32" Diameter, 1 lb.	\$12.31	\$24.62
2	7972A507	Gas Welding Rod for Aluminum, TIG, 1/8" Diameter, 1 lb.	\$12.89	\$25.78
2	92949A106	18-8 Stainless Steel Hex Drive Rounded Head Screw (Pack of 100)	\$2.71	\$5.42
1	92210A105	Passivated 18-8 Stainless Steel Hex Drive Flat Head Screw (Pack of 100)	\$3.24	\$3.24
1	N.A.	0.5" Alumin plate 8.5" by 9"	\$164.30	\$164.30
1	N.A.	1.5" Alumin plate 2'4" by 1'4"	\$220.78	\$220.78
1	N.A.	0.5" Steel plate 1' by 1'6"	\$59.40	\$59.40
2	N.A.	2"x2" 12GA 9' Steel Square Tube	\$48.88	\$97.76
1	N.A.	304 Stainless Steel Half Round 5/8" Diameter 6 feet	\$53.10	\$53.10
Total				\$3,964.86

Table 91: Mechanical Revised Material Cost

5.0.0 Design Schedule

Figure 95, Figure 96, and Figure 97 show the final design project schedule. (MPH)

ID	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1	SDP1 fall2017					
2	Project Design					
3	Preliminary Design Report					
4	Problem Statement	21 days	Tue 8/29/17	Mon 9/18/17		
5	Need	14 days	Tue 8/29/17	Mon 9/11/17	Mack	
6	Objective	14 days	Tue 8/29/17	Mon 9/11/17	Mike	
7	Background	14 days	Tue 8/29/17	Mon 9/11/17	Mack, Mike, Zac, Joey	
8	Marketing Requirements	14 days	Tue 8/29/17	Mon 9/11/17	Mike	
9	Objective Tree	14 days	Tue 8/29/17	Mon 9/11/17	Joey	
10	Rebuild SolidWorks Model	2 days	Tue 9/5/17	Wed 9/6/17	Mack	
11	Block Diagrams Level 0, 1, ... w/ FR tables	14 days	Tue 9/5/17	Mon 9/18/17		
12	Hardware modules (identify designer)	14 days	Tue 9/5/17	Mon 9/18/17	Mack, Joey	
13	Software modules (identify designer)	14 days	Tue 9/5/17	Mon 9/18/17	Mike	
14	Research Pre-Driver ICs	1 day	Wed 9/6/17	Wed 9/6/17	Mack	
15	SolidWorks Model for Weapon Motor Magnet Disk	1 day	Wed 9/13/17	Wed 9/13/17	Mack	
16	Motor Controller Schematic Design	28 days	Wed 9/20/17	Tue 10/17/17	Mack	
17	Research IGBT vs MOSFET Control	2 days	Sun 9/24/17	Mon 9/25/17	Zac	
18	Find Motor Characteristics	1 day	Sun 9/24/17	Sun 9/24/17	Zac	
19	Gather Articles and Parts List	1 day	Sun 9/24/17	Sun 9/24/17	Zac	
20	Determine Max Current of Motor	1 day	Sun 9/24/17	Sun 9/24/17	Zac	
21	Figure Out Star Connect Design for Sensors	1 day	Mon 9/25/17	Mon 9/25/17	Mike	
22	Research Parts for Sensor Board	1 day	Mon 9/25/17	Mon 9/25/17	Mike	
23	Motor Controller Board Layout	46 days	Mon 9/25/17	Thu 11/9/17	Mack	
24	Sensor Board Schematic Design	31 days	Sat 9/30/17	Mon 10/30/17	Mike	
25	Weapon Motor Part Search	6 days	Tue 10/3/17	Sun 10/8/17	Zac	
26	Test Lidar Sensor	5 days	Wed 10/4/17	Sun 10/8/17	Mike	
27	Weapon Motor Controller Schematic Design	38 days	Sun 10/8/17	Tue 11/14/17	Zac	
28	Research Motor Encoders	1 day	Mon 10/9/17	Mon 10/9/17	Mack	
29	Testbed for Sensor System Design	2 days	Wed 10/11/17	Thu 10/12/17	Zac	
30	Research Encode Quadrature Decoder IC	1 day	Wed 10/11/17	Wed 10/11/17	Mack	
31	Midterm Report	28 days	Tue 9/19/17	Mon 10/16/17		
32	Design Requirements Specification	7 days	Tue 9/19/17	Mon 9/25/17	Mack, Mike, Joey, Zac	
33	Midterm Design Gantt Chart	28 days	Tue 9/19/17	Mon 10/16/17	Mike	
34	Design Calculations	28 days	Tue 9/19/17	Mon 10/16/17		
35	Electrical Calculations	28 days	Tue 9/19/17	Mon 10/16/17		
36	Communication	28 days	Tue 9/19/17	Mon 10/16/17	Mack, Mike, Joey, Zac	

Figure 95: Final Gantt Chart

ID	Task Name	Duration	Start	Finish	Predecessors	Resource Names
37	Computing	28 days	Tue 9/19/17	Mon 10/16/17		Mack, Mike, Joey, Zac
38	Control Systems	28 days	Tue 9/19/17	Mon 10/16/17		Mack, Mike, Joey, Zac
39	Power, Voltage, Current	28 days	Tue 9/19/17	Mon 10/16/17		Mack, Mike, Joey, Zac
40	Radiation	28 days	Tue 9/19/17	Mon 10/16/17		
41	Thermal	28 days	Tue 9/19/17	Mon 10/16/17		Mack, Mike, Zac
42	Mechanical Calculations	28 days	Tue 9/19/17	Mon 10/16/17		
43	Structural Considerations	28 days	Tue 9/19/17	Mon 10/16/17		Mack, Mike, Zac
44	System Dynamics	28 days	Tue 9/19/17	Mon 10/16/17		Mack, Mike, Zac
45	Block Diagrams Level 2 w/ FR tables & ToO	7 days	Tue 9/19/17	Mon 9/25/17		
46	Hardware modules (identify designer)	7 days	Tue 9/19/17	Mon 9/25/17		Mack, Mike, Joey, Zac
47	Software modules (identify designer)	7 days	Tue 9/19/17	Mon 9/25/17		Mack, Mike, Joey, Zac
48	Testbed Design for LIDAR Sensor	26 days	Thu 10/12/17	Mon 11/6/17		Mack, Zac
49	Test LIDAR Sensor with testbed	26 days	Thu 10/12/17	Mon 11/6/17		Mike
50	Midterm Design Presentations 9:55-11:35am Part 1	1 day	Tue 10/17/17	Tue 10/17/17		Mack, Mike, Zac, Joey
51	Final Design Report	42 days	Tue 10/17/17	Mon 11/27/17		Mack, Mike, Zac, Joey
52	Abstract	42 days	Tue 10/17/17	Mon 11/27/17		Mack, Mike, Zac, Joey
53	Software Design	42 days	Tue 10/17/17	Mon 11/27/17		Mack, Mike, Zac, Joey
54	Modules 1...n	42 days	Tue 10/17/17	Mon 11/27/17		
55	Pseudo Code	42 days	Tue 10/17/17	Mon 11/27/17		Mike, Mack, Zac
56	Hardware Design	42 days	Tue 10/17/17	Mon 11/27/17		Mack, Mike, Zac, Joey
57	Modules 1...n	42 days	Tue 10/17/17	Mon 11/27/17		
58	Simulations	42 days	Tue 10/17/17	Mon 11/27/17		Mack, Zac, Mike
59	Schematics	42 days	Tue 10/17/17	Mon 11/27/17		Mack, Mike, Zac, Joey
60	Parts Request Form	42 days	Tue 10/17/17	Mon 11/27/17		Mack, Mike, Zac, Joey
61	Budget (Estimated)	42 days	Tue 10/17/17	Mon 11/27/17		Mack, Mike, Zac, Joey
62	Implementation Gantt Chart	42 days	Tue 10/17/17	Mon 11/27/17		Mike
63	Conclusions and Recommendations	42 days	Tue 10/17/17	Mon 11/27/17		Mack
64	Proximity Sensor Board Layout	34 days	Fri 10/27/17	Wed 11/29/17		Mike
65	Review PCB Layouts	43 days	Fri 10/27/17	Fri 12/8/17		Mack
66	Review MDB Gerber Files and Correct Points of Interest	1 day	Sun 10/29/17	Sun 10/29/17		Mack
67	Added New DC/DC Parts	1 day	Wed 11/1/17	Wed 11/1/17		Zac
68	IGBT Driver Alteration	1 day	Sun 11/5/17	Sun 11/5/17		Zac
69	Guide Rail Mounting Hardware	1 day	Sun 11/5/17	Sun 11/5/17		Mack
70	SolidWorks Bill of Material	1 day	Sun 11/5/17	Sun 11/5/17		Mack
71	Create Eagle Libraries for Proximity Sensor Board	1 day	Mon 11/6/17	Mon 11/6/17		Mike
72	Test LIDAR Testbed at 550 RPM	1 day	Mon 11/6/17	Mon 11/6/17		Mike

Figure 96: Final Gantt Chart

ID	Task Name	Duration	Start	Finish	Predecessors	Resource Names
73	Edit Main Driver Board	1 day	Mon 11/6/17	Mon 11/6/17		Mack
74	Weapon Motor Driver Board Schematic Revision	8 days	Tue 11/7/17	Tue 11/14/17		Zac
75	Proximity Sensor Board Layout Revision	9 days	Wed 11/8/17	Thu 11/16/17		Mike
76	Proximity Sensor Board Schematic Revision	7 days	Wed 11/8/17	Tue 11/14/17		Mike
77	Motor Driver Board Schematic Revision	1 day	Wed 11/8/17	Wed 11/8/17		Mack
78	Motor Driver Board Layout Revision	2 days	Wed 11/8/17	Thu 11/9/17		Mack
79	Revise Proximity Sensor Board Parts List	1 day	Tue 11/14/17	Tue 11/14/17		Mike
80	Research Mount Hardware for Proximity Sensor Board	1 day	Tue 11/14/17	Tue 11/14/17		Mike
81	Weapon Motor Driver Board Layout	16 days	Tue 11/14/17	Wed 11/29/17		Zac
82	Weapon Motor Driver Board Review	1 day	Sat 11/18/17	Sat 11/18/17		Mack
83	Write Pseudo Code for Proximity Sensor Board	9 days	Sat 11/18/17	Sun 11/26/17		Mike
84	Main Control Board Schematic	4 days	Wed 11/22/17	Sat 11/25/17		Mack
85	Main Control Board Layout	4 days	Wed 11/22/17	Sat 11/25/17		Mack
86	Find Fab. Plant to Roll Weapon Ring	1 day	Sun 11/26/17	Sun 11/26/17		Mike
87	Final Design Presentations 9:55-11:35am Part 1	1 day	Tue 11/28/17	Tue 11/28/17		Mack,Mike,Zac,Joey
88	Computational Calculations for Proximity Sensor Board	1 day	Tue 11/28/17	Tue 11/28/17		Mike
89	Bill of Materials for Weapon Motor Control Board	1 day	Sat 10/28/17	Sat 10/28/17		Zac
90	Bill of Materials for Main Control Board	1 day	Wed 11/29/17	Wed 11/29/17		Joey
91	Started Main Control Board Shopping Carts	1 day	Wed 11/29/17	Wed 11/29/17		Mack
92						

Figure 97: Final Gantt Chart

Figure 98 and Figure 99 show the actual schedule for Spring of 2018. (MPH)

ID	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1	SDP2 Spring 2018	105 days	Tue 1/16/18	Mon 4/30/18		Mack, Mike, Joey, Zac
2	Revise Gantt Chart	28 days	Tue 4/3/18	Mon 4/30/18		Mike
3	Implement Project Design	97 days	Tue 1/16/18	Sun 4/22/18		Mack, Mike, Joey, Zac
4	Hardware Implementation	56 days	Tue 1/16/18	Mon 3/12/18		Mike, Zac, Mack
5	Populate MCB	15 days	Tue 1/16/18	Tue 1/30/18		Mack
6	Populate WMCB	15 days	Tue 1/16/18	Tue 1/30/18		Zac
7	Populate Extra WMCB	3 days	Tue 4/3/18	Thu 4/5/18		Mike
8	Populate 8 PSBs	10 days	Tue 1/16/18	Thu 1/25/18		Mike
9	Populate 8 Revised PSBs	9 days	Tue 3/13/18	Wed 3/21/18		Mike
10	PSB to MCB Wiring Harnesses	5 days	Mon 3/26/18	Fri 3/30/18		Mack, Mike, Zac
11	MDB to MCB Wiring Harnesses	4 days	Tue 1/16/18	Fri 1/19/18		Mack
12	WMCB to MCB Wiring Harnesses	4 days	Tue 3/13/18	Fri 3/16/18		Zac
13	LIDAR Wiring Harnesses	4 days	Tue 3/13/18	Fri 3/16/18		Mike
14	Hall Effect Sensor Wiring Harnesses	4 days	Tue 3/13/18	Fri 3/16/18		Mike
15	New PSB PCB Layout	1 day	Thu 3/1/18	Thu 3/1/18		Mike
16	RS485 Communication Solution	3 days	Mon 2/26/18	Wed 2/28/18		Mack, Zac, Mike
17	Mount PSBs to Chassis	3 days	Tue 3/20/18	Thu 3/22/18		Mack, Zac, Mike
18	Mount MDBs to Chassis	1 day	Tue 3/6/18	Tue 3/6/18		Mike
19	Mount WMCB to Chassis	1 day	Thu 3/29/18	Thu 3/29/18		Zac
20	Mount MCB to Chassis	1 day	Tue 3/6/18	Tue 3/6/18		Zac
21	Battery and Switches Wiring	4 days	Tue 4/3/18	Fri 4/6/18		Mack, Zac, Mike
22	Mechanical Implementation	97 days	Tue 1/16/18	Sun 4/22/18		Mack
23	Weld Weapon Ring	1 day	Wed 3/21/18	Wed 3/21/18		Mike
24	Weld Weapon Guide Tubes	5 days	Mon 3/26/18	Fri 3/30/18		Mack
25	Weld Impact Bars to Weapon Ring	3 days	Tue 4/10/18	Thu 4/12/18		Mack
26	Drill Holes for Weapon Impact Bars	5 days	Mon 3/19/18	Fri 3/23/18		Mack
27	Cut Out Windows for LIDARs	4 days	Tue 4/3/18	Fri 4/6/18		Mack
28	Drill Holes for PSBs	1 day	Mon 4/2/18	Mon 4/2/18		Mike
29	Drill Holes for Weapon Guide Tubes	5 days	Mon 3/12/18	Fri 3/16/18		Mack
30	Drill Holes for Top and Bottom Covers	4 days	Mon 3/19/18	Thu 3/22/18		Mack
31	Drill Mount Holes for Suspension Blocks	5 days	Mon 3/12/18	Fri 3/16/18		Mack
32	Make Cover for MDBs and WMCB	3 days	Tue 3/27/18	Thu 3/29/18		Zac
33	Drill Holes for Power Switches, MCBS and WMCB	4 days	Tue 4/3/18	Fri 4/6/18		Mack
34	Drill Holes for Locomotion Motors	4 days	Tue 2/20/18	Fri 2/23/18		Mack

Figure 98: Actual Gantt Chart

ID	i	Task Name	Duration	Start	Finish	Predecessors	Resource Names
34	1	Drill Holes for Locomotion Motors	4 days	Tue 2/20/18	Fri 2/23/18		Mack
35	2	Mount Spring Loaded Tension Plate	7 days	Mon 3/26/18	Sun 4/1/18		Mack,Zac,Mike
36	3	Midterm: Demonstrate Hardware	5 days	Mon 3/5/18	Fri 3/9/18		Mack,Mike,Zac,Joey
37	4	3D Print Board Enclosures	43 days	Mon 2/26/18	Mon 4/9/18		Zac
38	5	Mount Power and Autonomous Lights to Chassis	1 day	Tue 4/17/18	Tue 4/17/18		Mack
39	6	Assemble and Disassemble Robot During Testing	28 days	Mon 3/26/18	Sun 4/22/18		Mack,Zac,Mike
40	7	Software Implementation	97 days	Tue 1/16/18	Sun 4/22/18		Mack,Mike,Zac,Joey
41	8	MDB Software Development	97 days	Tue 1/16/18	Sun 4/22/18		Mack
42	9	WIMCB Software Development	97 days	Tue 1/16/18	Sun 4/22/18		Zac
43	10	MCB Software Development	97 days	Tue 1/16/18	Sun 4/22/18		Zac
44	11	PSB Software Development	97 days	Tue 1/16/18	Sun 4/22/18		Mike
45	12	Neural Networks	53 days	Thu 3/1/18	Sun 4/22/18		Mack,Zac
46	13	Midterm: Demonstrate Software	5 days	Mon 3/5/18	Fri 3/9/18		Mack,Mike,Zac,Joey
47							
48	14	System Integration	42 days	Mon 3/12/18	Sun 4/22/18		Mack,Mike,Zac
49	15	Assemble Complete System	14 days	Mon 3/12/18	Sun 3/25/18		Mack,Mike,Zac
50	16	Test Complete System	21 days	Mon 3/26/18	Sun 4/15/18		Zac,Mack,Mike
51	17	Revise Complete System	21 days	Mon 3/26/18	Sun 4/15/18		Mack,Mike,Zac
52	18	Demonstration of Complete System	7 days	Mon 4/16/18	Sun 4/22/18		Mack,Mike,Zac
53	19	Develop Final Report	105 days	Tue 1/16/18	Mon 4/30/18		Mack,Mike,Zac,Joey
54	20	Write Final Report	98 days	Tue 1/16/18	Mon 4/23/18		Mack,Mike,Zac,Joey
55	21	Submit Final Report	1 day	Tue 5/1/18	Tue 5/1/18		Zac,Mack,Mike,Joey
56	22	Project Demonstration and Presentation	1 day	Mon 4/23/18	Mon 4/23/18		Mack,Mike,Zac,Joey
57	23	RoboGames Competition	3 days	Fri 4/27/18	Sun 4/29/18		Mack,Mike,Zac

Figure 99: Actual Gantt Chart

6.0.0 Design Team Information

Below is a list of the design team's members with their majors and project responsibilities.
(MJH)

- Mackenzie Hawkins, Team Leader, Hardware Development, Electrical Engineering
- Michael Hritz, Hardware Development, Electrical Engineering
- Zachary Kilburn, Software Development, Electrical Engineering
- Hong Lau, Archivist, Electrical Engineering and Applied Mathematics

7.0.0 Conclusions and Recommendations

From the research conducted a clear understanding of the project has been established.

With the aid of the patents and various articles discussed above the project goals will be achievable within in a reasonable amount of time. The availability and low cost of microcontrollers and proximity sensors means the cost to complete the project should also be reasonable. The technical design of the project should perform to meet the design requirements proposed. (MJH)

Overall the project was successful on meeting all the design requirements. The robot was able to compete at Robogames and though it did not place well, the ways in which the robot failed in competition gave great insight into how the robot could be rebuilt to be very competitive. (MJH)

The most common failure was the bearing block bending the chassis and becoming lose. The bearing blocks should contact much more surface area on the chassis to attempt to prevent this. Instead of having the bearing block be 0.5" wide, the block should be as wide as possible or the entire length of a section of the octagon that makes the chassis. The block should be secured in with multiple bolts with many levels of redundancy to ensure they do that come loose. The other better option would be to weld the larger bearing bock onto the cassis. (MJH)

Larger bearing or other means of supporting the weapon assembly should also be used. The small contact area of the bearing resulted in dents in the weapon support ring every time the weapon stuck something. Larger contact area would help prevent this. Ideally, the contact area would be the entire width of the weapon suspension tube. The bearing should also be made of a softer material. The original design has the bearing pressed directly against the weapon

suspension tube. A softer roller assembly should be made with a material such as nylon, a hard rubber, or Teflon. (MJH)

The weapon assembly itself could also be improved substantially. The cylinder shape would ideally be made of a single, seamless tube. The original construction method involved rolling a flat piece of aluminum plate into a cylinder and welding the seem together. This method resulted in the cylinder not being perfectly round. The suspension tubes should be made of a harder material such as steel. Being made of aluminum they were very susceptible to deforming and denting. The plates that cover wings that protrude out from the weapon assembly are made from a cold rolled steel plate. This material experienced mage that could be prevented if the material were changed to a hardened tool steel. (MJH)

Improvement to the wire routing in the robot would significantly help the assembly processes. The current design required harnesses to be routed though the battery box. This made assembly and repair of the robot very difficult. The wire routing should be done such that the battery box could be removed without a significant amount of harnesses needing removal. (MJH)

The use of a proper crimp tool would also help the quality of the wire-harnesses. Better quality wire harnesses would help ensure that proper connections are being made and thus contact failure is less likely. During the final round of competition, the robot lost the fight due to the connection between the RC receiver and the MCB becoming loose. The loose connection cut communication between the controller and the robot and resulting in a loss of the round. With better quality crimp connection this failure could have been prevented. (MJH)

A better mechanism to balance the robot should also be implemented. The current design uses stainless steel that skids on the steel floor of the arena. A better solution to this would be Teflon due to its low coefficient of friction and good durability. (MJH)

During competition the portion of the chassis that supports the weapon motor bent down from its original position. This resulted in inadequate pressure on the drive system and thus the weapon couldn't spin. Better support on this part of the chassis needs implemented so that adequate pressure will always be applied to the weapon assembly. (MJH)

The connection between the battery box and the power distribution panel also needs improvements. The connection between the batteries and the harness to the power distribution panel were difficult to implement. A connection that is made when the battery box is inserted would be much more ideal. This would mean that the connection to the batteries are made by simply inserting the battery box assembly. (MJH)

Videos of the robot operating can be seen at the following links,

- <https://www.youtube.com/watch?v=kjFwVxN4GKw>
- <https://www.youtube.com/watch?v=vhkdaqHNcA0>
- <https://www.youtube.com/watch?v=YL6E6RNHnCQ>

8.0.0 References

[1] S. Kumpakeaw, "Twin low-cost infrared range finders for detecting obstacles using in mobile platforms," *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Guangzhou, 2012, pp. 1996-1999.

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6491261&isnumber=6490927>

[2] K. K. Wojewoda, P. R. Culmer, J. F. Gallagher, A. E. Jackson and M. C. Levesley, "Hybrid position and orientation tracking for a passive rehabilitation table-top robot," *2017 International Conference on Rehabilitation Robotics (ICORR)*, London, United Kingdom, 2017, pp. 702-707.

<http://ieeexplore.ieee.org.ezproxy.uakron.edu:2048/stamp/stamp.jsp?tp=&arnumber=8009330&isnumber=8009210>

[3] P. J. Mattaboni, "Autonomous mobile robot." U.S. Patent 4638445A, issued January 20, 1987.

[4] J Gabay, "Sensor-Based Collision Avoidance Solutions for Drone Fleets." *Digi-Key Article Library*, Web. 3 March 2016.

<https://www.digikey.com/en/articles/techzone/2016/mar/sensor-based-collision-avoidance-solutions-for-drone-fleets>

[5] W. Burgard, D. Fox, and D. Hennig, "Fast grid-based position tracking for mobile robots," in *KI-97: Advances in Artificial Intelligence: 21st Annual German Conference on Artificial Intelligence Freiburg, Germany, September 9--12, 1997 Proceedings*, G. Brewka, C. Habel, and B. Nebel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 289–300.

https://doi.org/10.1007/3540634932_23

[6] Victor Abreu, Matthew Beutler, Brent Eisenmann Hisham Hassan, Thomas Schriefer, Peng Xie, "Autonomous Target Tracking Robot," Michigan State University ECE Design Project, 26 April 2013.

<http://www.egr.msu.edu/classes/ece480/capstone/spring13/group07/downloads/files/Final%20Report%20PDF.pdf>

[7] Paul J. Mattaboni "Mobile robot guidance and navigation system" U.S. Patent 5165064 A, issued Nov 17, 1992

<https://www.google.com/patents/US5165064>

[8] Hector H. Gonzalez-Banos, Victor Ng-Thow-Hing, Allen Y. Yang “Interface for robot motion control” U.S. Patent 8060251 B2 issued Nov 15, 2011

<https://www.google.com/patents/US8060251>

9.0.0 Appendices

9.0.1 FastTransfer Arduino Library

```
FastTransfer.cpp                                         Thursday, November 23, 2017 12:37 PM
1   #include "FastTransfer.h"
2   #include "Arduino.h"
3
4   #define DEBUG
5
6
7   //Captures address of receive array, the max data address, the address of the
8   //module, true/false if AKNACKs are wanted and the Serial address
9   void FastTransfer::begin(int * ptr, uint8_t maxSize, uint8_t givenAddress,
10   boolean error, HardwareSerial *theSerial){
11       receiveArrayAddress = ptr;
12       moduleAddress = givenAddress;
13       _serial = theSerial;
14       maxDataAddress = maxSize / 2;
15       sendStructAddress = (byte*)&ring_buffer;
16       AKNAKsend = error;
17       alignErrorCounter = 0;
18   }
19
20   //CRC Calculator
21   uint8_t FastTransfer::CRC8(const uint8_t * data, uint8_t len) {
22       uint8_t crc = 0x00;
23       while (len--) {
24           uint8_t extract = *data++;
25           for (uint8_t tempI = 8; tempI; tempI--) {
26               uint8_t sum = (crc ^ extract) & 0x01;
27               crc >= 1;
28               if (sum) {
29                   crc ^= polynomial;
30               }
31               extract >>= 1;
32           }
33       }
34       return crc;
35   }
36
37   //Sends out send buffer with a 2 start bytes, where the packet is going, where
38   //it came from, the size of the data packet, the data and the crc.
39   void FastTransfer::sendData(uint8_t whereToSend){
40
41       //calculate the crc
42       uint8_t CS = CRC8( sendStructAddress , ring_buffer.count);
43
44       _serial->write(0x06); //start address
45       _serial->write(0x85); //start address
46       _serial->write(whereToSend);
47       _serial->write(moduleAddress);
48       _serial->write(ring_buffer.count); //length of packet not including the crc
49
50
51       //send the rest of the packet
52       for(int i = 0; i<ring_buffer.count; i++){
53           _serial->write(*(&sendStructAddress+i));
54       }
55
56       //send the crc
57       _serial->write(CS);
58
59       //record the sent message data for anak check later
60       crcBufS_put(&crc_buffer, whereToSend, CS, 0);
```

```
61
62     // clears the buffer after a sending
63     ringBufS_flush(&ring_buffer, 1);
64
65 }
66
67 boolean FastTransfer::receiveData(){
68
69     //start off by looking for the header bytes. If they were already found in a
70     //previous call, skip it.
71     if(rx_len == 0){
72         //this size check may be redundant due to the size check below, but for
73         //now I'll leave it the way it is.
74         if(_serial->available() > 4 ){
75             //this will block until a 0x06 is found or buffer size becomes less
76             //then 3.
77             while(_serial->read() != 0x06) {
78                 //This will trash any preamble junk in the serial buffer
79                 //but we need to make sure there is enough in the buffer to
80                 //process while we trash the rest
81                 //if the buffer becomes too empty, we will escape and try again
82                 //on the next call
83                 alignErrorCounter++; //increments the counter whenever a byte
84                 //is trashed
85                 if(_serial->available() < 5)
86                     return false;
87             }
88             if (_serial->read() == 0x85){
89
90                 rx_address = _serial->read(); // pulls the address
91
92                 returnAddress = _serial->read(); // pulls where the message came
93                 //from
94
95                 rx_len = _serial->read(); // pulls the length
96                 //make sure the address received is a match for this module if
97                 //not throw the packet away
98                 if (rx_address != moduleAddress) {
99                     addressErrorCounter++; // increments a counter whenever the
100                     //wrong address is received
101                     //if the address does not match the buffer is flushed for
102                     //the size of
103                     //the data packet plus one for the CRC
104                     for (int u = 0; u <= (rx_len + 1) ; u++) {
105                         _serial->read();
106                     }
107                     rx_len = 0; // reset length
108                     return false;
109                 }
110
111             //if the address matches the a dynamic buffer is created to
112             //store the received data
113             rx_buffer = (uint8_t*) malloc(rx_len + 1);
114         }
115     }
116
117     //we get here if we already found the header bytes, the address matched what
118     //we know, and now we are byte aligned.
119     if(rx_len != 0){
120
121         //this check is preformed to see if the first data address is a 255, if
122         //it is then this packet is an AKNACK
123         if (rx_array_inx == 0){
```

```
111         while( !_serial->available() >= 1));
112         if (255 == _serial->peek()) {
113             CRCcheck();
114             rx_len = 0;
115             rx_array_inx = 0;
116             free(rx_buffer);
117             return receiveData();
118         }
119     }
120
121
122     while( _serial->available() && rx_array_inx <= rx_len){
123         rx_buffer[rx_array_inx++] = _serial->read();
124     }
125
126     if(rx_len == (rx_array_inx-1)){
127         //seem to have got whole message
128         //last uint8_t is CS
129         calc_CS = CRC8( rx_buffer, rx_len );
130
131
132
133     if(calc_CS == rx_buffer[rx_array_inx-1]){//CS good
134
135         // reassembles the data and places it into the receive array
136         // according to data address.
137         for (int r = 0; r < rx_len; r = r + 3) {
138             if (rx_buffer[r] < maxDataAddress) {
139                 group.parts[0] = rx_buffer[r + 1];
140                 group.parts[1] = rx_buffer[r + 2];
141                 receiveArrayAddress[(rx_buffer[r])] = group.integer;
142             } else {dataAddressErrorCounter++;
143             }
144         }
145
146         if (AKNAKsend) { // if enabled sends an AK
147             uint8_t holder[3];
148             holder[0] = 255;
149             holder[1] = 1;
150             holder[2] = rx_buffer[rx_array_inx-1];
151             uint8_t crcHolder = CRC8(holder, 3);
152             _serial->write(0x06);
153             _serial->write(0x85);
154             _serial->write(returnAddress);
155             _serial->write(moduleAddress);
156             _serial->write(3);
157             _serial->write(255);
158             _serial->write(1);
159             _serial->write(rx_buffer[rx_array_inx-1]);
160             _serial->write(crcHolder);
161         }
162
163
164
165         rx_len = 0;
166         rx_array_inx = 0;
167         free(rx_buffer);
168         return true;
169     }
170
171     else{
172         crcErrorCounter++; //increments the counter every time a crc fails

```

```
173         if (AKNAKsend) { // if enabled sends NAK
174             uint8_t holder[3];
175             holder[0] = 255;
176             holder[1] = 2;
177             holder[2] = rx_buffer[rx_array_inx-1];
178             uint8_t crcHolder = CRC8(holder, 3);
179             _serial->write(0x06);
180             _serial->write(0x85);
181             _serial->write(returnAddress);
182             _serial->write(moduleAddress);
183             _serial->write(3);
184             _serial->write(255);
185             _serial->write(2);
186             _serial->write(rx_buffer[rx_array_inx-1]);
187             _serial->write(crcHolder);
188         }
189     }
190
191     //failed checksum, need to clear this out
192     rx_len = 0;
193     rx_array_inx = 0;
194     free(rx_buffer);
195     return false;
196 }
197 }
198 }
199
200
201     return false;
202 }
203
204
205 // populates what info needs sent and to what data address
206 void FastTransfer::ToSend(uint8_t where, unsigned int what) {
207     sendBuf_put(&ring_buffer, where, what);
208 }
209
210
211 // disassembles the data and places it in a buffer to be sent
212 void FastTransfer::sendBuf_put(struct ringBufS *_this, uint8_t towhere, unsigned
int towhat) {
213
214     group.integer = towhat;
215
216     if (_this->count < (BUFFER_SIZE - 3)) {
217         _this->buf[_this->head] = towhere;
218         _this->head = modulo_inc(_this->head, BUFFER_SIZE);
219         ++_this->count;
220         _this->buf[_this->head] = group.parts[0];
221         _this->head = modulo_inc(_this->head, BUFFER_SIZE);
222         ++_this->count;
223         _this->buf[_this->head] = group.parts[1];
224         _this->head = modulo_inc(_this->head, BUFFER_SIZE);
225         ++_this->count;
226     }
227 }
228 }
229
230
231
232 //pulls info out of the send buffer in a first in first out fashion
233 uint8_t FastTransfer::ringBufS_get(struct ringBufS* _this) {
234     unsigned char c;
```

```
235     if (_this->count > 0) {
236         c = _this->buf[_this->tail];
237         _this->tail = modulo_inc(_this->tail, BUFFER_SIZE);
238         --_this->count;
239     } else {
240         c = 0;
241     }
242     return (c);
243 }
244
245
246 //flushes the send buffer to get it ready for new data
247 void FastTransfer::ringBufS_flush(struct ringBufS* _this, const int clearBuffer) {
248     _this->count = 0;
249     _this->head = 0;
250     _this->tail = 0;
251     if (clearBuffer) {
252         memset(_this->buf, 0, sizeof (_this->buf));
253     }
254 }
255
256
257 // increments counters for the buffer functions
258 unsigned int FastTransfer::modulo_inc(const unsigned int value, const unsigned
259 int modulus) {
260     unsigned int my_value = value + 1;
261     if (my_value >= modulus) {
262         my_value = 0;
263     }
264     return (my_value);
265 }
266
267 //searches the buffer for the status of a message that was sent
268 uint8_t FastTransfer::AKNAK(uint8_t module) {
269     for (int r = 0; r < CRC_COUNT; r++) {
270         if (module == crcBufS_get(&crc_buffer, r, 0)) {
271             return crcBufS_get(&crc_buffer, r, 2);
272         }
273     }
274     return 4;
275 }
276
277
278 //returns align error
279 unsigned int FastTransfer::alignError(void) {
280     return alignErrorCounter;
281 }
282
283
284 //returns CRC error
285 unsigned int FastTransfer::CRCError(void) {
286     return crcErrorCounter;
287 }
288
289
290 //returns address error
291 unsigned int FastTransfer::addressError(void) {
292     return addressErrorCounter;
293 }
294
295 unsigned int FastTransfer::dataAddressError(void){
296     return dataAdressErrorCounter;
```

```
297     }
298
299 // after a packet is sent records the info of that packet
300 void FastTransfer::crcBufS_put(struct crcBufS* _this, uint8_t address, uint8_t
oldCRC, uint8_t status) {
301     _this->buf[_this->head] = address;
302     _this->head++;
303     _this->buf[_this->head] = oldCRC;
304     _this->head++;
305     _this->buf[_this->head] = status;
306     _this->head++;
307     if (_this->head >= CRC_BUFFER_SIZE) {
308         _this->head = 0;
309     }
310 }
311
312
313 // after a Ak or NAK is received that status is stored
314 void FastTransfer::crcBufS_status_put(struct crcBufS* _this, uint8_t time,
uint8_t status) {
315     if (time >= CRC_COUNT) {
316         time = CRC_COUNT - 1;
317     }
318     time = time + 1;
319     int wantedTime = time * 3;
320     if (wantedTime > _this->head) {
321         wantedTime = (CRC_BUFFER_SIZE) - (wantedTime - _this->head);
322         _this->buf[(wantedTime + 2)] = status;
323     } else {
324         _this->buf[(_this->head - wantedTime) + 2] = status;
325     }
326 }
327
328
329 // pulls data from the AKNAK buffer
330 uint8_t FastTransfer::crcBufS_get(struct crcBufS* _this, uint8_t time, uint8_t
space) {
331     if (time >= CRC_COUNT) {
332         time = CRC_COUNT - 1;
333     }
334     if (space >= CRC_DEPTH) {
335         space = CRC_DEPTH - 1;
336     }
337     time = time + 1;
338     int wantedTime = time * 3;
339     if (wantedTime > _this->head) {
340         wantedTime = (CRC_BUFFER_SIZE) - (wantedTime - _this->head);
341         return (_this->buf[(wantedTime + space)]);
342     } else {
343         return (_this->buf[(_this->head - wantedTime) + space]);
344     }
345 }
346
347
348 //when an AK or NAK is received this compares it to the buffer and records the
status
349 void FastTransfer::CRCcheck(void) {
350
351     while (!(_serial->available() > 3)); // trap makes sure that there are
enough bytes in the buffer for the AKNAK check
352
353     uint8_t arrayHolder[3];
354     arrayHolder[0] = _serial->read();
```

```
355     arrayHolder[1] = _serial->read();
356     arrayHolder[2] = _serial->read();
357     uint8_t SentCRC = _serial->read();
358     uint8_t calculatedCRC = CRC8(arrayHolder, 3);
359
360
361     if(SentCRC == calculatedCRC){
362
363         for (int rt = 0; rt < CRC_COUNT; rt++) {
364             if (returnAddress == crcBufS_get(&crc_buffer, rt, 0)) {
365                 if (arrayHolder[2] == crcBufS_get(&crc_buffer, rt, 1)) {
366                     if (arrayHolder[1] == 1) {
367                         crcBufS_status_put(&crc_buffer, rt, 1);
368                         break;
369                     } else if (arrayHolder[1] == 2) {
370                         crcBufS_status_put(&crc_buffer, rt, 2);
371                         break;
372                     }
373                 }
374             }
375         }
376     }else{ crcErrorCounter++; } //increments the counter every time a crc fails
377
378 }
```