Micromouse

EE 496 Final Report

Fall 2012

The Engineers:
Brylian Foronda
Darcy Bibb
Ryan Zukeran
Ethan Hottendorf Jr.

December 12, 2012 12:12

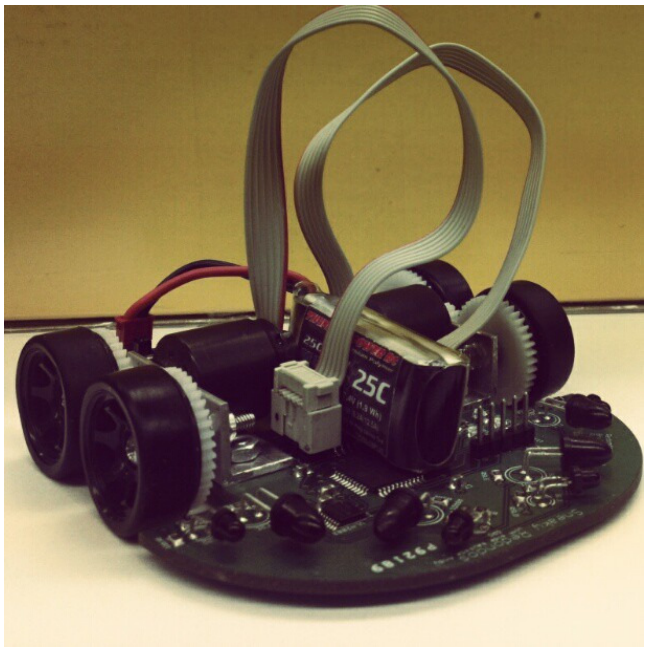
Faculty Advisor: Dr. Tep Dobry

Table of Contents

- Abstract
- Overview
- Objectives
- Design Considerations
- User Manual
- Hardware Components
  - Motors & Encoders
  - Chassis and wheels and tires
- Electrical Components
  - Universal Asynchronous Receiver/Transmitter (UART)
  - Microcontroller
  - Motor H-Bridge Drivers
  - Sensors
  - Power Supply and Battery
- Schematic Design and Component Wiring
  - Power circuit
  - Emitter/Detector Circuit
  - Motor Circuit
  - Microcontroller Circuit
- Board Design
- Software
  - Oscillator Configuration
  - UART Configuration
    - Baud Rates
    - Port Mapping
    - Serial Port Options
    - Changing Baud Rate Settings
  - Sensor Controls
    - ADC Configuration: Manual Sampling and Conversion
  - Motor Controls
    - DC Motor Control with PWM
  - Tracking
    - Timer Configuration
  - Maze Solving Algorithm
- Problems and Shortcomings
- Learning Outcomes
- References
- Datasheets
- Source Code

Abstract

A Micromouse is an autonomous robotic mouse that is designed to navigate and negotiate a path to the center of a maze within an optimal amount of time. The basic design of the mouse consists of a microcontroller, motors and drive circuit, power supply, and sensors, which act as the brain, legs, and eyes of the mouse. We have constructed a Micromouse utilizing a dsPIC33FJ128MC804 microcontroller, brushed dc micromotors, and infrared side sensors. This report provides the design and interface of the different modules that make up the Micromouse, as well as the data that is related to the performance of the mouse.



Overview

The goal of our Micromouse project is to design a self-powered autonomous robotic mouse that will navigate and negotiate its way to the center of a maze. The mouse will need to be built in accordance with "IEEE Micromouse Contest Rules ".

A previous iteration of this project consisted of a unipolar stepper motor driven mouse using infrared sensors and a dsPIC30F microcontroller. The maze solving logic used a modified flood fill algorithm and was able to successfully navigate to the center of the maze within the allotted ten minutes. For this project, the goal is to improve upon the success of our previous mouse and become more competitive.

Objectives

In compliance to the rules set by IEEE, the following objectives have been developed in order for our Micromouse project to succeed and to meet the expectations of an EE 496 project:

- Autonomously find the center of a 16x16 maze from a predetermined starting point and find its way back to the original starting point.
- Complete the maze in less than 10 minutes.
- The body of the mouse must fit inside a single 16cm x 16cm square

Within these rules, the objective of the project is to navigate to the center of the maze in the minimum amount of time.

Approach

In order to accomplish these goals, the Micromouse was divided into four main modules. These four modules are the chassis and motors, the electrical or power components, the electronic and signal components, and the software or programming of the microcontroller. The micromouse chassis will be constructed to house the motors and the electrical and electronic system, including the independent and mobile power supply. The electrical and power system will provide the individual components the necessary power (i.e. volts and current) that is needed in order to operate properly. The sensors will be used to detect walls and openings in the walls as the mouse traverses around within the maze. The microcontroller will keep track of movements and what cells has been previously visited. Through this method, the microcontroller will be used to solve the maze by finding the center of the maze and calculate the quickest time to the center of the maze. The microcontroller will also be responsible for steering the mouse through the maze without impacting the walls. This control of movement is called tracking and is essential for proper operation.

Design Considerations

We chose to design our micromouse around brushed DC motors instead of stepper type motors we had previously used. As such, design of the micromouse's platform and component selection was crucial for successful operation and desirable performance. Many of the design decisions that were made attempted to correct problems or difficulties we experienced our previous design. We also wanted to utilize technologies that allowed for a smaller and more

efficient mouse. These decisions created design constraints which made part selection and the design process more direct.

In our previous mouse design, we used stepper type motors and through-hole components. The circuit design was somewhat simple, but implementation was difficult and resulted in a heavy and bulky mouse. Learning from this, we decided that designing around a printed circuit board (PCB) platform using surface-mount technology (SMT) would be best. A PCB provides a rigid and light platform which can behave as the mouse's chassis, while SMT components are almost always smaller and lighter than their through-hole counterparts. Smaller SMT components and PCB traces instead of insulated wires would allow our mouse design to be quite compact. By designing around a PCB, computer-aided design (CAD) software could be used in the design process. A board designed using CAD software directly results in a final product, whereas going from schematic to perforated circuit boards in our previous mouse was a much less direct process. When using CAD to design a PCB, part placement is performed during the design process and is finalized before the board is fabricated. The final board has solder pads for every component, so we could simply solder components at established locations. However, when building our previous mouse, we had a difficult time with part placement and wiring components together. Without traces, insulated wire was used to connect components on a perforated board. Also, in order to establish the final layout, several iterations were required which meant desoldering and rearranging components. With the use of DC motors and SMT components, the PCB itself could also function as the platform of the micromouse, helping to reduce the weight and bulkiness of the mouse. The motor and

component size are also significantly smaller and lighter, allowing us to have a smaller footprint overall. Having a lighter and smaller mouse means less driving power is necessary and a lower capacity battery could be used.

During design decisions of the micromouse, we also considered experiences on a previous mouse design. In the programming phase of our previous micromouse, we found that the debugging process was difficult without having direct feedback during operation. We learned that by using a dsPIC microcontroller capable of UART communication, direct interfacing could be achieved. A UART module with a wireless transmitter and receiver would allow us to interact with the micromouse wirelessly while it was in operation, making the programming and debugging process much more streamlined. Another consideration from our previous micromouse was the success of the infrared emitter and detector pairs that were used. We did not encounter any major problems with the sensors previously used, so we opted to use the same sensors in this micromouse. However, we did decide to use a transistor array to power and control sensors individually instead of having them constantly on like our previous design.

User Manual

To operate the micromouse, a fully charged battery cell must be used at a voltage level of approximately 7.4 volts. This voltage level is necessary in order to provide sufficient power to the electronic circuitry, DC motors, and IR sensors. With the battery cell securely located between the motors of the mouse, connect the battery to the power connectors located at the right rear corner of the mouse adjacent to the ON/OFF switch. When performing this

connection, be sure that the correct polarity (+ positive/ - negative) is made. The correct polarity is silk screened on the PCB. Once this connection has been made and verified to be correctly installed, place Lil Vienna in the center of the starting square if solving the maze is desired. The performance of the mouse will be observed if Lil' Vienna is placed in a random square. Note that this placement, however, will not solve the maze. Once it is situated in the middle of the square and facing straight ahead, slide the switch to the ON position, or to the right side of the switch. At this point, the mouse should be operating. Should it crash and stop moving at any point during the operation of the mouse, slide the mouse to the OFF position (left side of the switch) as soon as possible to prevent damage to the motors. The mouse can then be placed back at the original square and turned back ON. When the use of the micromouse is complete, be sure to slide the switch to the OFF position and disconnect the battery pack from the mouse.

Hardware Components

Motors & Encoders

Most micromice are based on either stepper motors or DC motors. For this final project, we used DC motors. Our main design goal was to keep the micromouse small and compact. We decided to utilize motors from Faulhaber due to the available encoders. These encoders, IE2-512, provide resolution of 512 pulses per rotation while only adding 1.4 mm to the overall length of the motor. We decided to go with the Faulhaber 1717T 6V DC micromotors due to its lightweight nature and high speed output performance. Other motors that were considered were the 1524T and 1724T. They both are 24 mm in length as opposed to the 1717T's 17 mm length. The 1524T weighs 21 grams while the 1724T weighs 27 grams. The 1717T weighs 18 grams. The 1524T provides a stall torque of 7.12 mNm, the 1724T 13.2 mNm, and the 1717T 5.38 mNm. Our decision to use the 1717T was based on the size and weight to torque ratios. The low torque output could be easily compensated by gearing the motors accordingly.

Chassis and wheels and tires

The chassis for the mouse is the four-layer etched PCB which also contains the electrical components. The reason for this is to minimize the weight of the robot. The four-layer PCB is strong and rigid enough to provide a suitable platform on which everything is connected. Motor mounts are attached to the chassis so that the motors and reduction gearing can be fastened. These mounts were constructed with ½" x ½" aluminium L-shaped brackets. This material could be easily shaped and cut to the specification we required.

Two millimeter diameter screws were used as wheel shafts. Kyosho Mini-Z wheels were chosen because the small size of the wheels provide three millimeters of ground clearance. These wheels were glued to the spur gears and fastened to the wheel shafts with nuts. The overall diameter of the wheel and tire measures 22 mm.

To have sufficient torque to propel the mouse, a reduction gearing needed to be used to convert the high RPM of the DC motors to slower speeds for the wheels of the mouse. According to Figure 1, maximum torque is achieved at zero rotations per minute and at no load speed, torque is zero. To provide sufficient speed while still allowing for flexibility, we chose a gearing ratio of 40:12 for this purpose. At the desired speed of 2 m/s, a wheel rotation of 1736.6 rpm is required and therefore, a motor speed of 5787 rpm is needed. At this motor speed, the torque curve is nearly at the middle of the linear plot.

$$Motor\ Speed = 2\ \frac{m}{s} \times \frac{60\ sec}{1\ min} \times \frac{1\ rotation}{(0.022 \times \pi)m} \times \frac{40}{12} = 5787\ rpm$$

Equation 1: Required motor speed for 2 meters per second ground speed.

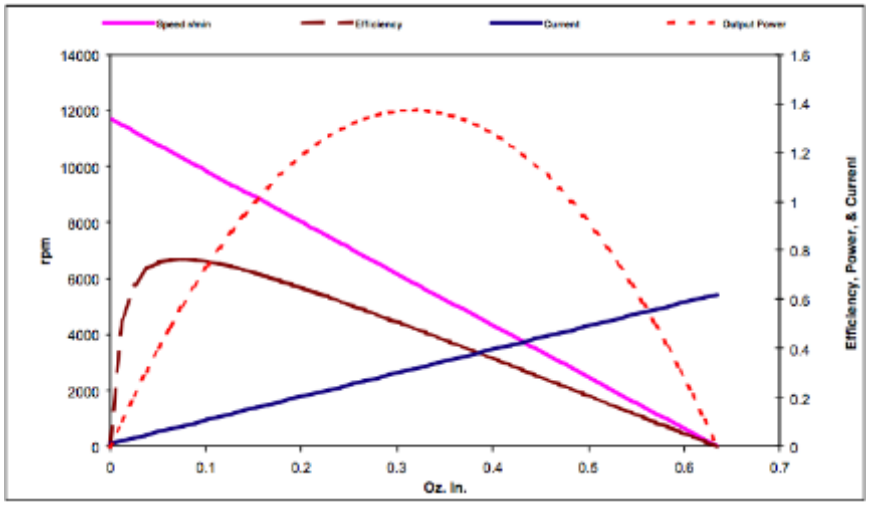


Figure 1: DC Motor Torque Curve.

Electrical Components

Universal Asynchronous Receiver/Transmitter (UART)

The Universal Asynchronous Receiver/Transmitter (UART) is a type of computer hardware that translates data between parallel and serial forms. The UART controls a computer's interface to its attached serial devices and provides the computer with the RS-232C Data Terminal Equipment interface so that data can be exchanged with serial devices. As part of this interface, the UART convert bytes of data it receives into single serial bit stream for outbound transmissions. On inbound transmission, the serial bit stream is converted into complete bytes that a computer handles. Having a microcontroller that includes a UART module is beneficial for micromouse communication. Sensor values, motor positions, duty cycles and other important information can be displayed on a computer screen wirelessly during mouse operation.

There are two options for selecting wireless UART connections: bluetooth-to-serial or USB-to-serial. Since not all computers come with bluetooth capability, we decided to employ wireless communication using two wireless XBee modules which uses IEEE 802.15.4 protocol stack. One XBee would be connected to the microcontroller and the other would be connected to the computer. Specifically, we chose the XBee wireless communication kit by Digi International which included two XBee modules with chip antenna and the popular USB XBee explorer that provides direct access to the serial and programming pins of the XBee unit. This complete XBee kit made wireless communication simple.

Microcontroller

Considering all of the desires and requirements established for this micromouse, the rest of the component selection needed to be made. Our selection of DC motors requires a microcontroller capable of generating pulse-width modulated (PWM) signals to drive the motors. The IE2-512 encoders used on the DC motors require the microcontroller to be able to process quadrature encoder interface (QEI) signals to interpret the speed and position of the motor shaft. We also needed to be sure that the microcontroller selected will have UART support if we want to communicate with the micromouse during operation. As we plan to have at least four infrared detectors, the microcontroller needs to have four analog inputs to process the detector signals. There needs to be at least four ports dedicated to driving the infrared emitters on the micromouse.

Experienced Micromouse designers tend to utilize the 32-bit STM32F series of processors using the ARM Cortex M4 due to its high processing speed, large memory space, and a floating point unit. However, creating programs and configuring the registers are more complex when compared with Microchip's dsPIC series so to allow for more testing and debugging, the STM32F was not an optimal choice considering the timeframe for this project.

Based on these requirements, we found that Microchip's dsPIC33FJ128MC804 microprocessor in an SMT package fulfilled all of our requirements for input and output ports. This microcontroller operates at 3.3V with a processing speed of 40 MIPS.

Motor H-Bridge Drivers

In order to drive the motors from a signal generated by the microcontroller, a motor

driver is needed to interface the motors with the output of the microcontroller. To drive DC motors, an H-bridge can be used, providing both forward and reverse operation. Figure 2 shows the structure of an H-bridge with a load (M) connected in the H structure. Depending on the state of the transistors, current will flow in one direction through the load. An H-bridge

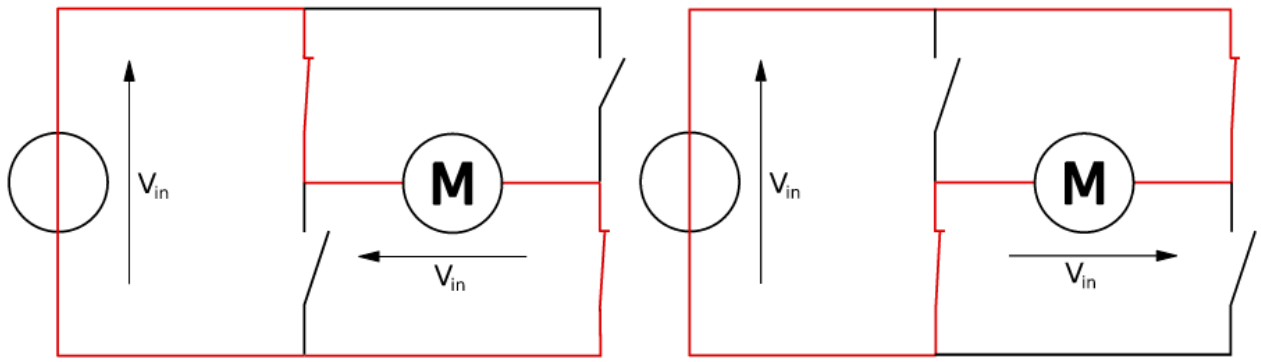


Figure 2: H-Bridge diagram.

device capable of operating at the nominal 6V of the DC motors was needed, as well as having fast switching speeds. For these reasons, the Zetex ZXMHC3A01T8 MOSFET H-bridge was selected as our motor drivers. This H-bridge device is available in an SM8 SMT package, and features low gate drive, fast switching speeds, and are good for applications up to 30V. However, the dsPIC33F microcontroller that we selected operates at 3.3V levels. While the H-bridge can be used to drive the motors, further interfacing is necessary to translate the signal from the microcontroller. MOSFET drivers can be used for this purpose, but in our application must be high speed and able to translate from CMOS levels to the voltage of our motors, while providing enough current. We found that the Maxim MAX4427 MOSFET drivers were excellent under these requirements and available in an SOIC8 SMT package. They can translate CMOS

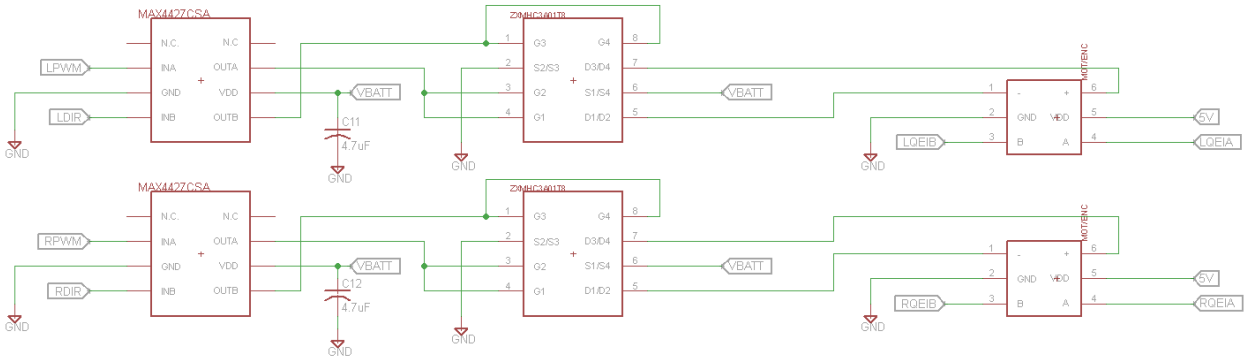


Figure 3: Schematic showing H-bridge to MOSFET connections.

levels to up to 18V, have up to 1.5A output current, and have extremely short delay times (typically less than 50 nanoseconds). Pairing the Zetex H-bridge and the Maxim MOSFET drivers provides a suitable interface between the microcontroller and DC motors.

Sensors

When researching different options for measuring the presence and distance of walls, we found most side-sensing mice choose to use Sharp GPD120 distance sensors. We also found that these sensors are not linear and are prone to erroneous values at close ranges. These sensors also have a fairly high cost. Primarily to keep costs down and the design compact, we decided to use individual infrared LEDs and infrared phototransistors which operate at the same wavelength. Specifically, SFH4545 emitters paired with TEFT 4300 phototransistors were used. Other options were not considered because we used these components on our previous micromouse with success. These matched pairs can then be used as a way to measure the distance of an object. The amount of reflected light depends on the distance of the reflecting

object. The amount of reflected light can then be measured as a voltage using a phototransistor. By converting this analog voltage to digital data in the microprocessor, we now have a means to not only detect the presence of an object, but also the distance of the object.

Instead of having our infrared emitters on all the time, we decided that we would like to be able to control the sensors individually when needed. Enabling the emitters only when needed consumes less current. It would also prevent the emitters from overheating as in the case of driving them full-time. This also provides us with a way to obtain more accurate sensor readings. By taking a reading with the emitters off, then subtracting this value from the real readings, allows us to mitigate errors due to ambient light. In order to switch the emitters from the microcontrollers output ports, a transistor switch must be used to provide the 100mA necessary for the emitters, as the microcontroller can only supply up to 20mA per port. A ULN2003A darlington transistor array was a cheap option available in an SO16 SMT package. This transistor array has seven transistors available which we could use to drive the infrared emitters. This array provides up to 500mA of output current per driver and low switching times (typically 0.25µs).

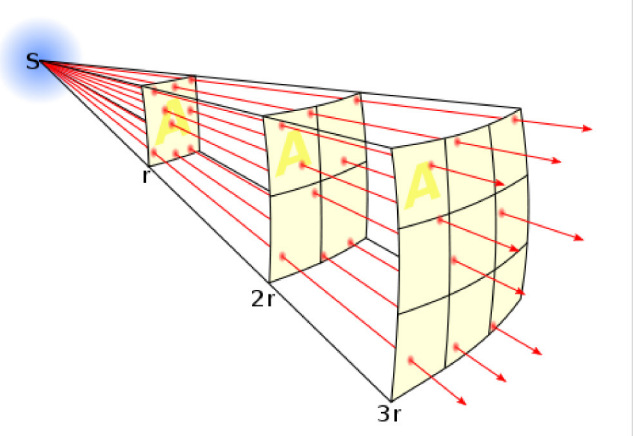
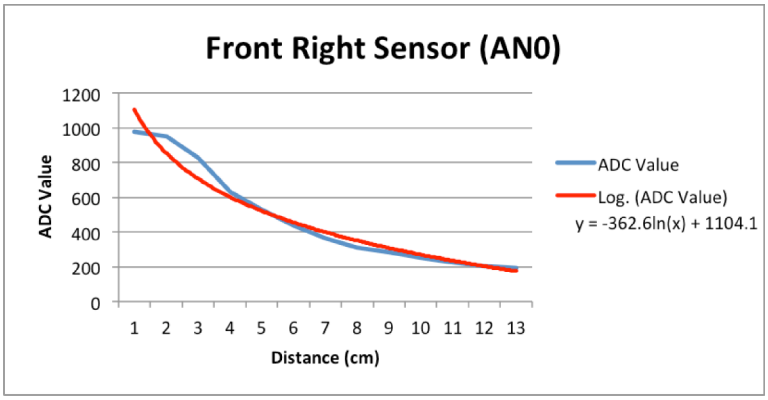


Figure 4: Illustration showing the square relationship between distance and illuminated area.

The amount of light reflected from a surface depends on the distance of the reflecting surface from the light source. Figure 4 illustrates the squared relationship of a point source and the illuminated area at some distance away from the source. Because of this inverse square law of light, the reflected infrared light detected from walls at varying distances will not result in a linear behavior. Also, the complete intensity of the beam will not be reflected due to absorption of the maze walls and angle of incidence. Therefore, measurement from the phototransistors will not form a linear plot with distance. The measurement of the infrared light that is reflected is an important factor when using a linear control system, such as a proportional controller for tracking. Figure 5 illustrates the non-linearity of measured values with a linearly changing distance. These values were obtained through experimentation, where ADC values were measured at incremental distances. By analyzing the data plots, an inverse exponential function gave an accurate relationship between distance and sensor reading. The linearizations are shown in Figure 5 below.



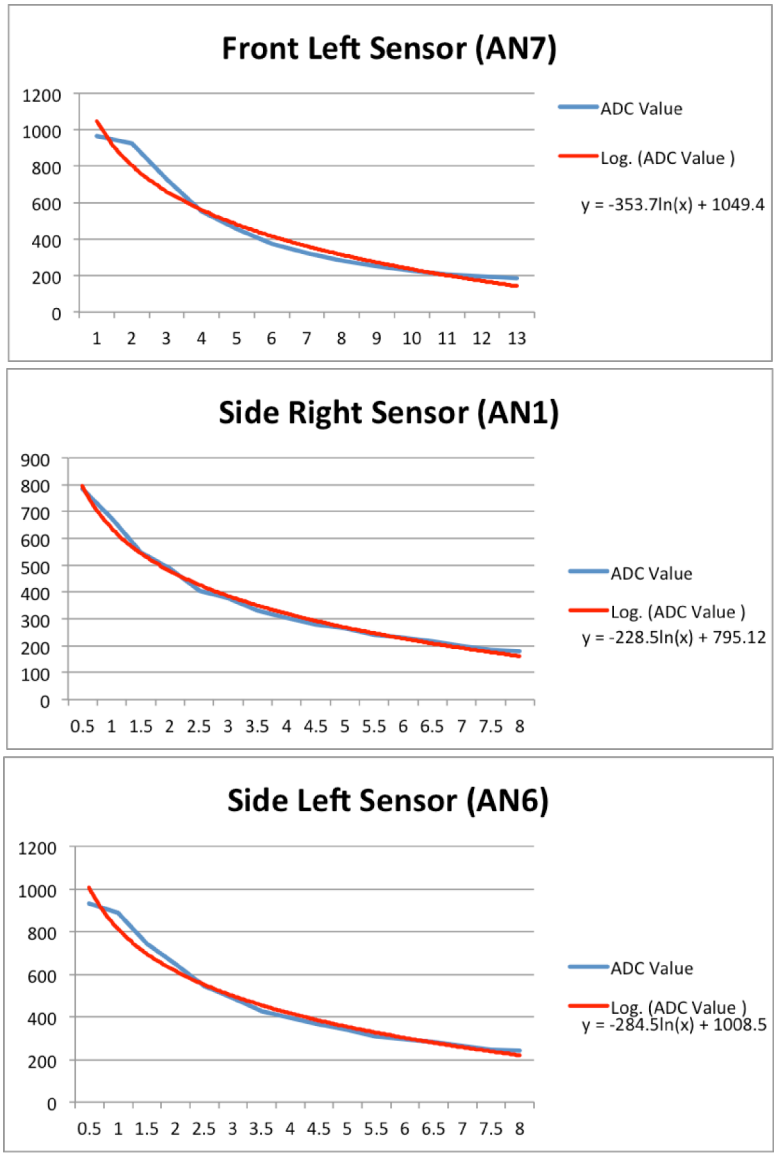


Figure 5: Sensor Function Calibration.

Power Supply and Battery

With all of the major components of the micromouse determined, power components need to be considered to supply power to the whole board. Reviewing each component, there are three voltage levels required on our board. The UART module and DC motor encoders operate at 5V. The microcontroller requires 3.3V for operation. The DC motors are specified to

run at about 6V. Rather than connecting the motors through the 5V voltage regulator, we

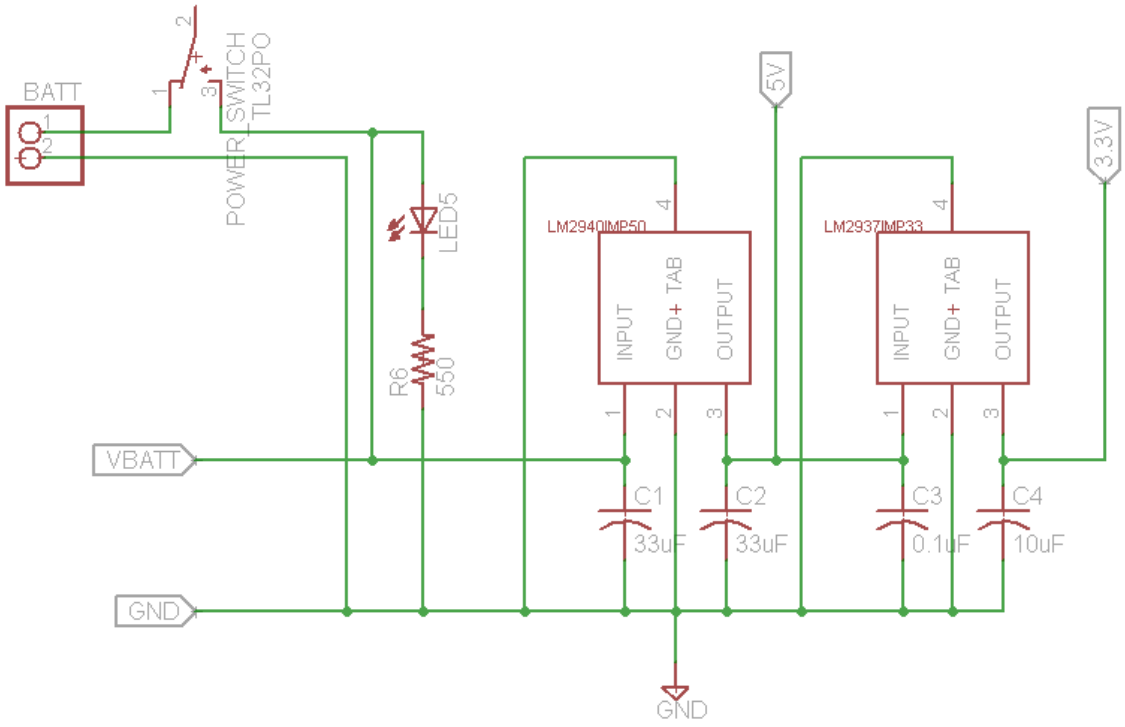


Figure 6: Schematic of power components.

decided to operate them at battery level. This would prevent the 1A current limitation of the 5V LM2940 voltage regulator, thus supplying more driving power, and isolate the motors from the other components. Due to the voltage requirements of the UART module, motor encoders, and microcontroller, we would need to have at least 5V and 3.3V supplies on our board. Since we need to have both of these voltage levels on our circuit, we decided to operate the infrared sensors on 5V instead of the 3.3V circuit to isolate them from the 3.3V analog reference of the microcontroller.

During the micromouse competition, the robot is required to operate for a time period of

10 minutes. The selection of the battery size and type would be dependent upon the maximum current consumption. According to the motor datasheet, maximum power consumption is 1.96W at 6V. Therefore, for two motors operating for 10 minutes, a current consumption of 108.89 mAh is expected.

$$Motor\ Current\ Consumption\ = 1.96W\ \times \frac{1}{6V}\ \times 2\ Motors\ \times 10\ Minutes\ \times \frac{1\ Hour}{60\ Minutes} = 108.89mAh$$

Equation 2: Motor Current Consumption.

The infrared emitters used for detection are designed for a maximum current of 100 mA. For every 15 ms, the emitters are turned on for 5 ms sampled, then turned off. Therefore, the duty cycle is 33.33%. If it is assumed that the four emitters are kept on for the 10 minute duration, a current consumption of 240 mA is calculated.

$$Emitter\ Current\ Consumption\ =\ 4\ Emitters\ \times 100mA\ \times 10\ Minutes\ \times \frac{1\ Hour}{60\ Minutes} \times 33.33\%\ = 22.22\ mAh$$

Equation 3: Infrared Emitter Current Consumption.

While debugging the micromouse, the XBEE UART transmitter is used. This component consumes a calculated 7.5 mAh as shown in Equation 4 below.

$$XBEE\ Current\ Consumption\ =\ 45mA\ \times 10\ Minutes\ \times \frac{1\ Hour}{60\ Minutes} = 7.5\ mAh$$

Equation 4: XBEE UART Transmitter Current Consumption.

The infrared emitters, DC motors, and UART transmitter are the major current consumers in the mouse. The other circuit components are negligible. Maximum current consumption for

10 minutes of operation is calculated to be 138.61 mAh. A lithium polymer battery was chosen to minimize the weight and size of the onboard battery. To achieve the required 6 volts for the motors, we chose a two-cell battery which provides a supply of 7.4 volts. The capacity of this battery is 250 mAh. This allows for more than sufficient capacity to run the mouse in competition. The weight of the battery is 17 grams, allowing the weight of the mouse to be minimized. The small size of the battery, 1cm x 2 cm x 4 cm, fits conveniently on the compact chassis between the DC motors.

Schematic Design and Component Wiring

All schematic and board design was performed with the use of CadSoft's EAGLE software. The EAGLE software makes it incredibly easy to transition from schematic to board design. EAGLE includes a schematic design platform and a board layout editor. By using parts from existing or user-created libraries, parts placed in the schematic editor automatically have the same part placed in the software's board editor with the appropriate footprint. Parts wired together in the schematic editor also have a tether created to indicate connecting pins in the board editor. This makes it very simple to design the PCB itself once the all parts are placed and wired in the schematic. Figure 7 illustrates the schematic editor in the EAGLE software.

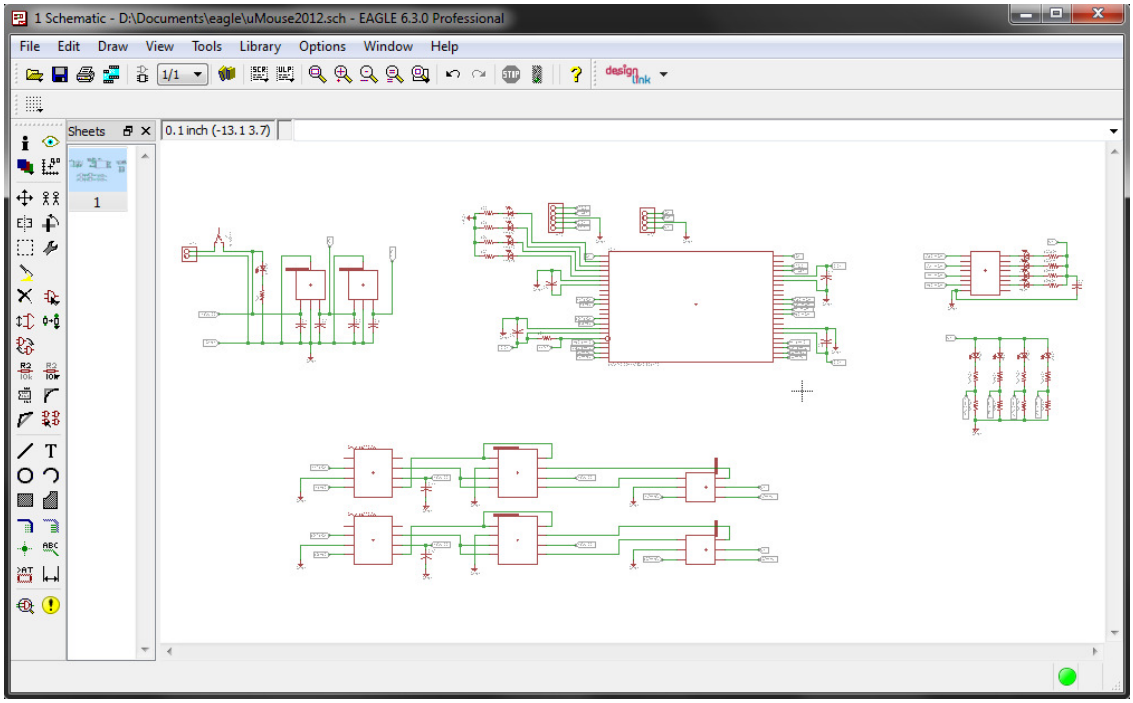


Figure 7. CadSoft EAGLE: schematic editor.

Before beginning the placement of parts and wiring components in the schematic, it was important to ensure that all components we planned to use on the board were available to use in the EAGLE software. Many common parts, such as resistors and LEDs, are available in the default library. Other components, such as common ICs like the ULN2003A we used to drive our infrared emitters, could be found in publicly available user-made libraries. However, several of the components we used needed to be manually created in the software. This means that a schematic component needed to be made, and a board footprint needed to be created to use on the final board design. By using existing components and footprints as a template, libraries were created for each missing part. After each part is created or ensured to be available, the schematic could be designed. In creating the circuit schematics, parts are simply added to the schematic editor and wired appropriately to other components.

Power circuit

The power circuit consists of the battery, a sliding switch, and the 3.3V and 5V voltage regulators. An LED and 550Ω resistor were employed to indicate the power status of the circuit. The positive terminal of the battery is supplied to one side of the switch, and the other side to the input pin of the 5V voltage regulator. The output of the 5V voltage regulator provides the 5V level of the circuit. This 5V level is also provided to the input of the 3.3V regulator. The output of this voltage regulator supplies 3.3V to the rest of the circuit. The body and ground pins of the voltage regulators are connected to the negative terminal of the battery. This terminal will be the ground reference to the whole circuit. Capacitors are used at the inputs and outputs of the voltage regulators for noise reduction and decoupling. The schematic of the power circuit is

shown on Figure 8 below.

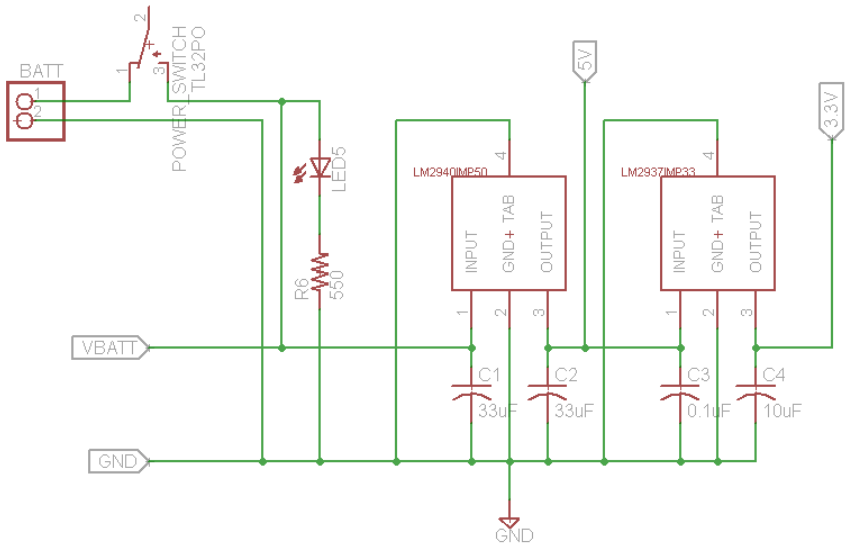


Figure 8: Power Circuit Schematic

Emitter/Detector Circuit

The emitter circuit consists of four infrared emitters, 30Ω resistors, and a darlington transistor array. Four control signals are connected from outputs of the microcontroller to the inputs of the transistor array. Emitters are connected to the transistor array such that the cathode of each emitter is connected to the array's output, and the anode is connected in series with a resistor to 5V. GND pin 8 is connected to the board's ground. In this configuration, the emitters are turned on when the darlington transistor's input is 5V, which causes the transistor's output to go to 0V and provide a path for the current through the emitters. A decoupling capacitor is used between the 5V supply to the emitters and ground.

The detector circuit consists of four infrared detectors, and voltage divider circuits for signal sensing. The collector of the detector is connected to 5V supply, and the emitter of the

infrared detector is connected in series with the voltage divider resistors to ground. The detector's voltage signal is taken between the two resistors and provided to the analog to digital converter of the microcontroller. The resistor values are chosen such that at maximum light detection, the voltage will be divided from 5V and the voltage across the bottom resistor will be 3.3V. Using the equation $3.3V = 5V * R2/(R1 + R2)$, we get that $R2 = R1 * 33/17$. The exact values are determined through experimentation with different ranges. The final resistor values were determined to be $R1 = 1k\Omega$ and $R2 = 1.941k\Omega$. As more infrared light is detected, the voltage seen at the output will be higher. The schematic of the emitter and detector pairs are shown on Figure 9 below.

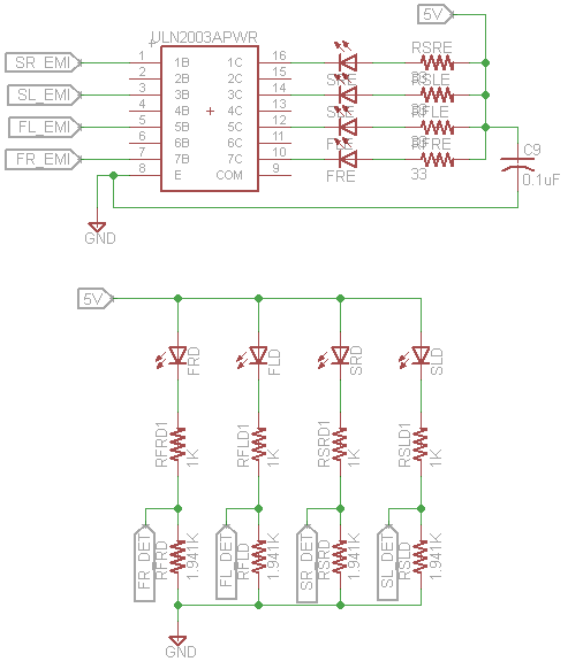


Figure 9: Emitter and Detector Pairs Schematic

Motor Circuit

The motor and motor driver circuit consists of a MAX4427 MOSFET driver, Zetex H-bridge driver, and the 6-pin header connection which interfaces with the DC motor and encoders.

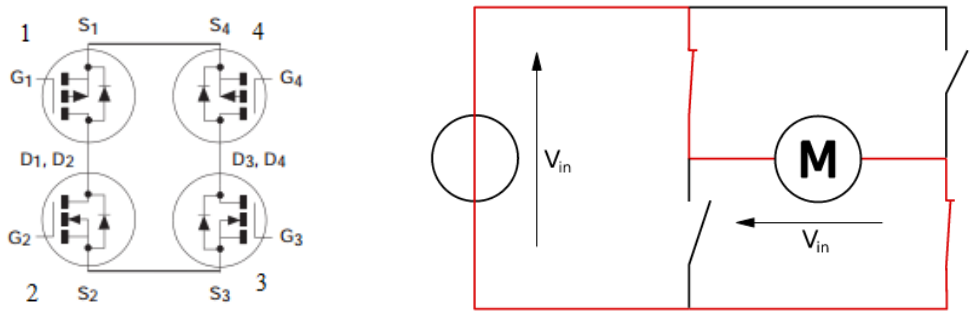


Figure 10: H-bridge structure of Zetex driver.

Referring to Figure 10, the motor is connected between D1,D2 and D3,D4 of the H-bridge device. Transistors 1 and 4 are N-channel devices, while transistors 2 and 3 are P-channel devices. In order for current to travel in one direction through the motor load, transistors 1 and 3 need to be closed and transistors 2 and 4 need to be opened. For current to travel in the opposite direction, transistors 1 and 3 need to be open while transistors 2 and 4 need to be closer. This operation provides the forward and reverse motor directions. In order to achieve this, PWM and complementary PWM signals from the microcontroller are supplied to the input channels of the MOSFET driver. The gates of transistor 1 and 2 are tied together, and the gates of 3 and 4 are tied together. When a signal is applied to gates 1 and 2, one or the other of the transistors will be closed. If the inverse signal is then applied to gates 3 and 4, the desired operation is achieved. Either gates 1 and 3 will be closed, or gates 2 and 4 will be closed. The

signal provided can then be manipulated in the microcontroller to control speed and direction. However, the H-bridge driver cannot be driven directly by the microcontroller. Because the motors are operated at 7.4V of the battery, the bias voltages at the gates of the H-bridge transistors must also be 7.4V. In order to translate the 3.3V output of the microcontroller to 7.4V, a MOSFET driver is used. The output signals from the MOSFET driver are then fed to the H-bridge driver as previously described. QEI channels A and B of the header are provided to the QEI inputs of the microcontroller. The DC motors utilize a 6-pin connector which connects directly to the 6-pin header. Figure 11 below illustrates the motor and encoder circuit schematic.

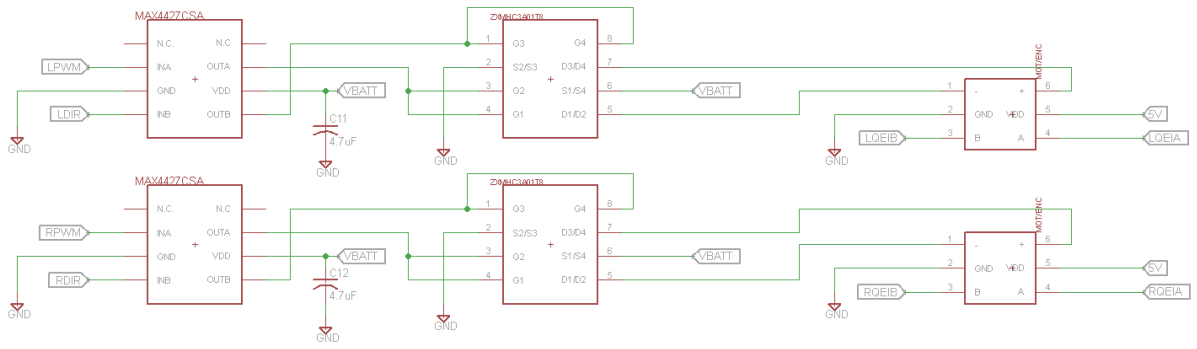


Figure 11: Motor and Encoder Circuit Schematic

Microcontroller Circuit

The microcontroller circuit consists of the dsPIC microcontroller, four debug LEDs, the programming interface, and the UART module. Connections between the microcontroller and

other components on the mouse are established at this point. Pins 2-5 of the dsPIC are used to drive the debug LEDs. These are four SMT LEDs driven by the microcontrollers output ports RC6-RC9 at 10mA each, and used for debugging purposes. The UART module uses 5V from the board's power, and the transmitter and receiver signals are connected to pins 1 (port RP9) and 44 (port RP8) of the microcontroller, respectively. The programming interface of the microcontroller uses a five pin header, which is connected according to Figure 12. Pin 1 of the header is connected to pin 18 (MCLR) of the microcontroller for reset during programming. Pins 4 and 5 of the header are connected to pins 41 (PGED3) and 42 (PGEC3) for programming of the microcontroller from the Pickit3 programmer.

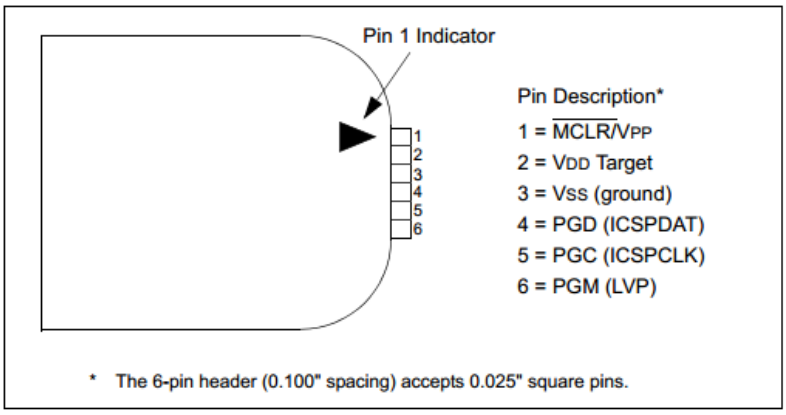


Figure 12: Pickit3 programmer pinout.

Other ports are reserved and designated for connections to the motor circuit and infrared sensor circuit. Ports PWM1H2 and PWM1L2 are used to interface with the left motor, and Ports PWM1H1 and PWM1L1 are used to interface with the right motor. Ports RB0 through

RB3 are used for QEI feedback from the motor encoders. Ports RA8, RB4, RA4, and RA9 are used to interface with the infrared emitter circuit. Finally, ports AN0, AN1, AN6, and AN7 are used for analog feedback from the infrared detector circuit. The microcontroller circuit is shown in Figure 13 below.

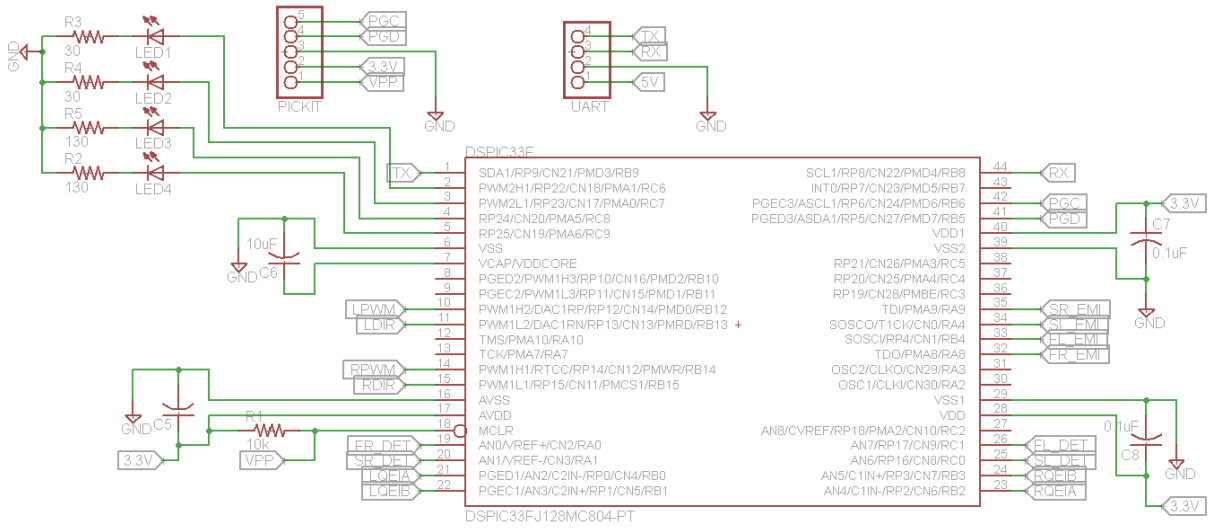


Figure 13: Microcontroller circuit schematic.

Board Design

Design of the PCB was performed using Cadsoft's EAGLE software. Once the schematics were completed, as described in the Schematic Design and Component Wiring section of this document, the outline and final design of the mouse's board could be completed using the board editor in the EAGLE software. To transition from the schematic editor to the board layout editor in EAGLE, we simply select the option to "Switch to board" from the schematic editor, as illustrated in Figure 14.

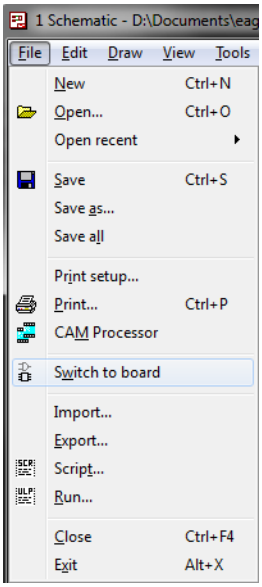


Figure 14: Switching from schematic to board editor.

Once we transition from the schematic editor to EAGLE's board layout editor, we are presented with the footprints of all the parts and a rat's nest of all connections. Figure 15 illustrates the board editor in the EAGLE software, and the initial state of the board after transition from the schematic. Also shown in Figure 15 are the air-wires, which indicate the

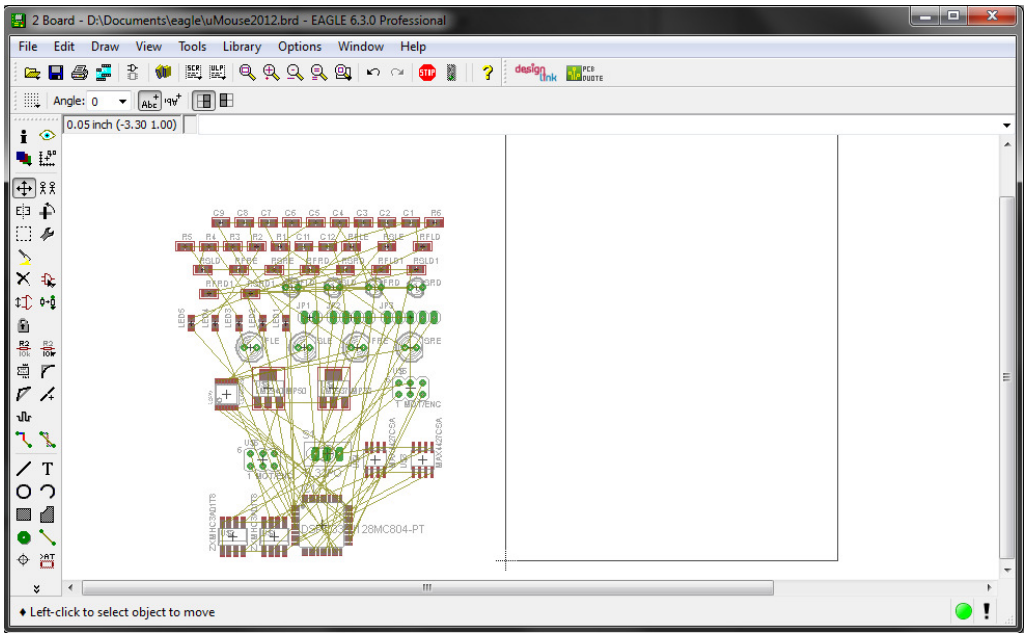


Figure 15: CadSoft EAGLE: board editor.

connections needed between each component. At this point, the first step is to establish the desired outline of the PCB. As our PCB will behave as our mouse's chassis and platform, we decided to implement a simple design with a round front edge. A rounded front edge would behave as a sort of bumper, and help prevent the mouse from being immobilized if there is contact with a wall or corner. Figure 16 shows the final board outline for our design. Space was also allocated for hardware to mount the motors and wheels of the mouse.

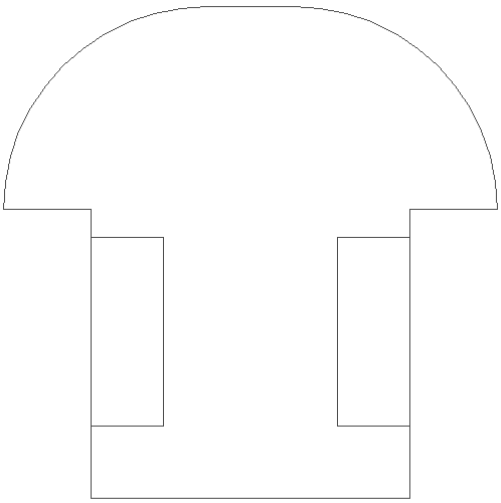


Figure 16. Final PCB design outline.

After the outline of the PCB was established, we could then place components within the board outline in EAGLE's board editor. Certain components were restricted to certain areas on the board. The emitters and detectors needed to be placed on the front of the PCB, to allow for front and side sensing. We placed the ULN2003A darlington transistor array on the front of the board to keep it close to the emitters and minimize the distance traces needed to be routed. We also wanted to keep the battery on the rear of the mouse in order to keep it from interfering with other components. This resulted in implementing the power supply circuit on the rear of the mouse, including the power switch and voltage regulators. Space needed to be reserved to allow room for the motors to be mounted, so much of the rest of the components were placed around the front half of the board. Based on these restrictions, we reached a final design with part placement shown in Figure 17.

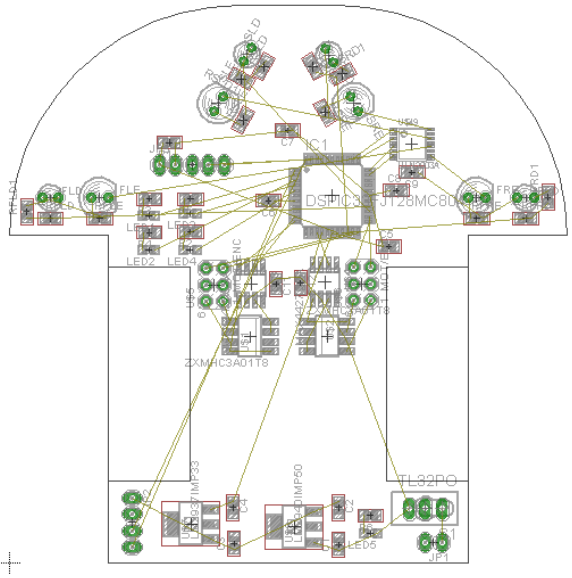


Figure 17. Board outline with final component placement.

Once the placement of parts was finalized, we could then begin routing traces for connections between components. We planned to have our board fabricated by Advanced Circuits[1], and they specify that trace widths must be at least 0.006in and spacing between traces must be at least 0.006in. Using a PCB trace width calculator[4] for external layers, we determined that signal traces should be 0.012in and high current traces used in driving the motors should be 0.016in. Using these parameters, we routed traces between pins of each component using the yellow air-wires of the rat's nest as a guideline, shown in Figure 18. Many of the traces were routed on the top layer. If signals could not be routed around other components or traces, we needed to route the signal on the bottom layer of the board. This is possible by using a via to transfer the trace to the bottom layer, and another via to bring the

[1] Advanced Circuits - http://www.4pcb.com/

trace back to the top layer. This allows us to route traces over and under other traces and components when routing around them is not possible.

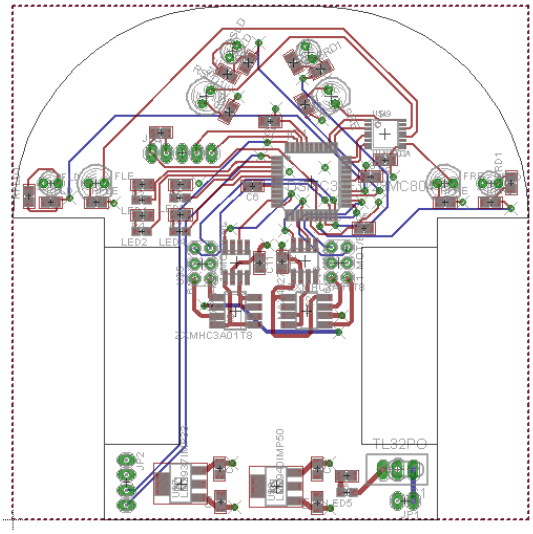


Figure 19. PCB with signal traces routed.

Figure 19 illustrates the PCB once all signal traces have been routed. Blue traces are signals that have been routed on the bottom layer of the board, and red traces are signals that are routed on the top surface of the board. Green via pads are shown where a trace may cross over from one surface of the board to the other. At this point, the board is nearly complete with only power and ground connections to be considered.

In order to minimize the number of power and ground traces that need to be routed around or through other traces and components, we utilized power and ground planes in this PCB design. This mouse design uses 3.3V, 5V, and 7.4V levels, with many of the components using the 5V source. For this reason, we chose to implement a four layer design, with two inner power planes and two outer ground planes. An entire inner layer was dedicated to the 5V power

plane, and the second inner layer was used for the 3.3V and 7.4V power planes. The bottom and top outer surfaces have a ground plane anywhere a trace or solder pad does not exist. The use of power and ground layers makes it extremely easy to provide power and ground connections to all components. For through-hole components, the solder pad that the component is soldered to is simply connected to the necessary ground or power plane. For surface mount components, the ground pin of each component is connected to ground by having that solder pad part of the ground plane. Voltage can then be supplied to SMT components with the use of vias, where a trace can be routed from the solder pad of a voltage pin to a via which is directly connected to the appropriate voltage layer. Figure 20 shows the top layer of the final board design, which includes the top and bottom layer signal traces, as well as the top layer ground plane. Not shown are the inner power layers and bottom ground layer.

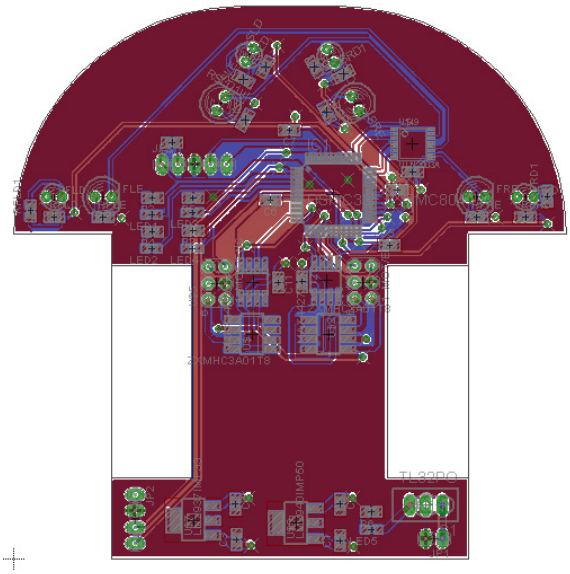


Figure 20: Final PCB design with power and ground planes.

Once the PCB design is completed using the EAGLE software, the necessary files were provided to Advanced Circuits for final design rule checks (DRC) and design for manufacturability checks (DFM). These checks test for errors and make sure that the board can be fabricated using their equipment. After passing these checks, the board can then be

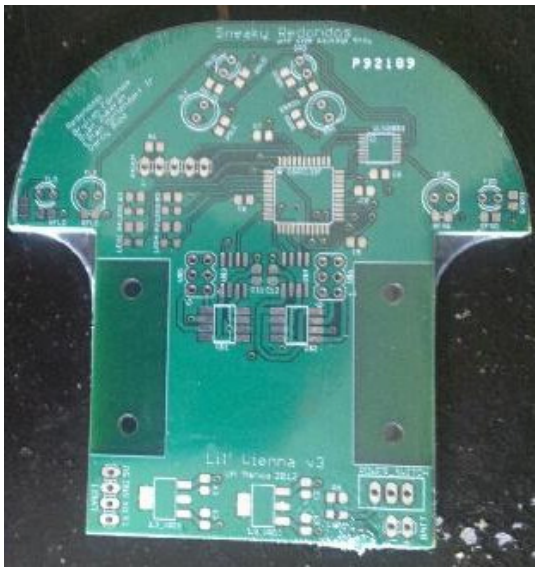


Figure 21: Fabricated board from EAGLE design.

manufactured. Figure 21 shows the fabricated board, a direct product from the CAD process. Once the fabricated board is received, we simply solder the components according to the schematic and board designs, and assemble the mouse for final implementation and programming.

Software

The software portion of the project integrates all hardware components of the mouse into one. It is programmed in embedded C using Microchip's software for PIC microcontrollers, MPLAB X. The current Micromouse software consists of six main sections: Oscillator configuration, UART configuration, sensor controls, motor controls, tracking, and the maze solving algorithm. Without an external clock, the oscillator must be configured to operate at a its maximum speed of 40 MIPS. The UART is used as a wireless communication interface between the computer and the mouse. The sensor controls are responsible for retrieving and processing the sensor readings. The motor controls handle the basic movement of the mouse such as going forward or turning, integrating QEI and PWM modules of the microcontroller. The tracking portion integrates the sensor and motor controls, specifically altering individual left and right motor speeds to ensure the mouse stays in the center of a cell. Finally, the maze solving algorithm takes care of the necessary movements needed to reach the destination goal.

Oscillator Configuration

In any digital device, there exists an oscillator (clock) that controls the timing of events. The clock is a signal source which runs at a particular high frequency. Specifically, the dsPIC33F is equipped with a 7.37 MHz Internal Fast RC. In the event where higher speeds are required, either an external clock can be used or the on-chip Phase-locked-loop (PLL) can be configured. Since no external clock was used in our design, the use of PLL was necessary to obtain the microcontroller's maximum operation of 40 MIPS.

To achieve this speed, the internal Fast RC is used and the input and output of the PLL's Voltage Controlled Oscillator (VCO) is modified. Figure 22 below shows a classic PLL configuration. The phase detector compares two input frequencies and generates an output that is a measure of the phase difference [2]. If FIN does not equal FVCO, the phase-error signal causes the VCO frequency to deviate from FIN. If conditions are right, the VCO will quickly "lock" to FIN to maintain a fixed relationship with the input signal and generate a replica of the input frequency.

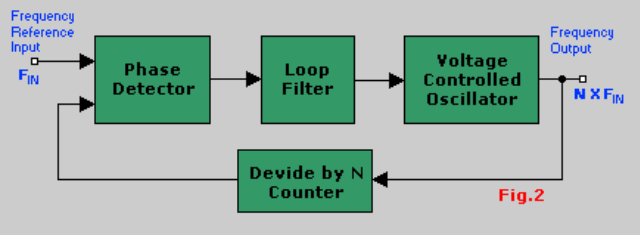


Figure 22: Classic PLL Configuration (Roon).

By controlling VCO multipliers (N in the figure), a multiple of the input frequency FIN can be generated. However, VCO multipliers can not be chosen arbitrarily. According to the dsPIC datasheet, there are limitations to the inputs and outputs of the PLL's VCO to ensure "locking" to FIN. These limitations are: (1) VCO inputs must be in the range of 0.8MHz to 8 MHz and (2) VCO outputs must be in the range of 100 MHz - 200 MHz.

To start with configuration, the system clock frequency (FOSC) must be set at 80 MHz, which is two times the instruction cycle (FCY). In other words, two oscillations per clock cycle (rising edge and falling edge). By using Equation 9-2: *Fosc Calculation* provided by the dsPIC33f family reference manual, the VCO multiplier (M) needed for an operating speed of 40 MIPS can be calculated.

$$Fosc = FIN \times (\frac{M}{N1*N2})$$

Equation 5: Equation used to find required VCO multiplier.

From the equation above, FIN is the internal FRC of the microcontroller, N1 is the prescale factor and N2 is the postscale factor. The first step is to configure the VCO input. By choosing the lowest prescale value of N1 = 2, the VCO input of 3.685 MHz (FIN scaled by a factor of 2) is achieved and falls within the limitations as specified by the datasheet. The second step is to calculate the VCO multiplier. By choosing a postscale of N2 = 2 (lowest postcale value), a multiplier of 43 is achieved, using the equation

$$M = (2 \times 2 \times 80MHz)/(7.37\ MHz) = 43$$

Finally, once the multiplier is obtained, the VCO output must be verified to be within 100 MHz - 200 MHz range. This can be calculated using the equation

$$VCO(out) = VCO(in) \times M = 3.685\ MHz \times 43 = 158\ MHz$$

The code which implements this configuration is shown on Figure X below.

```
// Configure Oscillator to operate the device at 40 MHz
// Fosc = Fin * M/(N1 * N2), Fcy = Fosc/2
// Fosc = 7.37M * 40(2 * 2) = 80MHz (7.37 MHz Internal Fast RC)
PLLFBD = 41;                    // M = 43
CLKDIVbits.PLLPOST = 0;         // N1 = 2
CLKDIVbits.PLLPRE = 0;          // N2 = 2
OSCTUN = 0;                     // Tune FRC oscillator, if FRC is used
RCONbits.SWDTEN = 0;            // Disable Watch Dog Timer
while(OSCCONbits.LOCK != 1) {}; // Wait for PLL to lock
```

Figure 23: Oscillator configuration to operate at 40 MIPS.

UART Configuration

The function of the UART is to enable sequential data transmission from the dsPIC onto the computer's serial port . To ensure proper operation, the UART module of the dsPIC must be configured and must match the settings of the receiving computer's serial port. The software configuration consists of three steps: baud rate, port mapping, and transmission/receiving settings. These three steps is described below.

Baud Rates

Any circuits that needs to use a lower frequency needs to use a sub-multiple of the dsPIC's master clock frequency. Since the microcontroller operates in multiples of 1 MHz and do not divide down to give exact baud rate, the nearest baud rate approximation must be selected. It is critical to select a baud rate with minimum error to ensure the quality of communication is not affected during operation [3].

The UART itself needs a clock signal with a frequency of 16 times the baud rate. If a clock is needed to drive a UART at 9600 baud, it must be divided down to a frequency of 16 times 9600, which is 153,600 Hz. With the dsPIC33FJ128MC804 operating at a maximum speed of 40 MHz with the phase-locked loop (PLL) enabled, the dividing factor is therefore 40,000,000/153, 600 Hz = 260.4167. This is not an integer value, but the nearest value would be 260. Therefore the actual baud rate is then 40,000,000/16/260 = 9615.38 baud, which is a ratio of 1.00016 (0.16%) from the exact value. Overall, lower baud rates mean higher factors by which the master clock cycle is divided. Table 1 below shows the increase in baud rate mismatch as the baud rates are increased.

Baud Rates	Actual Baud Rate	Error (%)
9600	9615	0.16
19200	19231	0.16
115200	113636	1.36
230400	227272	1.37

Table 1. Baud Rate Errors Based on 40 MHz.

For our implementation, the baud rate baud rates of 9600 and 19200 could be chosen to minimize transmission error. Since the XBee module comes pre-configured with a baud rate setting of 9600, a baud rate setting of 9600 was selected. Equation 6 below was used to calculate the value needed to load to the UART register for a specific baud rate of 9600 for a clock operating at 40 MHz.

$$BRGVAL = ((\frac{FCY}{Baudrate})/16) - 1 = 259$$

Equation 6: Calculates value needed for the UART register.

If higher baud rates of the XBee modules are required, see the section in *Changing Baud Rates* for a step by step procedure. Note that BRGVAL must be recalculated to incorporate the change in baud rate.

Port Mapping

The dsPIC33FJ128MC804 comes with a peripheral pin select feature which enables peripheral set selection and placement on a wide range of I/O pins. This increases the pinout options available on a particular device, allowing programmers to better tailor the microcontroller to fit application needs. This feature enables us to set the UART receive pin (RX) and transmit pin (TX) to any of the RPx pins available, providing flexibility in the board design. Figure X below, shows the register configurations for map the UART receive pin (RX) to RP8, and the UART transmit pin (TX) to RP9 of the microcontroller.

```
// I/O Port mapping for UART Modules

RPINR18bits.U1RXR = 8;          // UART Receive tied to RP8[Pin44] (U1RXR <4:0>) p.182 Datasheet
RPOR4bits.RP9R = 3;             // UART Transmit tied to RP9[Pin1] (U1TX <4:0>) p.167 Datasheet
```

Figure 24: I/O Port Mapping of UART Modules.

Serial Port Options

The next step is to configure the UART module transmission settings to match serial port settings of the computer. The BAUD rate register value is calculated in the header file using the equation provided in Figure 25.

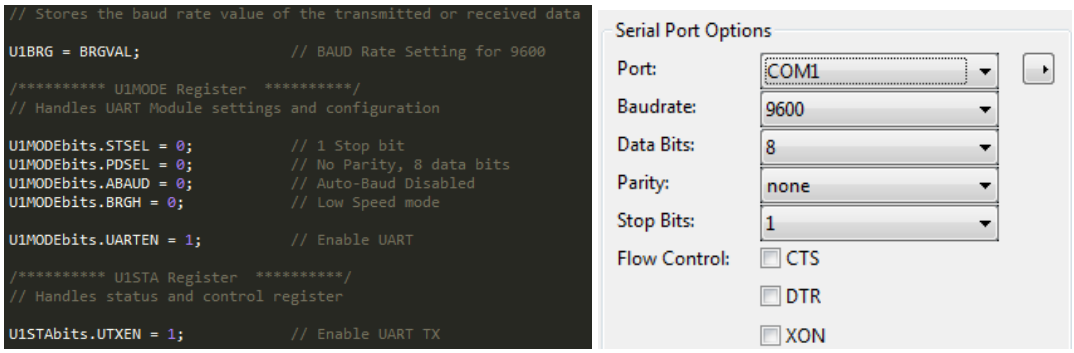
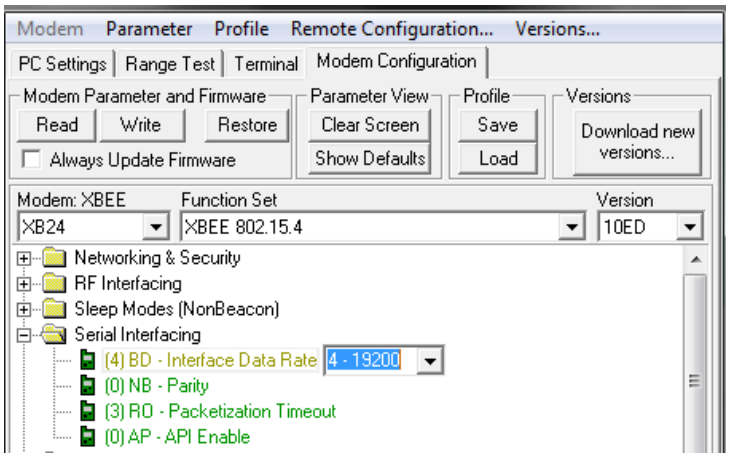


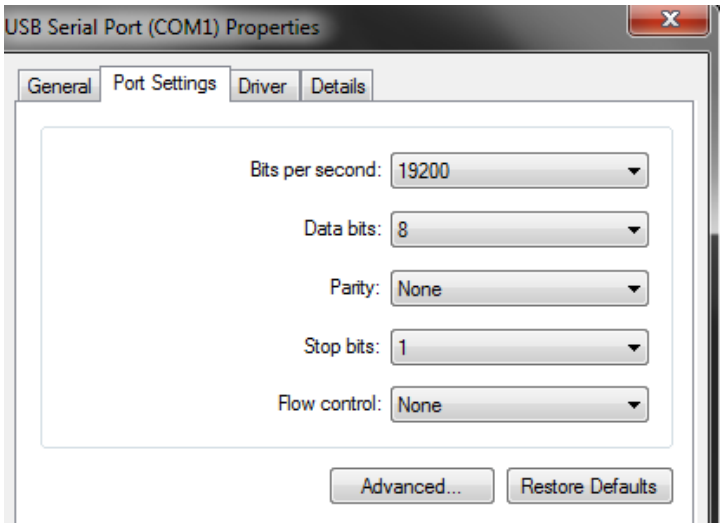
Figure 25: UART Module Transmission/Receive Settings.

Changing Baud Rate Settings

The list of steps below describes the process of changing the baud rate of the XBee.

1. Download and Install X-CTU
2. Plug in Xbee USB Explorer, and Mount XBee Module (In Correct Direction)
3. Launch X-CTU -> Modem Configuration -> Read
4. Change BaudRate to 19200
   - 

5. Under Device Manager -> Ports (COM LPT) -> USB Serial Port (COMX)
   - Change Bits per second

- 

6. Go Under PC Settings, and Change BaudRate to 19200 -> Test/Query
    - Make sure it properly reads Xbee
    - Otherwise, try disconnecting USB explorer or rebooting

Sensor Controls

The function of the sensor control is to handle the data received from sensor readings. Since the values received from the phototransistors are analog, this result must be converted to a digital signal to be processed by the microcontroller. Utilizing the Analog-To-Digital (ADC) control register of the dsPIC33FJ128MC804 microcontroller can do this. The ADC control register uses a 16-bit register for handling conversions and stores the value as a 10-bit integer value. The digital to analog conversion can be calculated by using the equation below. The numerator is divided by 1023 because the value is stored as a 10-bit integer. The reference voltage is the maximum voltage that can be passed to the ADC pin on the PIC.

*Av = (Vref * Dv)/(1023)*

Av = Analog Voltage

Dv = Digital Voltage

Vref = Reference voltage (Configured in PIC as 3.3V using voltage divider circuitry)

Equation 7

The dsPIC33FJ128MCf804 ADC control register port is capable of sampling and converting manually or automatically, and either sequentially or simultaneously based on the control register configuration. However, on a previous mouse design that implemented automatic sampling and conversion, we found it difficult to obtain reliable sensor readings. This was due to ambient light being converted. In order to remedy this issue in software, the phototransistors are first sampled with the emitters off, and then sampled once more with the emitters on. The difference between the two conversions is then taken to eliminate the noise. These values are then passed to the exponential equations generated from the sensor calibration. For details regarding ADC register configuration, please see the section *ADC Configuration: Manual Sampling and Conversion* below.

ADC Configuration: Manual Sampling and Conversion

1. Select 10-bit or 12-bit mode (ADxCON1<10>).
    i.
```
AD1CON1bits.AD12B = 0;       // 10-bit 4-Channel ADC Operation Mode
```
2. Select the voltage reference source to match the expected range on analog inputs (ADxCON2<15:13>).
    i.
```
AD1CON2bits.VCFG = 0;        // VREFH=Avdd, VREFL=Avss (VCFG <2:0>
```
    ii. A VREF of 3.3V was used for our design. (Maximum tolerable pin input voltage)
3. Select the analog conversion clock to match the desired data rate with processor clock (ADxCON3<7:0>).
    i. Since our device is configured to operate at 40 MIPS, the ADC module module is configured to sample at a 1.1 Msps throughput rate with 10-bit resolution. By setting the ADC conversion clock (TAD) to 75ns, the equation $TAD = TCY(ADCS + 1)$ provided by Section 16 of the datasheet can be used to calculate the ADCS value needed for this operation.
    ii.
```
AD1CON3bits.ADRC = 0;            // Use Instruction Clock Cycle (ADRC <0>)
AD1CON3bits.ADCS = 2;            // Set ADCS Control bits (ADCS <7:0>)
```
4. Select the port pins as analog inputs (ADxPCFGH<15:0> and ADxPCFGL<15:0>).
    i.
```
// AD1PCFGH/AD1PCFGL: Port Configuration Register
AD1PCFGL = 0xFFFF;          // Clear all Ports
AD1PCFGL = 0xFF3C;          // AN0,AN1,AN6,AN7 => Analog Input
                            // Rest Digital
```
    ii. The configuration above selects AN0, AN1, AN6, AN7 as analog inputs.
5. Select Manual or Auto Sampling.
    i.
```
AD1CON1bits.ASAM = 0;        // Automatic sampling disabled, controlled by software
```
    ii. Manual conversion is implemented.
6. Select how the conversion results are stored in the buffer (ADxCON1<9:8>).
    i.
```
AD1CON1bits.FORM = 0;        // Unsigned Integer Output (FORM <1:0>)
```

7. Finally, since manual conversion is implemented, a series of steps must be executed in order to sample and convert an emitter.
    i. Enable ADC Module
        a. `AD1CON1bits.ADON = 1;`
    ii. Set which Analog port to sample
        a. `AD1CHS0bits.CH0SA = c;`
    iii. Start sampling
        a. `AD1CON1bits.SAMP = 1;`
    iv. Delay for 10 us. (Efficient sampling time as specified by the datasheet)
        a. `__delay_us(10);`
    v. Stop Sampling (Automatically starts conversion)
        a. `AD1CON1bits.SAMP = 0;`
    vi. Wait for conversion to finish.
        a. `while(!AD1CON1bits.DONE);`
    vii. Read the result form the ADC Buffer.
        a. `return ADCBUF0;`

From here, separate functions can be created for sampling and converting different emitters.

Motor Controls

The motor control is responsible for movements of the mouse. In our previous design which utilized stepper motors, driving commands were simply done by using timer delays and interrupts to control pulses for each step. Speed could be increase with a shorter time delay. However, driving DC motors are far more complex. A PWM pulse train must be used to control the speed of our motor. Quadrature encoder interface (QEI) provides a motor rotation count which combined with timer interrupt values are interpreted and converted into position, speed,

and acceleration values.

DC Motor Control with PWM

To control the speed of our motor, pulse width modulation (PWM) is used to control the delivered power. Pulse frequency remains constant and the percentage of on time is called the duty cycle. This creates an effective "average" voltage seen by the motor. Suppose the duty cycle is 50% (half of the period loaded onto PWM timer register), therefore the motor is powered 50% of the time and an average of half the battery voltage is seen by the motor. The lower the PWM duty cycle, the slower the motor will turn. For a 100% duty cycle, the pulse appears as DC voltage and maximum power is delivered to the motor.

For the PWM frequency, in order to avoid observable variations in the motor speed during a PWM cycle, a desirable range from 10kHz to 40kHz must be chosen. If the frequency is too high, the H-bridge transistors will spend a significant amount of time switching between saturation to off and vice-versa. When these transistors switch from saturation to off, voltage is drained to ground, causing more power to be consumed. If the frequency is too low, undesirable variations in the motor speed will be experienced. For our design, a PWM frequency of 39.1 kHz was used which provides a PWM resolution of 11 bits. From the microcontroller datasheet, Equation X below is used for calculating desired frequencies by adjusting the PWM timebase period (PxTPER) of the microcontroller. The timer is configured to operate in "Free Running Mode" where interrupts are continuously generated to clear PWM Time base (PxTMR) when it matches the PWM time base period matches (PxTPER).

$$Equation\ 8:\ PTPER = FCY/(FPWM \times PTMR\ Prescalar) - 1$$

With a little manipulation, equation x becomes:

$$Equation\ 9:\ FPWM\ =\ FCY/(PTPER+1)\times PTMR\ Prescalar)$$

Finally, in typical H-bridge circuit drivers, a PWM signal and a direction port are used to reverse the rotational direction of the DC motors. However, in our design, the Zetex H-bridge drivers operate as full H-bridges. When the motor is moving forward, one of the two PIC PWM outputs gives a square wave while the other is off. To move in the opposite direction, the PWM signal from the PIC is switched to the other input of the H-bridge driver while the other switches off. In order to accomplish this, the PWM output mode must be operated in complementary mode.

DC Motor Feedback with QEI

Each DC motor has an encoder attached whose purpose is to provide the rotational status of the motor. The encoder has two outputs, QEAx and QEBx, which are QEI inputs to the microcontroller. When QEAx leads QEBx, the motor is moving forward and the 16-bit counter POSxCNT increments. When the motor is moving in reverse, QEBx leads QEAx and POSxCNT decrements. The state transition diagram is located in Figure 26. The quadrature decoder interface of the microcontroller can be found in Figure 27. The IE2-512 encoder which is used on the micromouse provides 512 pulses per revolution. To further enhance the resolution of the encoders, a four times configuration is used. For this state, the microcontroller counts each rising and falling edge of the two QEI channels to provide a total of 2048 counts per revolution.

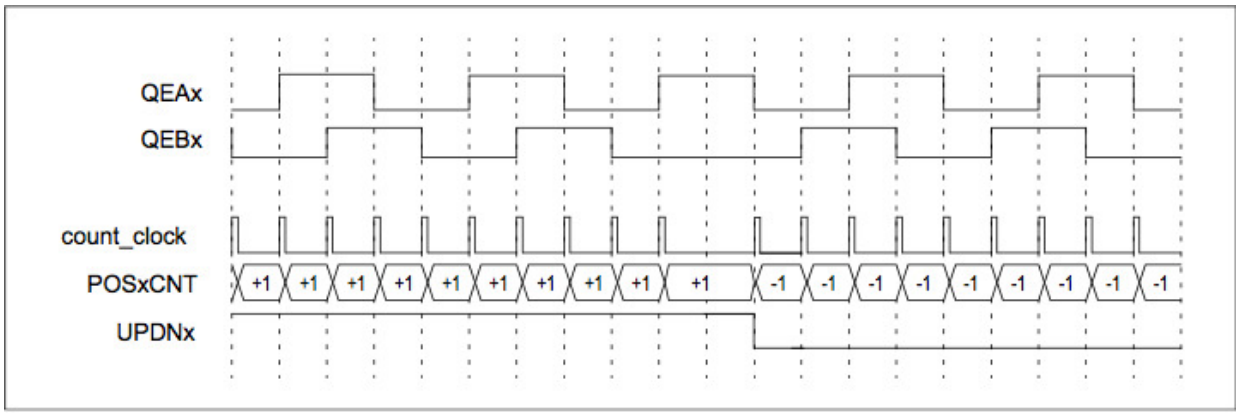


Figure 26: Quadrature Decoder Interface in x4 Mode.

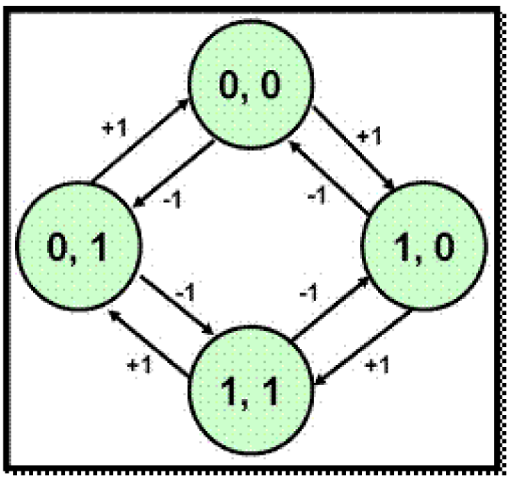


Figure 27: 4X Encoding State Transition Diagram

Knowing how many QEI counts are generated per revolution provides a way obtain approximate distances using the equation $C = \pi * d$. With a wheel diameter of 22mm, the distance traveled per revolution is calculated to be 69.1 mm. The distance traveled per each QEI count can also be calculated using the equation $D(per\ pulse) = \frac{C}{2048 \times Gear\ Ratio} = 0.01mm$, where the gear ratio is 40/12. By knowing the distance traveled per QEI count, a stepper motor like behavior can be achieved.

QEI interrupts can also be incorporated by loading an integer value into the MAXxCNT

register. In the event where a POSxCNT matches MAXxCNT, a QEI interrupt will occur. At this point, various controls can be implemented such as get obtaining sensor values, track, or modify the speeds of the motors. Figure 28 below shows the code used for initializing QEI modules for operation. While MAX1CNT is set to 2048 (one rotation per interrupt), this value can be changed through different sections of the program. For example, for a simple left turn, a POS1CNT is initialized to 0 and MAX1CNT is set to 3400 (approximate counts needed to complete a turn).

```
/********** QEI1CON 16-bit Register (Section 15. p.5) **********/
// Controls QEI operation and provide status flags for state modules

QEI1CONbits.QEISIDL = 0;        // Continue module operation in idle mode
QEI1CONbits.CNTERR = 0;         // Clear any count errors
QEI1CONbits.UPDN = 1;           // Position Counter Direction is positive (+)
QEI1CONbits.SWPAB= 0;           // QEA and QEB not swapped, A leads B
QEI1CONbits.INDX = 1;           // Read only - Index pin state status pin.
QEI1CONbits.POSRES = 0;         // No index pulse reset
QEI1CONbits.QEIM = 7;           // Use 4x Update mode with reset on MAXxCNT match (QEIM <2:0>)
QEI1CONbits.PCDOUT = 0;         // Counter Direction of Status Output (Normal I/O pin operation)

/********** DFLT1CON 16-bit Register (Section 15. p.7) **********/
// Digital filter used to filter spikes in QEI signals

DFLT1CONbits.CEID = 1;          // Interrupts for count errors are disabled
DFLT1CONbits.QEOUT = 0;         // Digital Output filter disabled
DFLT1CONbits.QECK = 0;          // 1:1 Digital Filter Clock Divide (QECK <2:0>)

/********** POS1CNT 16-bit Register  **********/
// Allows reading and writing of the position counter

POS1CNT = 0;                    // Initialize Position Counter => 0

/********** MAX1CNT 16-bit Register  **********/
// Register associated with a comparator for comparing POS1CNT counter

MAX1CNT = 2048;                 // Sets maximum count to 512x4 = 2048 resolution for IE-512 encoder

/********** Interrupt Registers Configuration **********/
IFS3bits.QEI1IF = 0;            // Clears QEI1 Interrupt Flag
IPC14bits.QEI1IP = 7;           // Interrupt Has the highest priority (QEI1IP <2:0>)
IEC3bits.QEI1IE = 1;            // Enables Interrupt for QEI, occurs when POS1CNT == MAX1CNT
```

Figure 28: QEI Initialization.

Tracking

Tracking is required to ensure that the mouse stays within the cell center to avoid crashing, getting lost, and incorrectly mapping walls. This is conducted through utilization of a Proportional-Derivative (PD) controller to improve the overall mouse stability. The proportional term P is the difference between what is considered center of the cell from the current sensor readings. This works efficiently well, since the higher the error, the higher the correction factor. The derivative term D on the other hand is the difference between the current P error and the previous P error over time. With this term, the step response due to the P error can be dampened to reduce the oscillations between corrections [5].

Individual proportional and derivative gains were also incorporated in the controller for finer tuning. K_p is a multiplying factor that adjusts the proportional response expressed in the equation $Pout = Kp * e(t)$. K_d is a multiplying factor that adjusts the derivative term as shown in $Dout = Kd\frac{d}{dt}e(t)$. With these gains K_p, and K_d, the rate at which the mouse corrects can be controlled for faster corrections times. Figure 29 below illustrates a simple effect of adjusting the proportional gain K_p.

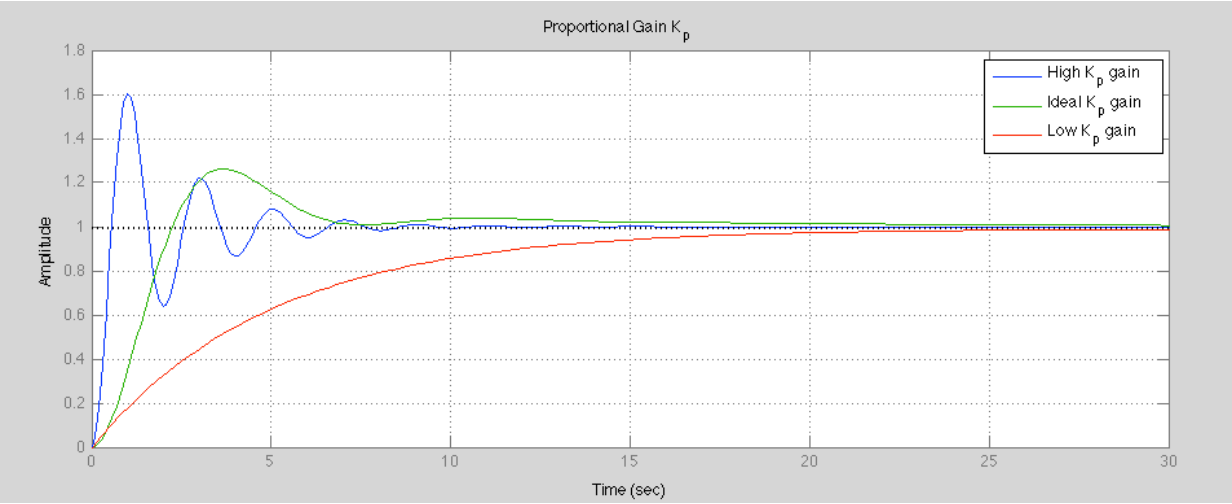


Figure 29: Effects of Proportional Gain Kp

By examining Figure 29 above, the value of the proportional gain will provide different changes in the systems transient response. If the K_p value is too large, the mouse could become unstable, possibly overshooting and crashing into the wall. In contrast, a too low of a K_p gain results in a small change in the output, which means a slower responsive mouse. The ideal value of K_p should be chosen as to minimize the overshoot response and increase the undershoot response. The ideal proportional gain will relocate the mouse near the center of the cell, however the K_p gain alone will not improve the settling time, and the overshoot is still an issue when it comes to a small area such as the cell. Therefore, to better improve the transient response, the derivative gain K_D will be implemented as well. Figure 30 below illustrates the effects of adjusting the derivative gain $K_{D.}$

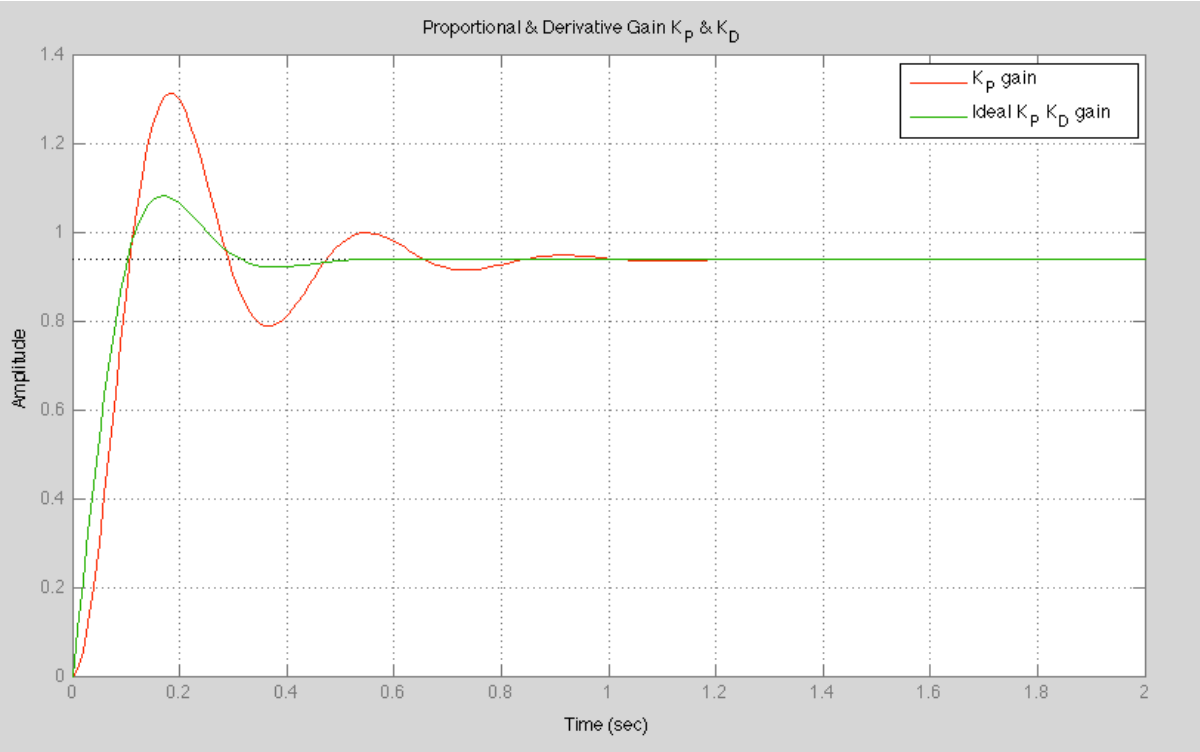


Figure 30: Effects of Proportional & Derivative (PD) Gain

Examining the gain response of Figure 30 above, the response of the proportional gain alone is shown in red, and the response of the proportional and derivative gain is shown in green. Although the proportional gain decreased the transient response, it also provided a substantial amount of overshoot and settling time; two parameters that are not ideal when dealing with a small area. To decrease the overshoot and the settling time, the derivative gain was infused into the overall transfer function of the system. Correctly choosing the best derivative gain value will provide an ideal dampened transient response as shown in Figure 30.

When a quick and accurate transient response is desired, it is usually found by incorporating a feedback control loop which feeds into the transfer function of the plant system

(mouse). Below is a block diagram that shows this feedback control loop.

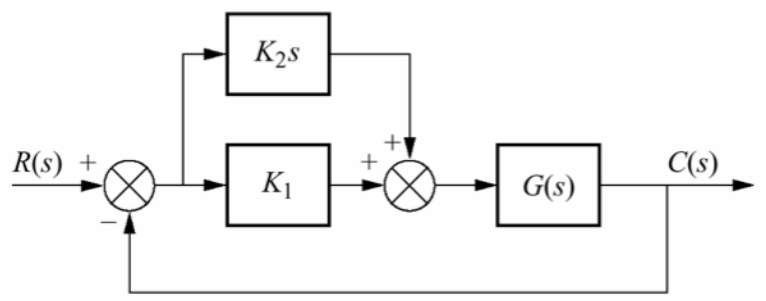


Figure 31: Block Diagram of the Proportional & Derivative (PD) Gain Feedback Control Loop

The feedback control loop above shows the basic process of the PD controller, where K1 is the Proportional gain, K2 is the Derivative gain, and G(s) is the system of the plant, or mouse. The output of the mouse is the current location of the mouse in the cell. The current location is compared to the desired location, which is the input of the system R(s). Should a difference between the two occur, that error value is then feed into the PD controller. With the correct values of the PD controller gains, the output of the PD controller is then the input to the plant system, where it performs the necessary actions to correct it's current location until the error is 0. Determining the values of the P and D gains is a complex process, however it will provide the exact values needed to perform the desired control response. In real world applications, the transfer function and parameters of the plant system is unknown ,and therefore makes calculating the PD gain values nearly impossible. However, it is possible to find the values by choosing arbitrary values for the PD gain and watching how it responds to the values. By understanding the concepts the PD controller and the role it plays in tracking, the values of the

proportional gain and derivative gain can be adjusted independently to achieve the desired response of the mouse.

With the concept of a PD controller examined and understood, the following logic diagram describes the process for implementing the controller in software using the timer module of the dsPIC.

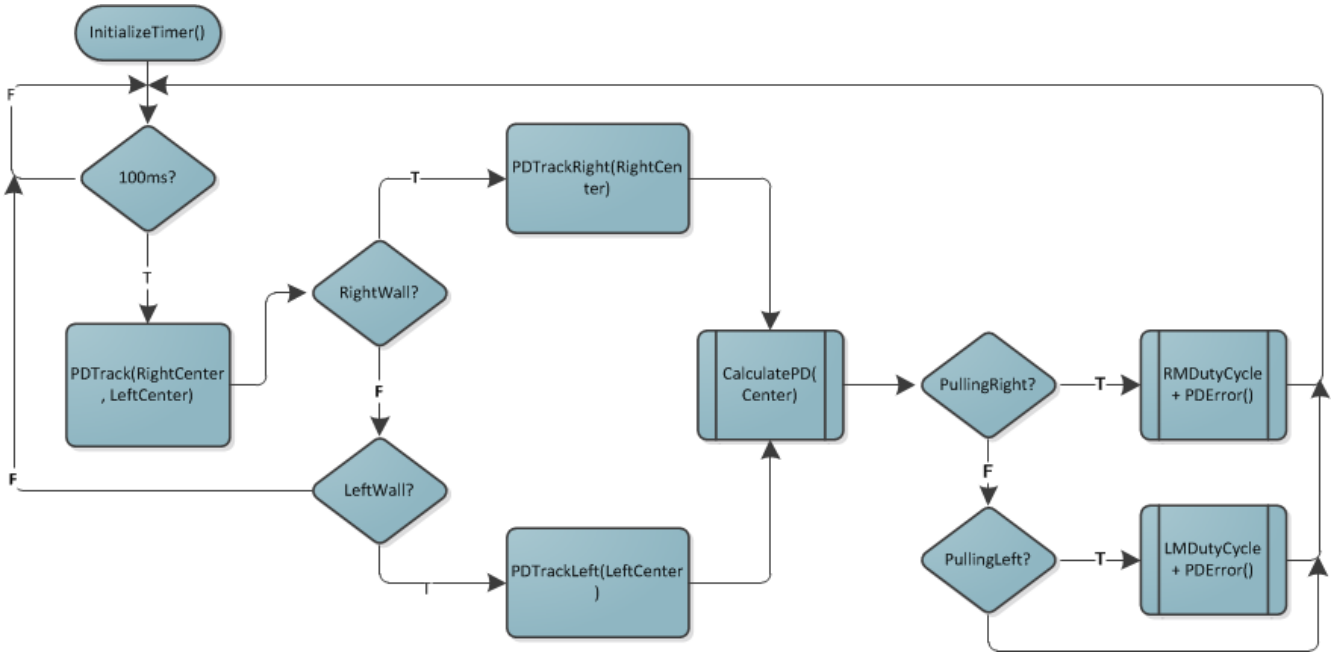


Figure 32: Tracking Logic Diagram.

From the logic diagram above, a timer was configured to interrupt every 100 ms (see *Timer* section for details of configuration). During interrupts, the PD controller is put into effect. If a right wall is detected, then the PD controller will base calculations on right sensor readings. If a right wall is not detected, then the left wall is checked for availability. If a left wall is detected, then the PD controller will base calculation on left sensor readings. Implementing this logic helps in keeping the mouse in the center in the event a series of either a right or a left

wall is not available. However, if no walls are available, the micromouse will drive “blind” since no sensor readings can be taken to refer to as what is considered cell center.

Timer Configuration

The dsPIC33F offers several 16-bit Timer modules. Each Timer module is a 16-bit timer/counter consisting readable/writable registers [11].

1. TMRx: 16-bit Timer Count register.
2. PRx: 16-bit Period register associated with the Timer register.
3. TxCON: 16-bit Timer Control register associated with the timer.

For our implementation, the timer module is configured to operate in Timer mode where the input clock to the timer is derived from the internal clock (FCY), divided by the programmable prescaler [11]. Figure 32 illustrates the code sequence used to configure the Timer3 in 16-bit Timer mode.

```
T3CONbits.TON = 0;      // Disable Timer
T3CONbits.TCS = 0;      // Select internal instruction cycle clock
T3CONbits.TGATE = 0;    // Disable Gated Timer mode
T3CONbits.TCKPS = 0b11; // Select 1:256 Prescaler

TMR3 = 0x00;            // Clear timer register
PR3 =  0x3D09;          // Period = Desired Time/TCY, TCY = 1/(FCY/Prescalar)
                        // Resolution of 100 ms = 1/10_sec

IPC2bits.T3IP = 7;      // Set Timer3 Interrupt Priority Level
IFS0bits.T3IF = 0;      // Clear Timer3 Interrupt Flag
IEC0bits.T3IE = 1;      // Enable Timer3 interrupt
T3CONbits.TON = 1;      // Start Timer
```

Figure 32: Code sequence for Timer Mode configuration.

In the code sequence above, Timer3 module was configured to interrupt every 100 ms to implement PD controller. In order to accomplish this, the value to load to the 16-bit register

must be calculated. This can be easily calculated by making note of a couple parameters: (1) instruction cycle time (TCY), (2) desired time. With an instruction clock rate of 40 MIPS, $TCY = 1/FCY = 25ns$. With a desired interrupt rate of 100 ms and the lowest prescaler value of 1, the value to load the register becomes, $PR3 = (\frac{100ms}{25ns})/1 = 40,000,000$. However, this value cannot be loaded into a 16-bit register. Thus, a prescaler of 256 was used to decrease the rate at which the timer increments. So instead of incrementing TMR3 register every clock cycle, it is now incremented every 256 clock cycles. Applying the prescaler, $PR3 = (100ms)/(25ns/256) = 15625$. This is a value within the bound of a 16-bit integer. The next and final code sequence just enables the interrupts and starts the timer.

Maze Solving Algorithm

The mouse incorporates a modified floodfill algorithm for the maze solving logic. This approach uses distance values to navigate towards the center of the maze. These distance values represent the cell count of how far the mouse is from the destination cell. The shortest path will be followed when the mouse travels through cells with distance values in descending order. A one-dimensional arrays is used to keep track of the distance values and another to map the walls of the maze. To ensure proper operation, north is the current cell plus 16, south is the current cell minus 16, west is the current cell minus one, and east is the current cell plus one. This is because a two dimensional maze is represented sequentially row by row in a one dimensional array.

Prior to the first movement of the mouse, distance values are initialized into the distance array, assuming no walls are present. This provides an initial movement path for the

mouse to take during its exploration of the maze. The robot will attempt to follow this path towards the center. However, after each arrival into a newly visited cell, the walls are updated according to the following bit masked configuration:

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
X	X	X	Visited	West	South	East	North

Table 2: Wall Map Array, Bit Mask Configuration

Bit4 is used to flag the cell if it has been visited and Bit3-Bit0 is used to signify which walls are present within that cell. After the wall update is complete, distance values are calculated by the following rule:

If a cell is not a destination cell, then the distance value will be one plus the minimum value of an open neighbor.

When the current location cell is updated, the surrounding cells may violate the distance value rule. Therefore, the following algorithm resolves this issue:

1. Ensure stack is empty

2. Push the current cell onto the stack

3. While the stack is not empty

a. Pull a cell from the stack

b. If the cell's distance value is one plus the minimum value of an open neighbor do nothing

c. Else, change the distance value to one plus the minimum value of an open neighbor

1. Push all the current cell's open neighbors

onto the stack

The algorithm only updates destination values when the distance rule is violated. This is why modified floodfill was chosen instead of a standard floodfill algorithm. In the standard floodfill approach, the distance values of all cells are recalculated after each movement by the mouse. For the modified approach, computing cycles are conserved.

After distance values are recalculated, the mouse moves to the neighbor with the lowest distance value. To ensure that the walls are updated correctly in relation to the maze, the orientation of the mouse is accounted for following each turn. In summary, the micromouse follows this algorithm after moving into a cell:

1. While current cell is not the destination cell (center),
    a. If cell has not be previously visited, update the wall map
    b. Update the map's distance values as necessary
    c. Decide which open neighbor has the lowest distance value
    d. Move to the neighboring cell with the lowest distance value

Problems and Shortcomings

The mouse can currently travel in a straight direction and properly track. Difficulty with turning, along with time limitations prevented further progress. Design considerations did not take into account the problems with turning from incorporating a four wheel design. The soft compound of the tires also enhanced this problem. A harder compound would allow for limited

tire slippage while turning, however, this may pose a problem with QEI counts and distance traveled. Wheels slipping registers rotation of the motors which do not translate to mouse movement.

Alterations to the motor mounts can help to resolve the problem with turning. While in the rest state, if only the two front wheels are on the ground, turning will be easier. This is because there would be only a single axle of rotation. To achieve this, the rear wheel axles need to be placed slightly higher than the front wheel axles. To be a success, balance should be over the front wheels. During forward acceleration, balance would shift between the front and rear wheels, causing all four wheels to provide traction.

Presently, the large tolerances within the wheels axles cause wobbling of the wheels. This leads to occasional binding of the gearing and inconsistency while attempting to track. To tighten these tolerances, obtaining machined mounts and utilizing bearings within the wheels would be beneficial.

Enhancements to the maze solving algorithm, in particular, flooding back to the start cell and incorporating speed runs would allow for a more competitive mouse. Flooding back to the start cell provides the mouse with the shortest path between the start cell and destination cell and leads to the utilization of speed runs. Speed runs are conducted after the shortest path to the center is discovered. The mouse will return to the start cell and follow a direct path to the center of the maze. This speed run would be completed faster than the first path to the center because exploration would not be involved.

Learning Outcomes

Through the course of the semester, many opportunities for learning were presented. Hardware design is an important component of any electrical design. Through this project, learning about hardware implementation, especially in DC motor design was achieved. Considerations such as power consumption, size, and design features need to be taken into account. Also, the compatibility of the components was important when creating an effective Micromouse. Both the hardware and software teams needed to work together to ensure the proper interface of the two modules. Reading and understanding the datasheets of the electrical components ensure the correct implementation to achieve the objectives of the project.

Printed circuit board design software allowed for four layer PCB to be obtained. This was important in achieving our design objective of building a light and compact micromouse.

The embedded C language used by Microchip was used to program the microcontroller. Header files specific to the type of microcontroller provided functions that were incorporated in the design. Among these functions were the software interrupts used during tracking and driving of the motors. Analog to digital conversion, QEI and PWM were functionalities of the microcontroller which were critical in the operation of the micromouse. Prior knowledge of software programming did not encompass these features. By reading the data sheets and following online tutorials, application was made to the robot.

Many intangible learning benefits were gained through the process of this project. Time

management was an important consideration when planning the intermediate goals to be incrementally accomplished. Setbacks to the plan had to be dealt with because through the course of a project, problems will arise, creating delays to the timeline. Communication and teamwork among members contributed to the success of the project. Complications prevented the achievement of the high level of success originally sought at the beginning of the project.

Conclusion

This Senior Design Project incorporated many aspects of electrical engineering. Practical application to the project was based upon the theoretical results. This revealed the differences between these two types of application. Extensive research was conducted to attempt the building of a successful micromouse. However, setbacks within software and hardware prevented a desirable outcome. Another iteration of the hardware was needed to mitigate the problems experienced with the mouse. Had ample time to conduct this revision was possesed, the team anticipated that the competitive objectives set forth in the beginning of the project would have been achieved.

References

1. Chapter 10: UART Module - Book: Programming dsPIC MCU in C - mikroElektronika. (n.d.).*MikroElektronika - Development tools, Compilers, Books*. Retrieved from http://www.mikroe.com/chapters/view/58/chapter-10-uart-module/#ch10.1

2. Roon, T. (2010, November 12). Phase-Locked Loop Tutorial, PLL. *Phase-Locked Loops*. Retrieved December 12, 2012, from http://www.sentex.ca/~mec1995/gadgets/pll/pll.html

3. The modified flood-fill algorithm. (n.d.). *MicroMouse — design of a working MicroMouse*. Retrieved December 12, 2012, from http://www.micromouseinfo.com/introduction/mfloodfill/

4. "PCB Trace Width Calculator." *The CircuitCalculator.com*. N.p., 31 2006. Web. 01 Dec 2012. <http://circuitcalculator.com/wordpress/2006/01/31/pcb-trace-width-calculator/>.

5. Wescot, T. (2000, October). PID without a PhD. *EE Times-India*. Retrieved from http://igor.chudov.com/manuals/Servo-Tuning/PID-without-a-PhD.pdf.

Datasheets

1. Infrared Emitter SFH 4545

    http://catalog.osram-os.com/media/_en/Graphics/00050048_0.pdf

2. Infrared Phototransistors TEFT 4300

    http://www.vishay.com/doc?81549

3. Faulhaber DC Micromotors 1717T SR006

    http://www.micromo.com/Micromo/DCMicroMotors/1717_SR_DFF.pdf

4. Faulhaber Motor Encoders IE2-512

    http://www.faulhaber.com/uploadpk/EN_IE2-1024_DFF.pdf

5. Darlington Transistor Array ULN2003APWR

    http://www.ti.com/lit/ds/symlink/uln2003a.pdf

6. IC Linear 3.3V Voltage Regulator LM2937IMP-3.3

    http://www.ti.com/lit/ds/symlink/lm2937-2.5.pdf

7. IC Linear 5.0V Voltage Regulator LM2940IMP-5.0

    http://www.ti.com/lit/ds/symlink/lm2940-n.pdf

8. MOSFET H-Bridge Driver ZXMHC3A01T8TA

    http://media.digikey.com/pdf/Data%20Sheets/Zetex%20PDFs/ZXMHC3A01T8.pdf

9. IC MOSFET Driver MAX4427CSA+

    http://datasheets.maximintegrated.com/en/ds/MAX4426-MAX4428.pdf

10. XBee Wireless Kit B004WLHE1G

http://www.digi.com/pdf/ds_xbeemultipointmodules.pdf

11. PIC Microcontroller dsPIC33FJ128MC804

http://ww1.microchip.com/downloads/en/DeviceDoc/70291G.pdf

Source Code

https://github.com/foronda/DCuMouse_2012