2012

Development of a Low-Cost Robotics Platform that Facilitates the Enhancement of Microcomputer Structures and Interfacing Learning Objectives

Justin Ryan Morris
West Virginia University

Development of a Low-Cost Robotics Platform that Facilitates the Enhancement of Microcomputer Structures and Interfacing Learning Objectives

Justin Ryan Morris

Thesis Submitted to the
College of Engineering and Mineral Resources
at West Virginia University
In partial fulfillment of the requirements
for the degree of

Master of Science
in
Electrical Engineering

Powsiri Klinkhachorn, Ph.D., Chair
Roy S. Nutter, Ph.D.
Afzel Noore, Ph.D.

Benjamin M. Statler College of Engineering and Mineral Resources

Morgantown, West Virginia
2012

Keywords: Robotics, Competition, Curriculum, Micromouse, Microprocessors

Abstract

Development of a Low-Cost Robotics Platform that Facilitates the Enhancement of Microcomputer Structures and Interfacing Learning Objectives

Justin R Morris

Robotics has become a common educational tool to teach basic concepts in mathematics, science, engineering, technology, world affairs, and much more. Programs such as For Inspiration and Recognition of Science and Technology (FIRST) robotics are demonstrating that the recipe for student inspiration and learning involves robotics, problem solving, teamwork, and friendly competition. The successes of FIRST robotics programs and results from universities that have integrated robotics platforms into their curriculum provide evidence that infusing robotics platforms and curriculum into engineering departments better their capabilities and increase attractiveness to current and future students. This effort details the design and development of a low-cost robotics platform and seamless set of supporting curriculum. The platform and seamless curriculum set is implemented in the West Virginia University's Lane Department of Computer Science and Electrical Engineering (LCSEE) microcomputer structures and interfacing laboratory, an undergraduate computer engineering course. The results provide detailed information on the robotics platform as well as detailed information on the seamless set of modules that make up the curriculum. The results demonstrate that a subset of students become significantly more motivated and are more willing to work additional hours to improve upon their design as compared to traditional laboratory sessions. These results are significant and demonstrate that robotics and seamless curriculum sets provide a solid platform to introduce computer engineering concepts that inspire and motivate students.

Acknowledgements

Justin McCarty. Justin was the primary software developer of the Micromouse robot software during the spring 2009 semester and helped me get started with this work. A few of the source code modules were reengineered from Justin's work on this effort during a previous semester.

Sam Castillo. Sam provided valuable insight into 2011 and 2012 results from the implementation of the micromouse curriculum. Sam also enhanced the micromouse curriculum and source code during the 2011 and 2012 semesters.

Dr. Powsiri Klinkhachorn. Dr. Klink provided continual oversight during the course of this effort. In addition, Dr. Klink provided the training and support in the design, development, and manufacturing of the eleven Micromouse robots that were used during the 2010 spring semester.

Spring 2010 CPE 313 students. These students were essentially the guinea pigs for trying out a brand new set of curriculum and helped iron out bugs, typos, and errors in lab handouts, questions, and assignments. Their comments, grades, and one-on-one conversations form the basis for the results described herein.

I would like to acknowledge my wife and son for providing continual support throughout the duration of my thesis. I love them very much and thank them for everything that they do.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Robotic platforms have become a common educational tool to teach students basic concepts in mathematics, science, engineering, technology, world affairs, and much more. Over 200,000 students worldwide are expected to participate in the 2012 For Inspiration and Recognition of Science and Technology (FIRST) LEGO Robotics League; a sports-like robotics competition for students ages 9-14 [3]. The program aims to inspire young people's interest and participation in science. At the core of the FIRST LEGO League is a robotics kit manufactured by LEGO which includes a 32-bit ARM7 processor, Bluetooth wireless communications, a series of input-output ports, and a 9 volt battery [3].

In addition to LEGO Robotics, FIRST offers two robotics programs for high school aged students as well as a program for children ages 6-8. To complement the efforts of FIRST and similar programs, higher education institutions have developed supporting robotics curriculum and materials. For example, Carnegie Mellon University (CMU) has developed a Robotics Academy to "Use the motivational effects of robotics to excite students about science and technology" [4]. The Academy develops and maintains a rich set of curriculum and student and educator robotics resources.

Government agencies such as NASA have invested in robotics education by standing up programs such as the NASA Robotics Alliance Project [5]. The Robotics Alliance Project strives to increase American support for the advancement of Robotics Technologies, inspire American high school students to pursue Robotics Engineering degrees, and inspire undergraduate and graduate students to pursue advanced degrees in robotics [5]. The Robotics Alliance Project sponsors FIRST robotics teams along with other educational robotics programs such as Botball, Boosting Engineering Science and Technology (BEST), National Underwater Robotics Challenge (NURC), and VEX robotics [6], [7]. Each of these programs differs in their organization but their underlying goals are generally the same; to teach students fundamental skills in science, technology, engineering, and mathematics.

Youth robotics programs are proving that integrating robotics into science, technology, engineering, and mathematics (STEM) curriculum is an effective and efficient way to inspire and educate students. Robotics programs such as FIRST are even inspiring students to later pursue STEM careers. A study conducted by Brandeis University, Center for Youth and Communities, revealed that students involved in the FIRST Robotics Competitions are twice more likely to major in a science and technology field than students that are not enrolled in the program [1]. Figure 1.1 captures a summary of the results of this study.

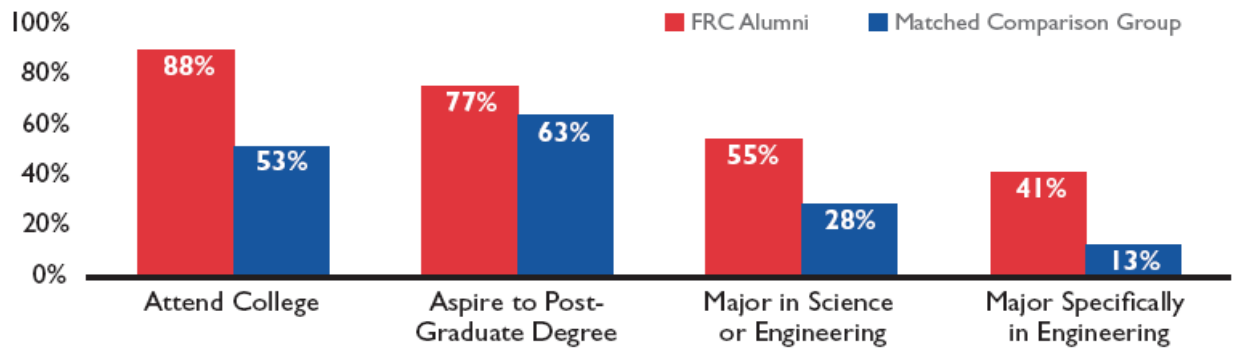


Figure 1.1: Brandeis University FIRST Survey Results **[1]**

In addition to increasing student motivation to pursue technology-based fields, robotics programs are continuing to demonstrate significant growth. Figure 1.2 captures the growth of the FIRST LEGO League (FLL) program. In its initial year, the FIRST Lego League consisted of 200 teams [8]. In 2012, it is projected that 20,500 teams and over 200,000 students across 60 countries will participate [8].

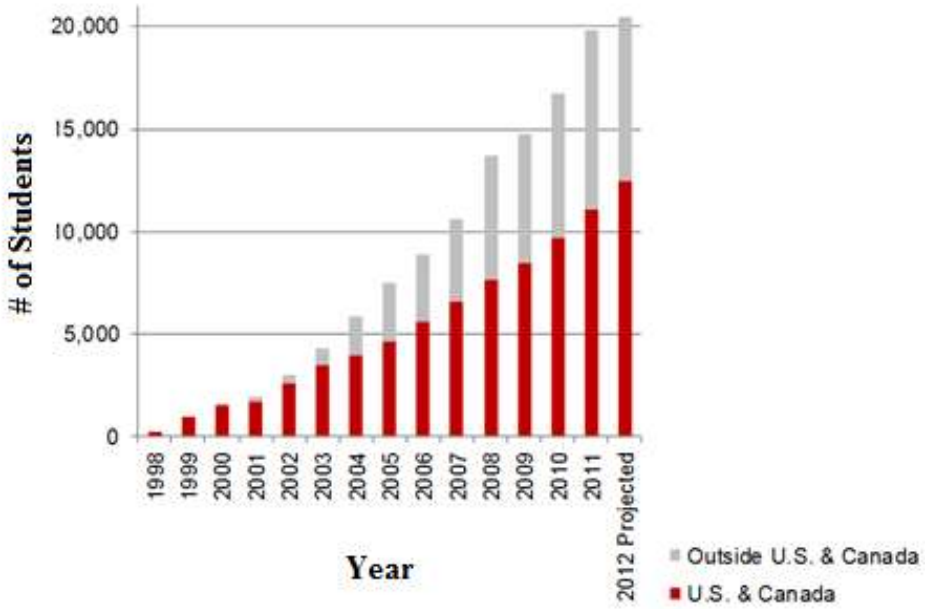


Figure 1.2: FIRST LEGO League Growth **[1]**

Traditional undergraduate engineering laboratory sessions can be described as goal-oriented (i.e. a lab session has a predetermined set of objectives). Once students have completed their predetermined set of objectives, they are finished with the laboratory session. Take for example, a common electrical engineering laboratory project, a clapper. A clapper is an electrical circuit that will turn on and off based on the detection of a loud sound such as the clapping of hands. The clapper project thus has a predefined goal that upon recognizing a clapping sound, a light will turn on and off. Once the basic functionality is achieved by students, no additional work is required. Robotics-based (and competition-based) laboratories provide the inherit capability to challenge students to achieve higher standards and remove the boundaries of traditional laboratory sessions.

Integration of another robotics platform, called the micromouse, with conventional undergraduate engineering curriculum has proven to be successful in motivating students and keeping their interest high [2]. Results from a large-scale integration of the micromouse curriculum at California State University, Fullerton, provided students with hands-on experience and enhanced classroom instruction, student retention, and curriculum. Additionally, many of the skills that students developed through using micromouse curriculum comply with the criteria of evaluation from the Accreditation Board for Engineering and Technology (ABET) [9].

This thesis investigates the benefits of integrating robotics-based curriculum into the West Virginia University Lane Department of Computer Science and Electrical Engineering (LCSEE) computer engineering undergraduate laboratory, microcomputer structures and interfacing. The demonstrated successes of the FIRST robotics programs and California State University curriculum provide a strong case for infusion of robotics curriculum into undergraduate engineering departments to better the their capabilities and attractiveness to current and future students.

A low-cost robotics platform was needed that would be capable of satisfying the learning objectives of the microcomputer and interfacing laboratory. The microcomputer and interfacing laboratory learning objectives are as follows:

1. Draw a detailed architecture diagram of the HCS12 microcontrollers
2. Write a program for HCS12 microcontrollers, i.e. know all addressing modes and full instruction of the HCS12 microcontrollers; and how to write a program using HCS12 assembler and debugging tools
3. Design and use interrupt as an integral part of the microcontroller
4. Design and use programmable timers as an integral part of the microcontroller
5. Design and use input/output ports as an integral part of the microcontroller
6. Interface and develop software to output a wide-range of process control signal (DC, AC, high power, variety of voltage/current compatible levels, I2C, Keyboards, LCD, etc.)
7. Design and use Pulse Width Modulation (PWM) as an integral part of the microcontroller
8. Understand the basic operation and use of Analog-to-Digital Converter (ADC) and Digital-to-Analog converter with microcontroller
9. Understand the basic operation of commonly used sensors/transducers

Due to the successes with the micromouse by California State University described in [2] and due to the fact that the micromouse platform has a set of pre-existing rules administered by the Institute of Electronics and Electrical Engineers (IEEE), the micromouse platform was selected as the primary alternative for a low-cost robotics framework. Additional documentation was available based on the results of LCSEE graduate robotics courses that demonstrated that the micromouse platform could be implemented for a reasonable cost.

1.2 Statement of the Problem

The first phase of this thesis identifies, designs, and develops a low-cost robotics platform. The platform will continue to utilize the Freescale HCS12 processor family and Freescale CodeWarrior Integrated Development Environment (IDE) with Processor Expert. The budget shall not exceed $200 per robotics platform in order to minimize laboratory costs. The robotics platform will be prototyped, tested, and eleven robotic platforms will be developed.

The second phase of this thesis is to develop a rich set of curriculum to accompany the robotics platform. The curriculum will be modular, challenging and introduce new topics not previously introduced in the LCSEE department courses. Most importantly, the curriculum will be seamless from beginning to end to ensure that students understand the relevance of all topic areas covered. The curriculum will be implemented and evaluated to assess the benefit of its infusion in the department. Both student motivation and grades will be assessed and compared with previous semesters in which the robotics platform was not utilized.

1.3 Thesis outline

Chapter 2 includes the review of literature broken into four major subsections: micromouse, mechanical platform, electrical platform, and software platform. The micromouse section details the micromouse, micromouse competition, and includes some standard micromouse designs. The mechanical platform section details the research results on chasses, sensor mounts, and motors; the electrical platform section includes details on the microcontroller, power source, sensors, and motors; and the software platform section presents the common micromouse maze solving algorithms. Chapter 3 includes the details of the design and deployment of the utilized micromouse platform and bill of materials of the as-designed micromouse platform. Chapter 4 provides the results from this research effort including the details from implementation of the curriculum and additional applications for the micromouse platform. Chapter 5 concludes with a summary and recommendations for future work.

Chapter 2

Literature Review

This chapter discusses the robotics platform followed by the discussions on the technical trade-offs of the mechanical, electrical, and software platforms. Each section describes the advantages and disadvantages of potential solutions to develop a low-cost robotics platform to support the undergraduate computer engineering course, microcomputer structures and interfacing.

2.1 Robotics Platform

The micromouse will serve as the robotics platform for this effort. The micromouse is an autonomous robotic mouse whose objective is to solve a 16x16 maze. A maze cell is typically a 180 mm square with 50 mm high walls with passageways between the walls 168 mm wide. The floor of the maze is painted black, and the maze walls are painted white. Figure 2.1 provides an example of a typical micromouse maze whereas the car denotes the starting cell and mouse position and “F” denotes the destination cell.

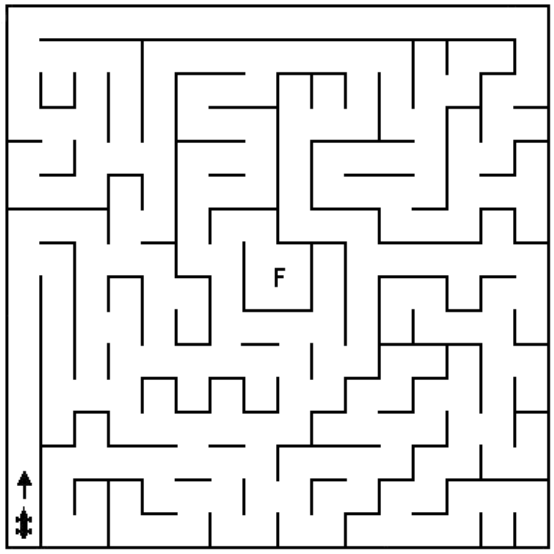


Figure 2.1: Micromouse Maze Example **[10]**

The micromouse is provided with a predetermined starting position and must correctly traverse an unknown maze and determine when it has reached the goal cell. Once the correct route has been identified, the mouse should return to the starting position, and then run the correct route in the shortest possible time. A micromouse robot is generally composed of some combination of motors (DC or stepper) and sensors (proximity or distance), power source, display mechanism, chassis, wheels, and switches. A micromouse robot typically is controlled using a microcontroller which reads sensors, controls micromouse movement, implements error detection and correction algorithms, and implements various searching algorithms. Figure 2.2 provides a few images of some common micromouse designs [11]**.**

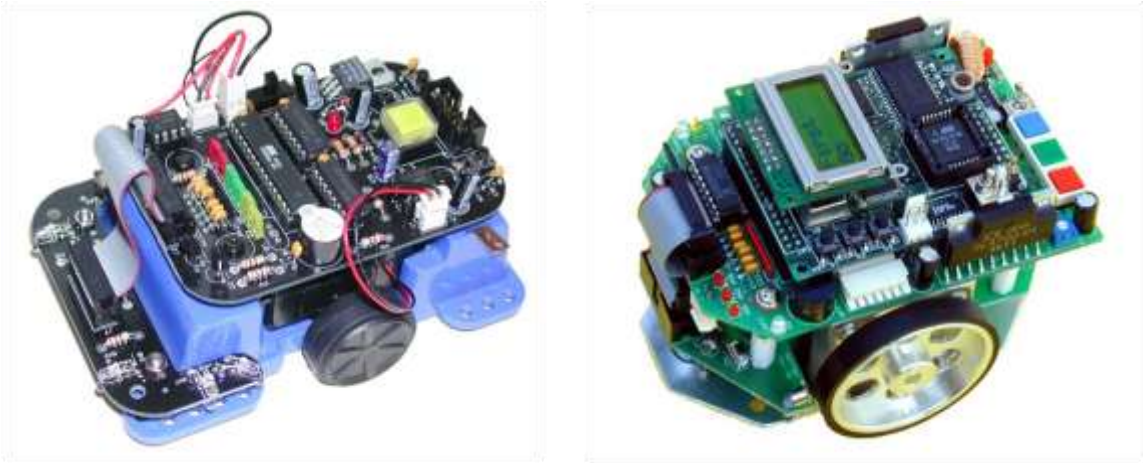


Figure 2.2: Micromouse Designs **[11]**

In 1977 the Institute of Electronics and Electrical Engineers (IEEE) created the micromouse competition concept, and the first competition was held in New York in 1979 [12]. In a matter of a few years, the competition went global, and the first world competition was held in Japan in 1985 [12]. Competitions today are held across the United States and the world. IEEE micromouse competitions are held based on geographical region. Region 2 includes West Virginia, Delaware, District of Columbia, Maryland, Southern New Jersey, Ohio (except Toledo), Pennsylvania, and Northern Virginia. The 2010 Region 2 Micromouse Competition was hosted by Temple University and held on April 17, 2010 [13]. Appendix A includes a complete copy of the 2010 Region 2 Micromouse Competition Rules.

The micromouse competition is a robotics competition where an autonomous robot is constructed with the objective to solve an unknown maze in the shortest possible amount of time. For each run, a micromouse is given ten minutes to access the maze, and the minimum time that the mouse can solve the maze is its recorded time. The start of a micromouse maze is at one of the four corners where the starting square is surrounded by walls on three sides, and the destination unit is the four cells in the center of the maze. The four primary rules of a micromouse competition are as follows [13]:

1. A micromouse shall be self-contained (no remote controls). A micromouse shall not use an energy source employing a combustion process.
2. A micromouse shall not leave any part of its body behind while negotiating the maze.
3. A micromouse shall not jump over, fly over, climb, scratch, cut, burn, mark, damage, or destroy the walls of the maze.
4. A micromouse shall not be larger either in length or in width, than 25 centimeters. The dimensions of a micromouse that changes its geometry during a run shall not be greater than 25 cm x 25 cm. There are no restrictions on the height of a micromouse.

2.2 Mechanical Platform

The mechanical platform consists of the chassis and all the required physical components to hold the micromouse together. There are four overarching design characteristics to consider when designing and building a micromouse robot [14]. These characteristics should be considered during micromouse research and design.

1. Overall size should be small. Minimizing the size of the robot decreases the probability of getting caught on walls and collisions.
2. Minimize the moment of inertia. Minimizing the moment of inertia allows the robot to turn quicker and more accurately.
3. Low center of mass. Lowering the center of mass increases the robot's stability and decreases the likelihood of toppling. The chassis should be built with lightweight and simple, replaceable components.
4. Wheel placement is important. Wheel placement affects the robot's ability to traverse the maze.

Two common micromouse wheel layouts were recognized [15]. The first layout is often referred to as the wheelchair mouse. The wheelchair mouse has a castor on one or both ends of the chassis and commonly utilizes two motors. The second layout utilizes four motors such that one can drive all four wheels. The wheelchair design must have a good weight distribution to prevent the robot from toppling and is less efficient during acceleration as the weight distribution changes. The four wheel mouse will handle the weight distribution on the wheels during acceleration better than the wheelchair mouse; however, additional wheels increase the total weight of the robot. The four wheel mouse is also more difficult to steer and control due to the additional motors. The additional motors require more programming to control and synchronize the wheel movements compared to the wheelchair mouse. Figure 2.3 provides a block diagram of the two common micromouse layouts.

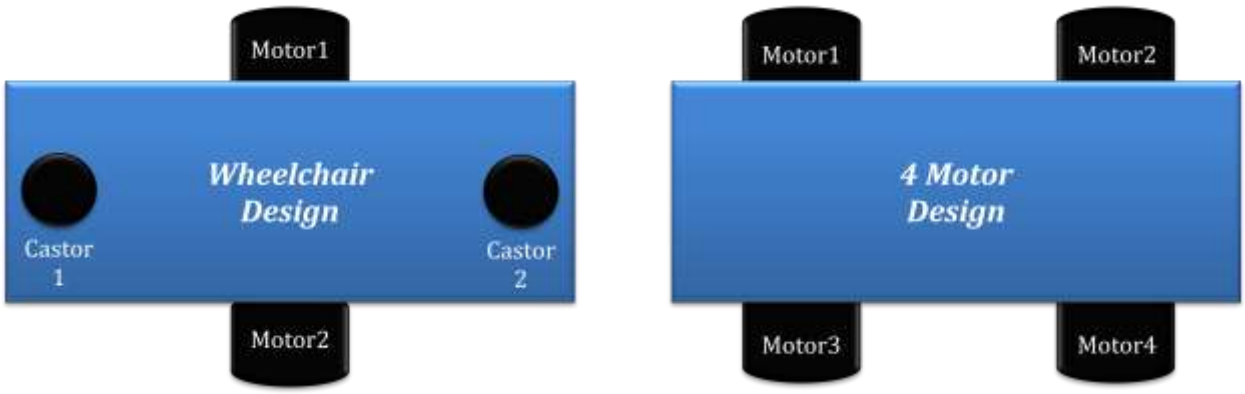


Figure 2.3: Wheelchair versus 4-Motor Design **[16]**

Wheel diameter and weight need to be considered for an efficient micromouse platform. Since speed and more importantly, acceleration, are essential to designing a quality micromouse, the following equations are introduced to understand the relationships between wheel mass (m), acceleration (a), wheel radius (r), and torque (τ).

$$F = ma = \frac{\tau}{r}, \text{ solving for acceleration yields: } a = \frac{\tau}{rm}$$

To increase the robot's acceleration, a motor that has high torque (τ) and minimal size is most favorable. Robots with wheels that are too large are also more likely to tip over. As a result, the micromouse design needs to minimize the wheel size while ensuring that the mouse chassis does not drag.

2.3 Electrical Platform

The electrical platform can be broken into three basic parts: computing platform, inputs and outputs as shown in Figure 2.4. The computing platform houses the software application with the algorithm required for the robot to traverse the maze and provides interfaces for the micromouse input and output components. The input components are the sensors which provide the robot with information regarding its current state and orientation within the maze. The output components include the motors and display devices for debugging and status information.

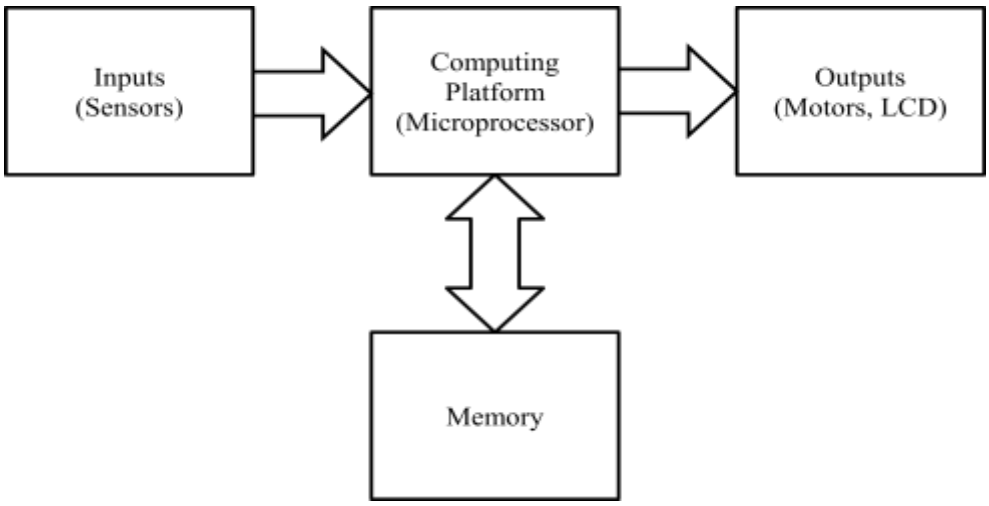


Figure 2.4: Electrical Platform Block Diagram

2.3.1 Computing Platform

The computing platform is quickly reduced to a microcontroller application due to the mouse's physical size constraints. The microcontroller is generically responsible for controlling the navigation of the mouse, tracking maze information, and optimizing a path back to the starting maze cell. The microcontroller must be low power and provide sufficient ports to interface with all external peripherals. It is important for the microcontroller to be low cost due to limited project budgets. In addition, the computing platform options were limited for this project as it was desired to utilize same microcontroller family of processors that had been utilized in previous microcomputer structures and interfacing laboratories. The computing platform was to utilize the HCS12 family of Freescale microcontrollers.

Two single board computer project boards, shown in Figure 2.5 that utilized the HCS12 family of Freescale microcontrollers were considered in the design of the micromouse, the TinyDragon project board (left) [17] and the Dragonflybot project board (right) [18]. Comparing the two single board computers, the TinyDragon board provided more memory, more input-output lines, smaller in size, and was $19 cheaper than the Dragonflybot board. The Dragonflybot board provided more on-board switches and LEDs and also had on-board connectors for H-bridges to drive two stepper motors or two DC motors. Table 2.1 provides an in-depth comparison of the two single board computers.

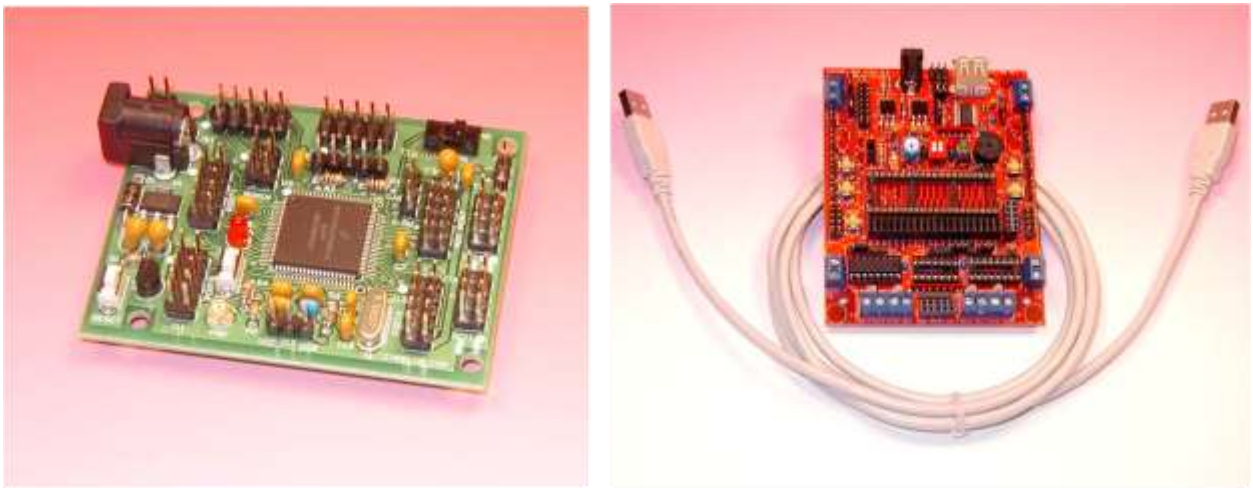


Figure 2.5: Microcontroller Project Boards **[17], [18]**

Table 2.1: Microcontroller Project Board Comparison **[17]**, **[18]**

	TinyDragon	**Dragonflybot**
Memory	256k flash, 12k RAM, and 4K EEPROM	32k flash, 2k RAM, and 2K EEPROM
Features	1 PB switch, 1 LED indicator	4 PB switches, 8 LEDs
Input-Output	51 accessible I/O pins; 5 Ports	31 IO lines
On-chip peripherals	3 SPIs, 2 SCIs, 2 CANs, I2C interface, 8 16-bit timers, PWMs, 8-channel 10-bit ADC	1 SPI, 1 SCI, CAN port, I2C, PWMs, 8-channel 10-bit ADC
Speed	8 MHz, 4 MHz default bus speed, 25 MHz PLL	8 MHz, 24 MHz default bus speed, 25 MHz PLL
Size	1.9” x 2.5”	3.3" X 4.3"
Power Options	AC/DC Adapter or 2-position header for external battery	USB or AC/DC adapter
Noteworthy Features	Increased memory space for micromouse algorithm	Dual H-bridges to control two motors (stepper); separate voltage supplies for motors and microcontroller
Cost	$59	$78

2.3.2 Sensors

The sensors on the micromouse are utilized to detect movement errors as well as provide information needed to solve the maze. Considering the desired functionality of the micromouse to successfully traverse the unknown maze, there are a few factors that must be considered when selecting the appropriate sensors. These factors include repeatability, accuracy, and sampling time [14]. Repeatability is "the ability of a component to repeat producing the same result under the same conditions", and highly repeatable sensors produce samples that have small variance [14]. As a result of the importance of getting consistent sensor results as a micromouse traverses a maze to avoid collisions and incorrect turns, one can conclude that selecting a highly repeatable sensor will help produce a more robust micromouse design.

The final sensor characteristic that should be considered is the sampling time, the amount of time it takes for the sensor to capture an object (wall in our case) to the time that the software is aware that the sampling has occurred [14]. Micromouse applications require short sampling times as the robot is constantly in motion. If sample times are too great, the robot may no longer be in the same location and the readings will be received too late to be useful.

In Tondra De's, The Inception of Chedda [19], three primary sensor types are compared for effectiveness in micromouse applications: infrared, ultrasonic, and touch. De concludes that touch sensors may be useful to guide the robot but should not be used for primary sensing; sonar sensors have their advantages but are subject to interference; and infrared sensors provide the best alternative as they are not subject to electromagnetic interference and more often cheaper than sonar sensors [19]. Taking this work into consideration and examining other primary references [14], [15], [20], sensor type research is limited to infrared proximity sensors.

Infrared sensors operate by sending a pulse of infrared light (light emitter) and the light travels outward in the field of view and either hits an object or continues. If the light does not reflect off an object, the sensor does not return a reading; however, if the light reflects off an object, it returns to the linear sensor array (detected by what is called a light detector) [21]. Now having three information points (reflection point, emitter location, and detector location), the sensor can calculate the distance to the object by knowing the reflected angle the light returned [21]. Figure 2.6 provides an illustration depicting a typical infrared sensor reading.

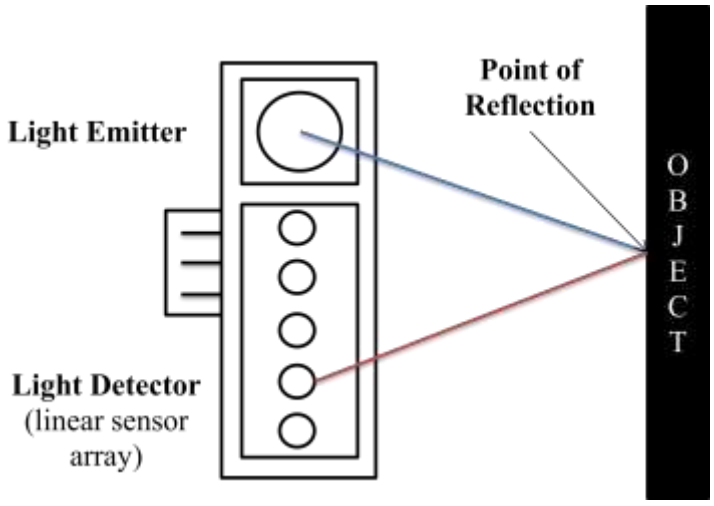


Figure 2.6: Infrared Sensor Reading **[21]**

Due to the nature of infrared sensors, infrared sensors come in many varieties as there are many different distance ranges that would need to be detected for different applications. Figure 2.7 provides two examples of infrared sensors provided by Sharp Electronics [21] . The black-shaded area is the distance range that the sensor cannot detect, and the gray-shaded region is the sensor's detectable distance range. For example, the Sharp GP2D120 can detect distances in the range of 4 to 30 cm [22].

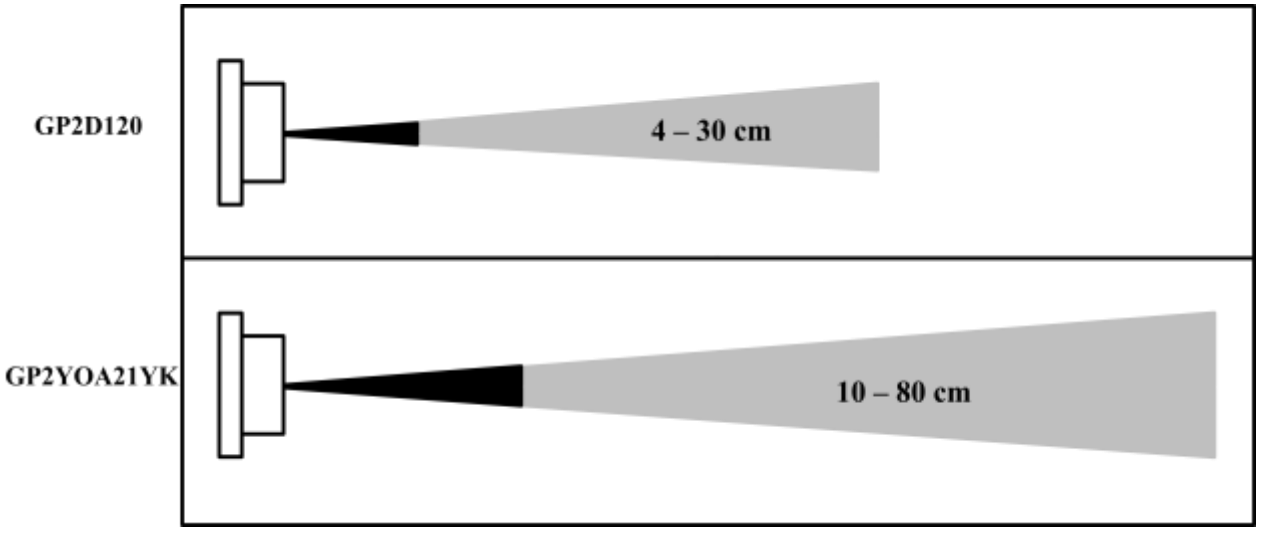


Figure 2.7: Sharp Electronics Example Sensor Ranges (in cm) **[21]**

Figure 2.8 captures the analog output voltage versus distance from the reflected surface (in cm) for the Sharp GP2D120 infrared sensor [22]. The detectable range of the GP2D120 sensor is from 4 to 30 cm. Figure 2.8 illustrates that in this range, the curve is not completely linear. Additionally it is important to recognize that this curve will be slightly different for each detector utilized [21]. As a result, it is recommended to linearize the output using either a lookup table or a parameterized function in order to calibrate each detector.

In addition, the plot illustrates that voltage readings less than 4 cm from the reflective object drop off significantly. Readings less than 4 cm could be misinterpreted as a further distance. As a result, it is important to design a micromouse platform that prevents an infrared sensor from receiving a reading that is not within the sensor's detectable range.

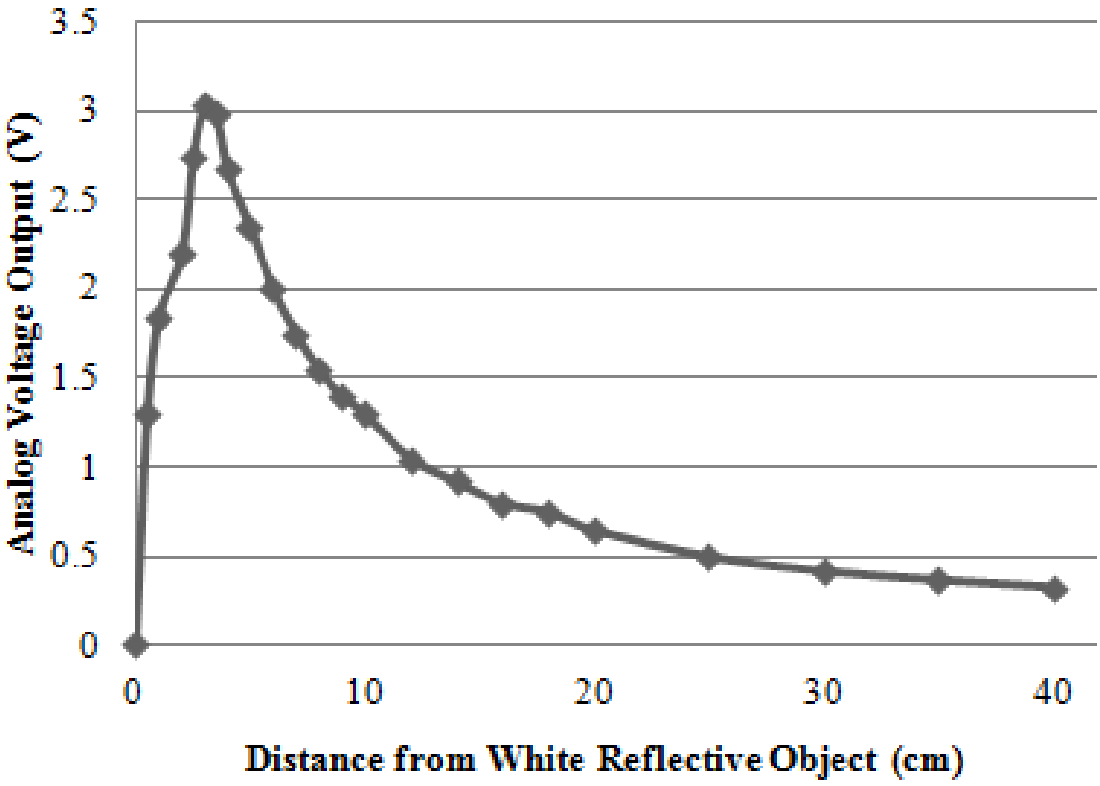


Figure 2.8: Sharp GP2D120 Analog Voltage Output **[21]**

2.3.3 Motors

Stepper motors and direct current (DC) motors are the two most common types of motors used in micromouse applications [15]. Ideally, the motors will be able to be easily controlled in terms of acceleration, position, and speed [15]. Table 2.2 presents the desired features for the micromouse robot and compares and contrasts stepper motors versus DC motors.

Table 2.2: Stepper versus DC Motors **[15]**

Feature	Notes	Selection
Weight	Stepper motors are generally heavier than DC motors	DC
Speed	DC motors perform better than stepper motors in regards to speed and acceleration; DC motor requires a gearbox to provide ample torque	DC
Control	Stepper motors achieve movement through stepping through small increments; DC motion control is often more difficult as feedback control plus encoders are necessary to obtain position information	Stepper
Mounting	Stepper motors are extremely easy to mount to chassis. DC motors require gearbox to leverage available motor torque.	Stepper
Torque	DC motors require gearbox to provide ample torque. Stepper motors are available off-the-shelf that provide suitable torque for micromouse robots.	Stepper
Software	Stepper motor software requires two independent pulses, and stepper motors are sensitive to achieving smooth pulse trains.	Stepper

Examining Table 2.2, either DC motors or stepper motors are appropriate for micromouse development. Selection is based on designer preference in comparing and contrasting the design tradeoffs of selecting one or the other.

2.3.4 Power Source

Batteries are the most practical power source for micromouse platforms. Batteries are rated based on a nominal voltage and power rating. Voltage readings will range from 15% above voltage rating (fully charged) to 15% below voltage rating (fully discharged), and batteries of different types provide different amounts of current [23]. In order to increase battery voltage, wire multiple batteries in series, and in order to increase current output, wire multiple batteries in parallel.

The following battery types are good candidates for micromouse robots: Alkaline, Lithium-Ion/Polymer, and Nickel Metal-Hydride (NiMH). The advantages and disadvantages of these battery types are highlighted in Table 2.3. When selecting a battery type, size and weight are also important factors. Battery size is limited by the size of the maze cells, and battery weight should be minimized in order to minimize the total weight of the robot [19]. Since many potential micromouse users may have limited experience with working with electronics, another factor that should be considered during battery selection is operational safety.

Table 2.3: Battery Type Comparison **[23]**, **[24]**

	Alkaline Rechargeable	**Nickel Metal-Hydride (NiMH)**	**Lithium-Ion/Polymer**
Pros	• Lose charge gradually (provides warning batteries need replaced) • Low self-discharge rates • High voltage • Non-toxic	• High current output • High energy capacity • Rechargeable as often as necessary • Lightweight • Non-toxic • Safest alternative	• High energy capacity • High power rates
Cons	• Low power capacities • Lose portion of capacity after each recharge	• Comes in different capacities (must purchase wisely) • Less voltage than Alkaline • Short shelf life when not in use	• Most expensive type • Potential to catch fire

It is important to recognize that battery size and selection will need to be considered during the design of the micromouse chassis. The micromouse chassis must have enough free space to support the battery pack, one of the largest and heaviest components. The battery should also be mounted to facilitate recharging.

2.4 Maze Algorithms (Software)

The micromouse robot's primary objective is to solve an unknown maze in a repeatable and efficient manner; as a result, the micromouse robot must deploy a maze solving algorithm in order to solve the course. Traditional maze solving algorithms typically break down when faced with real-world constraints as the maze information changes as the mouse progresses [25]. The following sections introduce five common maze solving algorithms: wall follower [25], DJIKSTRA [25], flood fill [15], modified flood fill [15], and A* algorithm [26], and discuss their respective advantages and disadvantages. Relevant to all algorithms is that on the returning trip from the maze center, it is often a good practice to double check the wall positions [15].

2.4.1 Wall Follower Algorithm

The wall follower algorithm involves a simple strategy to have the mouse follow either the left or right wall until the center/goal cell is reached. Since the wall follower algorithm is quite simple, its main advantage is that it is easy to implement. The disadvantage of the wall follower algorithm is that it lacks intelligence by not detecting position or direction [25]. There are many wall follower competitions; however, the algorithm is not suitable for the micromouse competition as mazes are designed such that this algorithm will not work [15].

2.4.2 Flood Fill Algorithm

The flood fill is an algorithm that determines the area connected to a given node in a multi-dimensional array. It sets the goal values (destination cells) to zero and "floods" the surrounding cells with radiating, increasing values. Figure 2.9 provides an example using a 5x5 maze.

4	**3**	**2**	**3**	**4**
3	**2**	**1**	**2**	**3**
2	**1**	**0**	**1**	**2**
3	**2**	**1**	**2**	**3**
4	**3**	**2**	**3**	**4**

Figure 2.9: Flood Fill Algorithm **[15]**

Examining Figure 2.9, note that the center value is zero and all other cells are filled with values corresponding to their distance from the goal cell. After "flooding" the maze with values, the algorithm then searches the adjacent nodes for the smallest value to determine which cell to travel [15]. It then continues to follow the values in descending order until it has reached the center. This algorithm is rather simple and provides a reliable method to finding the center of the maze. The disadvantages with the flood fill are that the software algorithm is more difficult to implement than the wall follower and the entire maze map has to be flooded after every move [27].

2.4.3 Modified Flood Fill Algorithm

Most implementations of the flood fill algorithm are tailored versions and are rightfully called modified flood fill algorithms. The modified flood fill algorithm is similar to the flood fill algorithm for it also uses distance values to navigate the maze [15]. The primary difference is that the modified flood fill algorithm does not "flood" the entire maze with values. It modifies only the values that need to be changed. For example, if a wall is encountered and the robot is not in the destination cell, it updates the value of that cell to ***1 + the minimum value of its open neighbors.*** Figure 2.10 provides an example of this process.

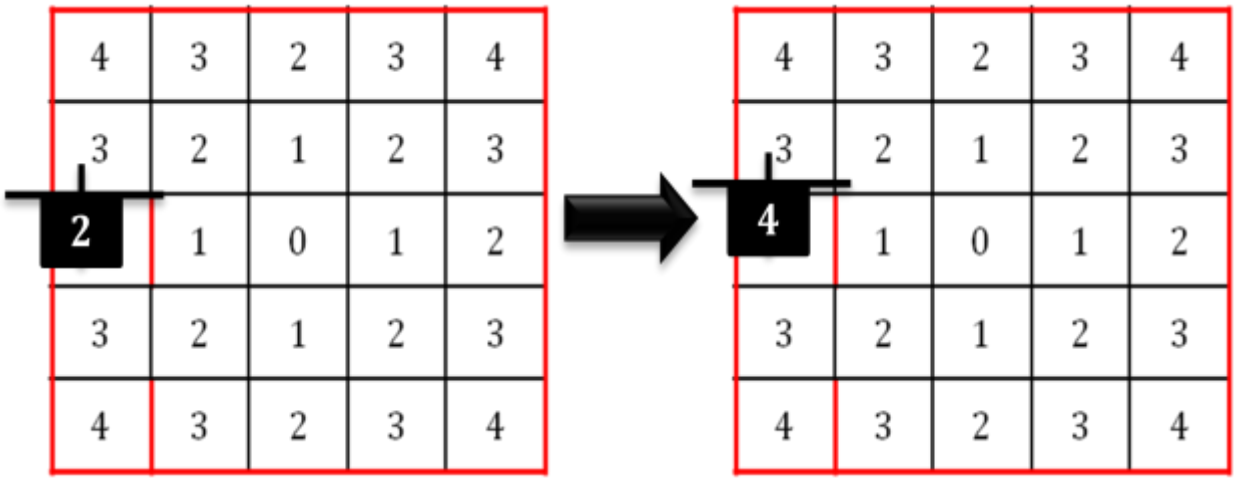


Figure 2.10: Modified Flood Fill Algorithm **[15]**

When the robot encounters a wall to the east and can only move north or south, the north and south cells (open neighbors) are checked and we find that the current cell's new value is ***1 + the minimum value of its open neighbors*** or 1+3=4. Once the robot has found the destination cell, it can return to the beginning of the maze using the distance values. Pseudo code for the modified flood fill process for updating the distance values [15] is provided.

```
Update the distance values (if necessary)
Make sure the stack is empty
Push the current cell (one robot is standing on) onto stack
Repeat the following instructions until stack = empty

{
      Pull cell from stack
      Is distance value of cell = 1+min value of open neighbors?
            No
            {
                  Change cell to 1+min value of open neighbors
                  Push all of cell's open neighbors onto stack to be
                  checked
            }
            Yes
            {
                  Do nothing
            }

}
```

The modified flood fill algorithm yields improvements over the flood fill algorithm in not requiring the entire maze to be flooded with new values upon each step of the maze. The disadvantage, as with the flood fill algorithm, is that the modified flood fill algorithm can be difficult to implement in software.

2.4.4 DIJKSTRA Algorithm

The DIJKSTA's algorithm is a more advanced algorithm that finds the shortest path for solving the unknown maze. For a node in the graph, the algorithm identifies the shortest path between the vertex and every other vertex [25]. An outline of the steps necessary to implement the DJIKSTRA algorithm is described is the following steps [25], [28].

```
/* Assign a distance value to every unvisited node except starting
node.  Set starting node = 0 and all other nodes = infinity */
for every i in the vertex
      distance (i) = infinity;
      previous (i) = undefined;
end for

/* The distance from the starting node to starting node is zero */
distance (s) = 0;

/* Examine all unvisited vertices and select the node with the shortest
distance value from starting point that is not in the Ready set */
while vertices exist that are unvisited
      pick the vertex,v, in unvisited vertices with shortest path to s
      add v to list of visited vertices
      for each edge of v (v1, v2)
            if distance(v1) + length (v1,v2) < distance (v2)
                  distance(v2) = distance (v1) + length (v1,v2)
                  previous(v2) = v1
                  update list of finished vertices as necessary
            end if
      end for
end while
```

The main advantage of this algorithm is that it finds the shortest path to solve the maze. The disadvantages are that the whole maze must be examined to identify the nodes as such it requires a significant amount of time to find the shortest path [25]. The algorithm is also requires a high demand on data memory [14].

2.4.5 A* Algorithm

The A* algorithm combines the DIJKSTRA algorithm (in finding the shortest possible path) with Best-First-Search information (utilizes heuristic function to guide itself toward the goal/center cell) [26]. As the maze is traversed, the A* algorithm maintains a sorted priority queue of alternate path options. If the portion of the maze has a higher cost (longer distance to goal cell) than any previously traversed portion, the algorithm goes back and traverses the lower-cost path. This process continues until the goal/center cell is reached. The most important component of the algorithm is the heuristic estimate function as this function directly relates to the efficiency and performance of the algorithm [26]. The A* star algorithm is an improvement over the DJIKSTRA algorithm; however, it is more difficult to implement. The A* algorithm also requires a high demand on data memory [14] which makes it a less attractive solution for microcontroller applications with limited available resources.

Chapter 3

Robotics Platform

The objective of this chapter is to provide a detailed description of the micromouse platform designed, developed, and deployed for the undergraduate computer engineering course, microcomputer structures and interfacing. The micromouse was selected as it provided a framework to design, develop, and utilize a low-cost robotics platform. The micromouse has a set of pre-existing rules that can be leveraged to form the foundation of the curriculum set.

The first part of this chapter introduces each major micromouse subsystem in isolation (drive train, sensor platform, and computing platform). The second part of this chapter describes the functional operation of the micromouse platform and the interactions between the micromouse subsystems. The third part of this chapter describes the micromouse application software and development. The final part of this chapter details the micromouse bill of materials.

3.1 Micromouse Architecture

The micromouse architecture is decomposed into the three subsystems: drive train, sensor platform, and computing platform. The drive train consists of a bottom frame plate, two servo motors, two ball castor wheels, and two rubber wheels. The sensor platform includes a sensor mount and three infrared sensors. The computing platform consists of the microcontroller platform and a top frame plate.

3.1.1 Drive Train

The micromouse drive train includes a bottom frame plate, two servo motors, two ball castors wheels, two rubber wheels, and connectors. The bottom frame plate was inspired by the commercial AIRAT-2 robot [11], shown in Figure 3.1, due to its simple and practical design. The plate material is an aluminum-alloy. The aluminum-alloy is a durable material with a high strength-to-weight ratio and will wear well over several years with little to no maintenance. The aluminum alloy material was also readily available at the university at no additional cost. The bottom frame plate stands one centimeter from the driving surface in order to lower the center of gravity. Figure 3.2 provides a CAD drawing of the bottom frame plate.

Figure 3.1: AIRAT-2 Robot Bottom Plate Picture **[11]**

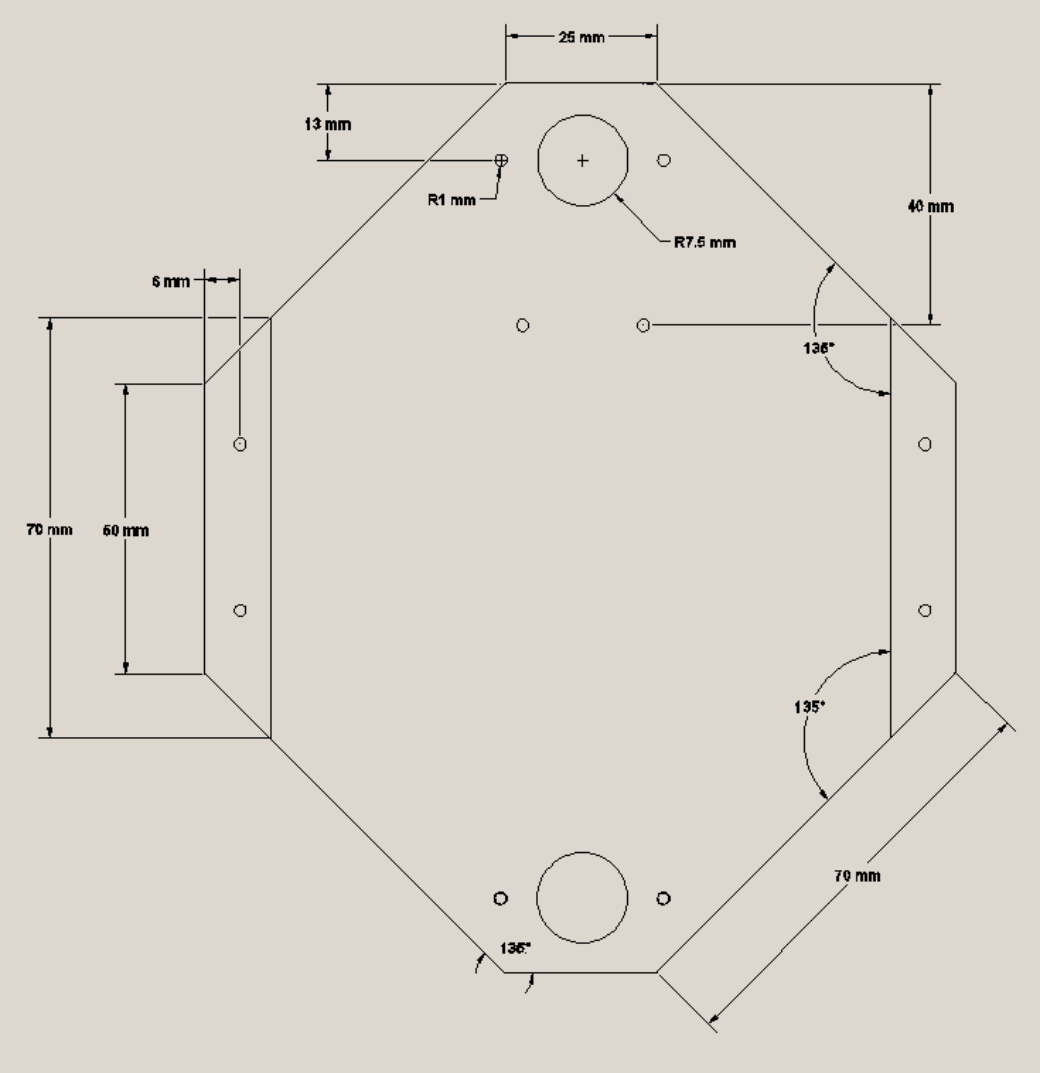


Figure 3.2: Micromouse Bottom Plate CAD Drawing

The wheelchair chassis was selected over the four wheel mouse to minimize the cost, weight, and complexity of the micromouse platform. The wheelchair design allows the utilization of low cost ball castors to provide the micromouse with sufficient turning capability. The castors are easily replaceable and can be adjusted to raise and lower the position of the micromouse. Two 7.5 mm (in radius) ball wheel castors are mounted through the bottom frame plate. The ball wheel castors are fastened with a piece of aluminum-alloy, spacer, and screws. The wheel castors can be adjusted higher or lower by inserting different sized spacers and fine-tuned by adjusting the screws. Figure 3.3 provides a picture of one of the micromouse ball wheel castors and mounts connected to the bottom frame plate.

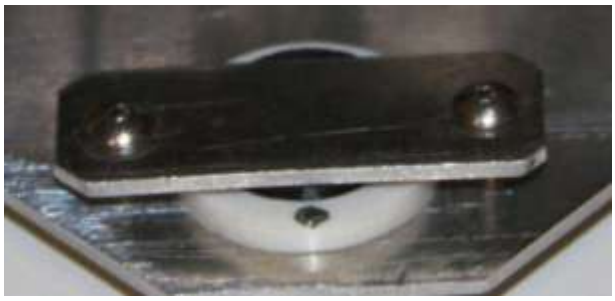


Figure 3.3: Micromouse Ball Wheel Castor and Mount

In the center of the micromouse, two Shinano Kenshi unipolar stepper motors [29] are mounted side-by-side. The primary reason that stepper motors were chosen over DC motors is due to the fact that stepper motors provide a more maintenance free micromouse robotics platform for the microcomputer structures and interfacing laboratory. DC motors require additional circuitry and a gearbox to have a suitable amount of torque for the micromouse application, whereas stepper motors can be simply controlled using two independent pulses. Utilizing the stepper motors provide students with the ability to precisely control the robot's movement by incrementing and decrementing the motors in small steps while being able to hold motor position and resist turning [19].

Connected to the stepper motor shafts are two 3.0 cm (radius) rubber wheels. Rubber wheels were chosen in order to reduce wheel slippage on the maze floor and minimize the total weight of the micromouse. Figure 3.4 provides a complete picture of the micromouse drive train including the bottom plate, ball caster wheels and mounts, stepper motors, and rubber wheels.

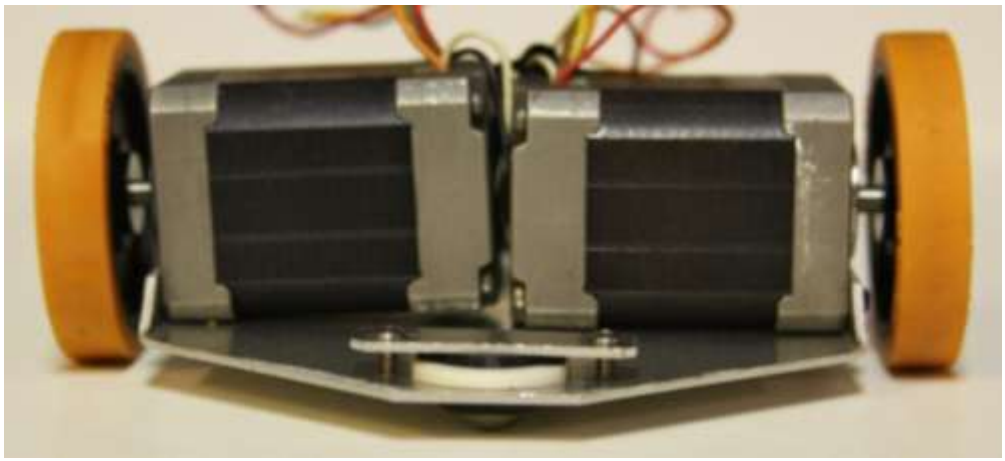


Figure 3.4: Micromouse Drive Train

3.1.2 Sensor Platform

The micromouse application requires a means to determine the location of the walls and detect when the micromouse gets too close to the maze walls during maze traversal. Sharp GP2D120 proximity sensors were chosen due to their small sampling time, low cost, and micromouse application-appropriate sensing range from 4-30 cm. The sampling time of the GP2D120 proximity sensor is 48 milliseconds.

Two mounting positions, shown in Figure 3.5, were initially tested to determine which mounting positioned better functioned. In sensor mounting method 1 (left), the sensors are mounted perpendicular to the reflective surface. In sensing method 2 (right), the front sensor is still perpendicular to the front wall; however, the left and right sensors are mounted at 60 degrees and 120 degrees, respectively.

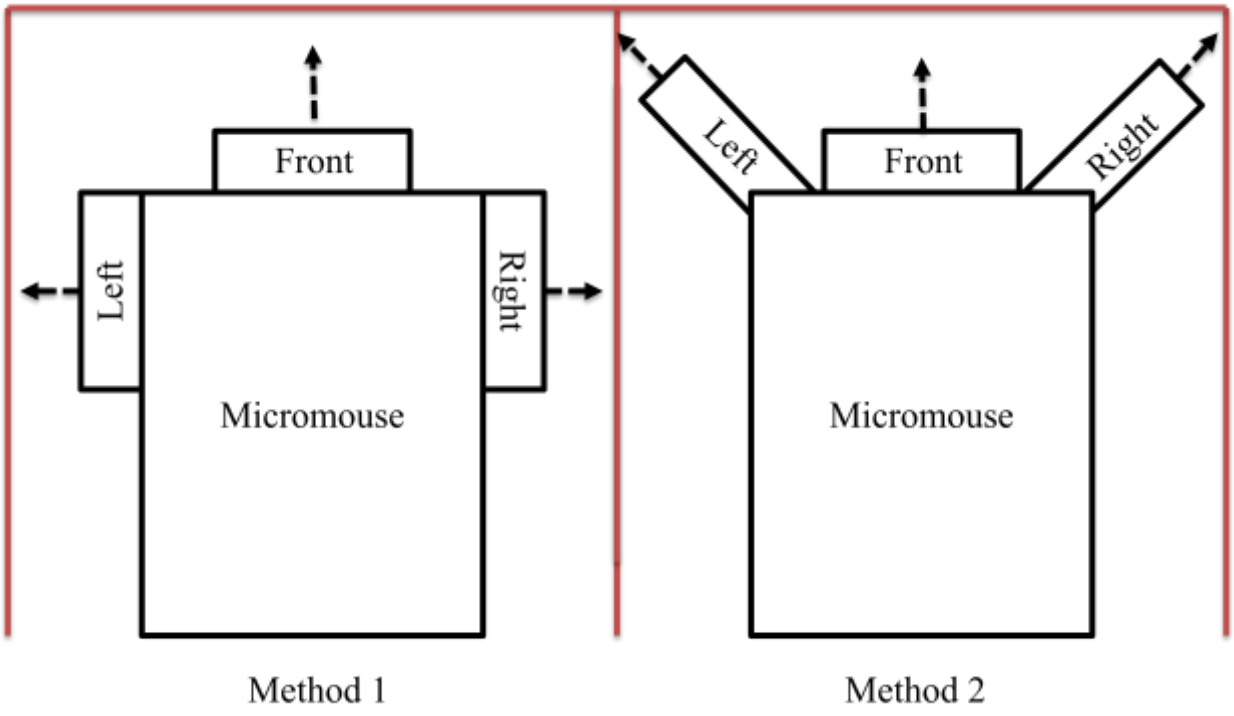


Figure 3.5: Sensor Mounting Positions

Table 3.1 provides the empirical results from reading proximity sensor outputs taken five centimeters away from a white reflective maze wall. Two measurements were taken at each

mounting angle, and the difference between the two measurements was calculated to determine the sensor consistency. The results demonstrated that sensors mounted at a 90 degree angle from the wall produce more regular outputs than sensors mounted at the 60 and 120 degree angles. Based on these results and since mounting the sensors at 60 and 120 degree angles proved more difficult than mounting sensors at right angles, it was determined that the sensors would be mounted to directly face the maze walls.

Table 3.1: Sensor Measurements taken at Different Mounting Angles

Method	Angle (Degrees)	Measurement #1 (V)	Measurement #2 (V)	Difference (V)
1	90	2.441	2.431	0.01
2	60	2.441	2.422	0.019
	120	2.432	2.456	0.024

Considering the sharp drop-off of proximity sensor readings outside of the detectable range (less than 4 cm for the GP2D120 sensor), sensor mounting method 1 was adjusted to ensure that the sensors could not get closer than 4 cm from the maze walls. This was accomplished by positioning the sensors within the body of the micromouse versus external to the mouse body as described in mounting methods 1 and 2. Figure 3.7 illustrates this adjustment and ensures that sensor readings are only taken in the sensor's detectable range. The sensors are mounted directly facing the maze walls.

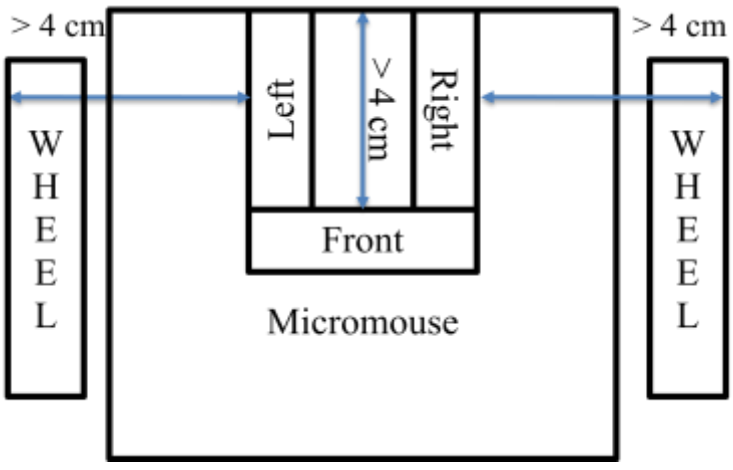


Figure 3.6: Micromouse Sensor Mounting Position

Figure 3.7 provides a CAD drawing of the micromouse sensor mount. An aluminum-alloy was also used to build the sensor mount. The sensor mount, shown in Figure 3.8, provides three vertical mounting points for the micromouse sensors. The sensor mount also connects to the bottom plate. The height of the sensor mount was driven by the height of the proximity sensors.

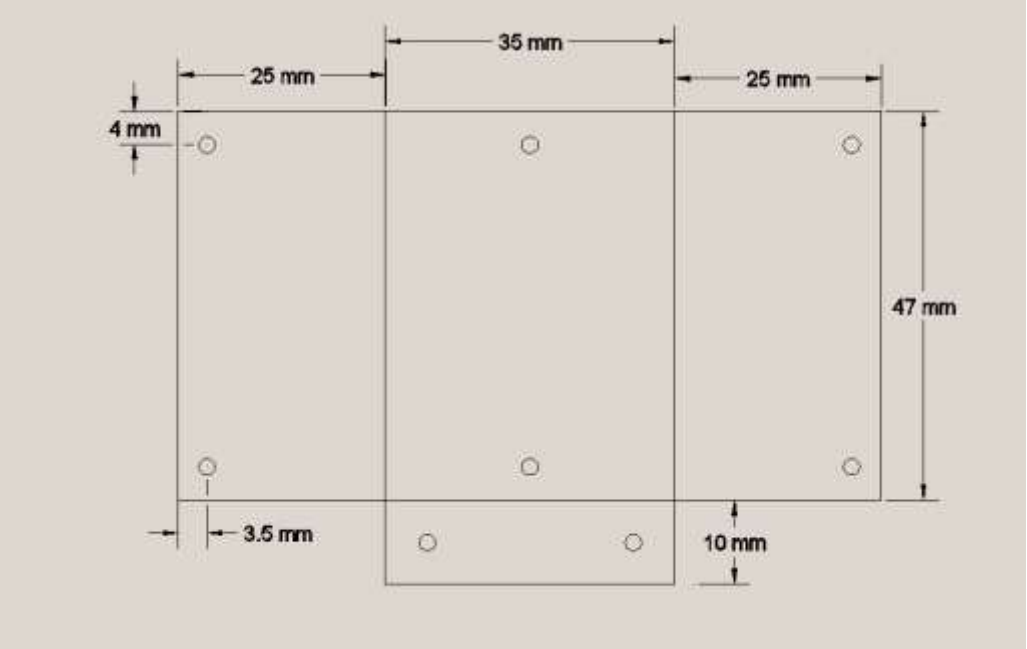


Figure 3.7: Micromouse Sensor Mount CAD Drawing

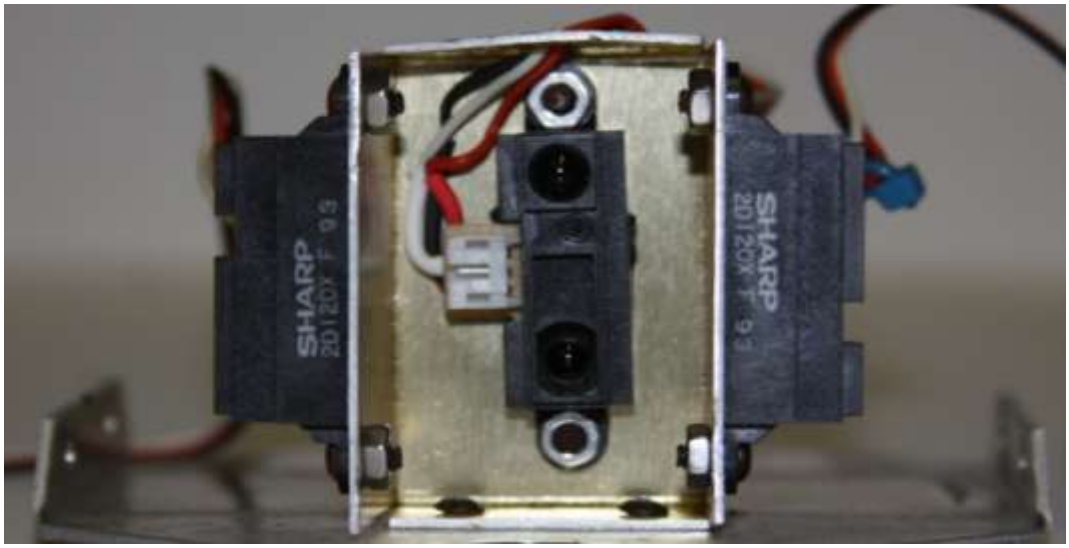


Figure 3.8: Micromouse Sensor Mount and Proximity Sensors

Since the GP2D120 infrared sensors vary in their output measurements on a sensor-by-sensor basis, multiple sensor readings must be considered to determine the range of values acceptable for a certain sensor reading (distance value). As a result, multiple sensor readings from multiple sensors were taken at different distances to identify a range of acceptable readings. Table 3.2 provides the list of sensor readings (converted to voltage readings, V) measured by the GP2D120 sensor and an average sensor reading at varying distances from the maze wall (4 to 8 cm). Figure 3.9 plots the voltages (V) versus distance from the wall. This information could be used for the micromouse and error correction algorithms to track the position of the micromouse within the maze.

Table 3.2: Sensor Values at Different Distances

Distance (cm)	Sensor Reading 1 (V)	Sensor Reading 2 (V)	Sensor Reading 3 (V)	Average Reading (V)
4	2.715	2.802	2.823	2.780
5	2.349	2.403	2.355	2.369
6	1.909	2.001	2.111	2.007
7	1.763	1.777	1.753	1.764
8	1.563	1.555	1.568	1.562

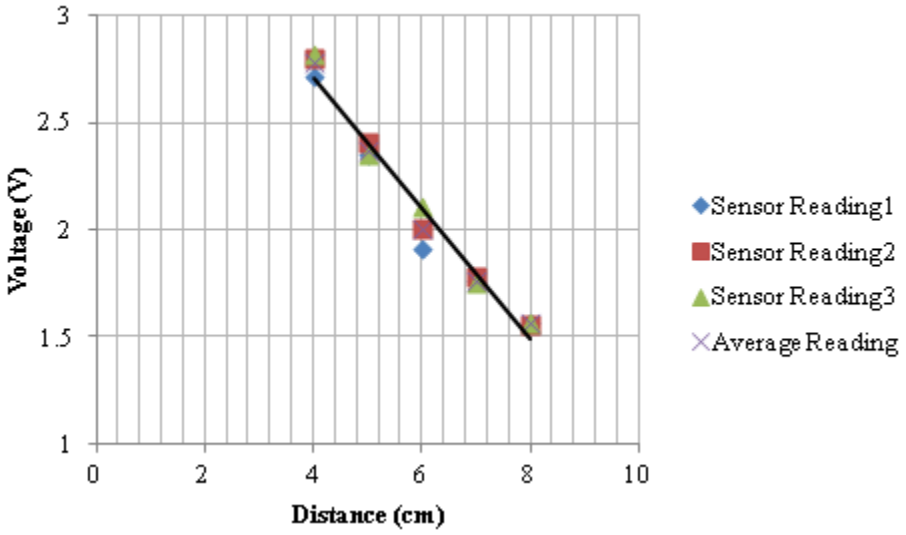


Figure 3.9: Proximity Sensor Output (Detectable Range)

The sensor data can be further simplified for the micromouse application for both the micromouse and error correction algorithms. First, to determine if a maze wall is present, a threshold can be identified to notify the micromouse algorithm of an existing wall. Thus if the sensor ADC reading comes back higher than this threshold, the micromouse knows that a wall is present. For example, left and right 10-bit ADC sensor readings greater than 240 indicate that a wall is present. Secondly, to input information to the error correction algorithm, a nominal value is measured by positioning the micromouse is the center of the maze cell. The 10-bit ADC measured value for this implementation of the micromouse application is 365 or 1.782V. Based on this sensor reading, the error correction algorithm can be tuned to adjust the path of the micromouse robot as it traverses the maze.

3.1.3 Computing Platform

A Dragonflybot single board computer was selected over the TinyDragon single board computer to serve as the computing platform due to the fact that the Dragonflybot is equipped with the necessary components to interface with two stepper motors (or DC motors) and simplifies the micromouse design. The Dragonflybot single board computer is mounted on top of the top frame plate as shown in Figure 3.10. Consequently, the top frame plate dimensions were driven by the physical dimensions of the Dragonflybot single board computer. Similar to the bottom frame plate, the top frame plate material is an aluminum-alloy metal. Figure 3.11 provides a CAD Drawing of the top frame plate.

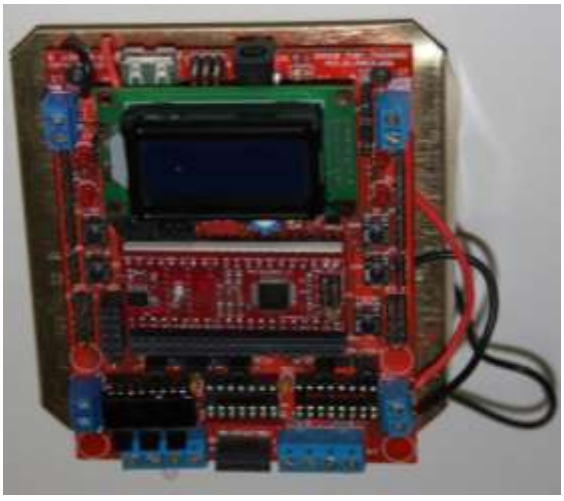


Figure 3.10: Dragonflybot Mounted on Micromouse Top Frame Plate

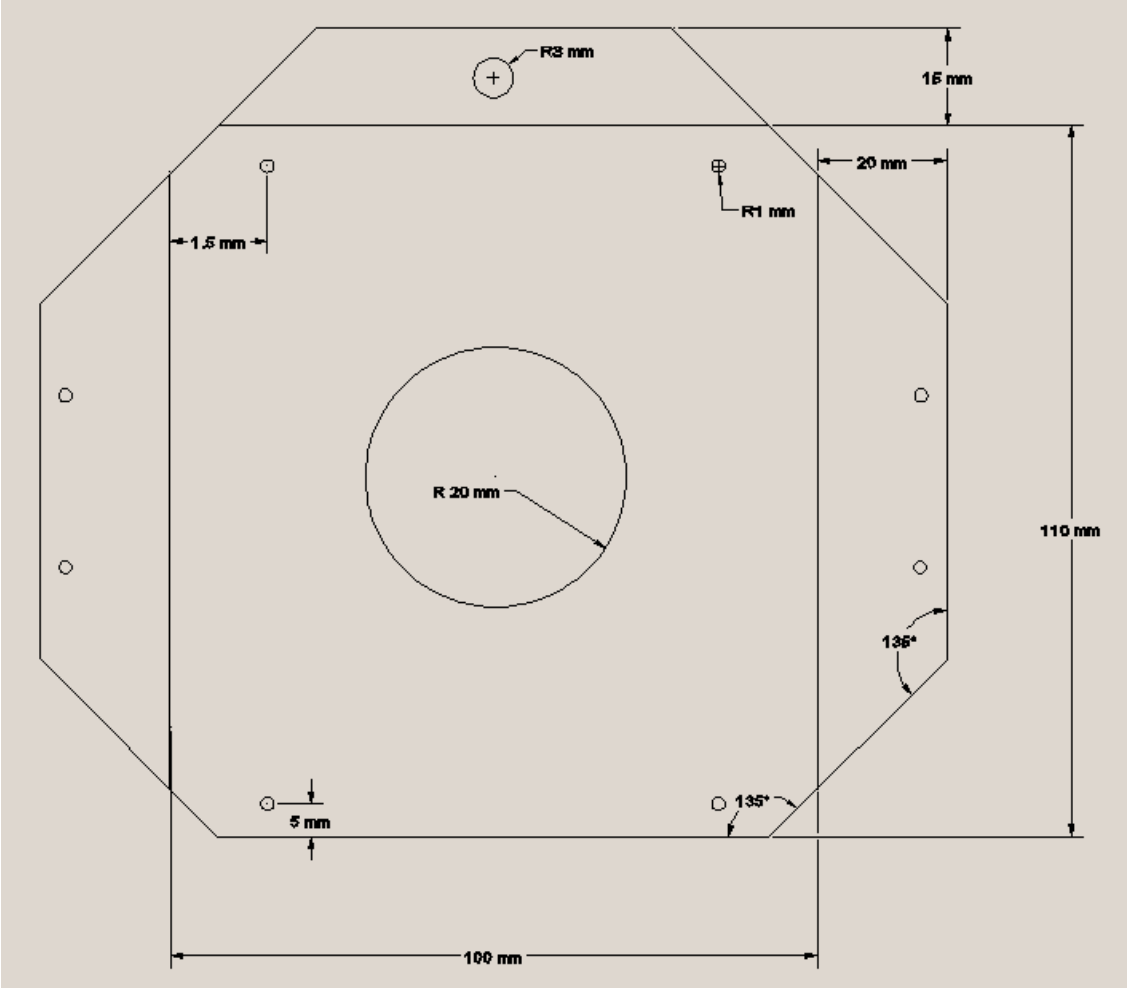


Figure 3.11: Micromouse Dragonflybot Single Board Computer Mount

A single 8.4 volt 1400 mAh nickel metal hydride (NiMH) rechargeable battery pack was selected as the power source for the micromouse. Separate power sources could have proved beneficial to energize the motor and microcontroller separately; however, a single battery was selected in order to minimize the size and weight of the micromouse. With this decision, the micromouse utilizes an 8.4 V battery with stepper motors that require 3.3 V input.

The NiMH batteries were selected because of their high current output and high energy capacity. In addition, the NiMH batteries were selected as they can be recharged and do not have to be completely discharged in order to recharge. NiMH batteries have no memory [23]. NiMH batteries are the safest alternative. These characteristics of NiMH batteries make them ideal for lab environments. The packaging style of the battery pack was chosen based on the amount of spare space on the rear of the micromouse. Figure 3.12 provides an image of the

battery pack. The battery pack fits snugly between the top and bottom plates and is not connected to the mouse. This allows the batteries to be interchanged between the robots.

Figure 3.12: 8.4 V 1400 mAh Ni-MH Rechargeable Battery Pack **[30]**

3.1.4 Micromouse Platform

The micromouse platform is assembled with the drive train, proximity sensors and mount, and Dragonflybot single board computer mount. The drive train provides mounting points for both the sensor mount and Dragonflybot computer mount. The sensor mount is connected using screws to the drive train bottom frame plate, and the Dragonflybot computer mount is connected to the two stepper motors with a pair of set screws. Figures 3.13 and 3.14 provide pictures of the assembled micromouse.

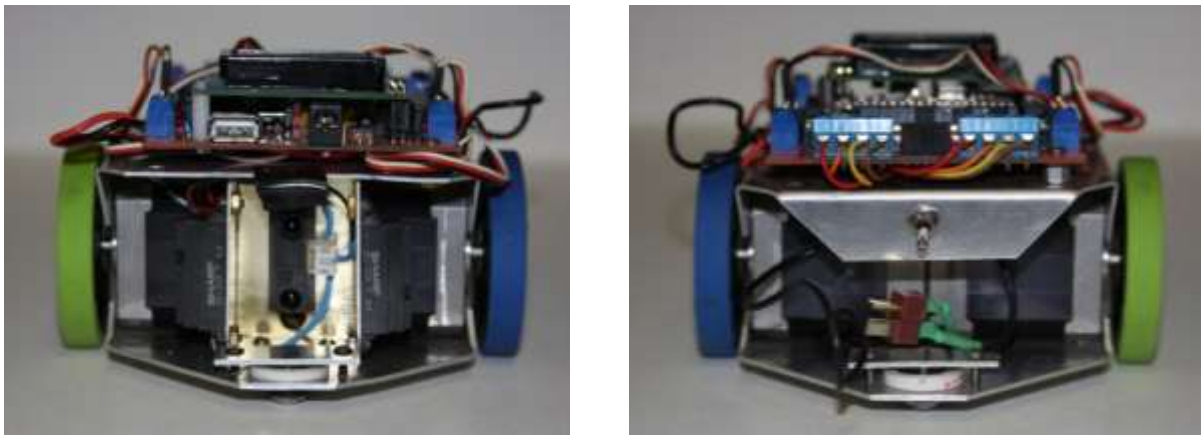


Figure 3.13: Micromouse Front and Rear View

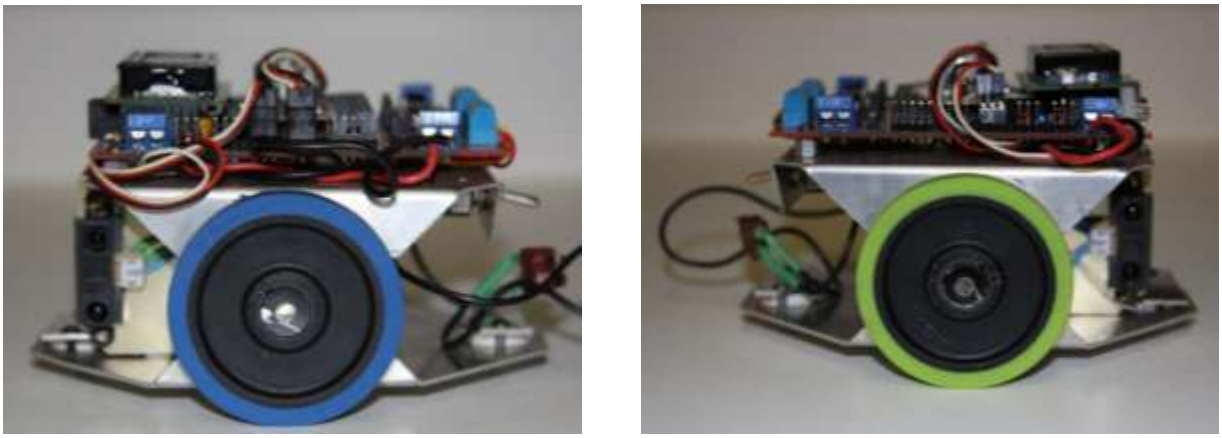


Figure 3.14: Micromouse Side Views

3.2 Functional Operation

The purpose of this section is to detail the interactions and connections between the micromouse major electrical components (infrared sensors, computing platform, stepper motors) and their functional operation in the micromouse application. Figure 3.15 provides an updated illustration of the micromouse electrical platform including these components.

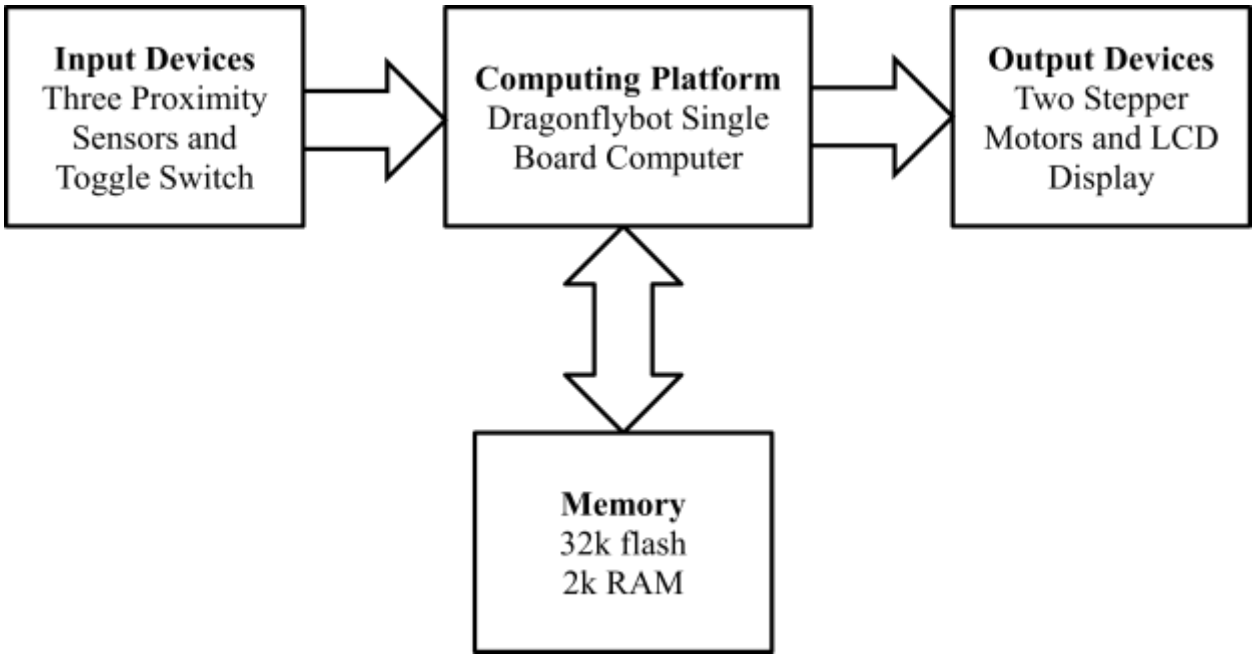


Figure 3.15: Micromouse Electrical Platform

3.2.1 Dragonflybot Single Board Computer

The Dragonflybot computing platform includes the Dragonflybot prototype board and the Dragonfly12 module. The Dragonfly12 module houses the HCS12 microcontroller and makes 40 breakout pins available for the micromouse application by connecting to the Dragonflybot board. The micromouse application utilizes the following components on the prototype board.

- Analog sensor input headers (AN0-AN2) for infrared sensor inputs (J5, J6, J7)
- Dual H-Bridge headers (U5 and U7)
- Motor terminal blocks (T4 and T5)
- External Motor Voltage (T3)
- LCD Port (J13)
- 5V Regulator for VCC (U2)

Figure 3.16 provides a component side schematic of the Dragonflybot prototype board, and Appendix B contains a list of the pin-out connections for the micromouse application

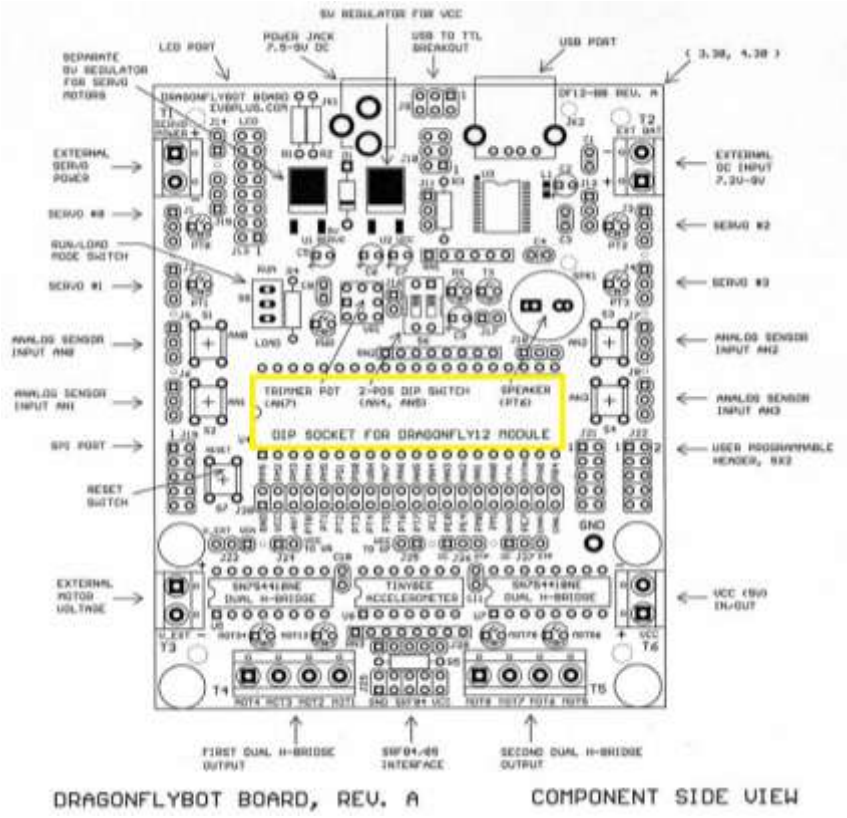


Figure 3.16: Dragonflybot Board – Component Side Schematic **[18]**

3.2.2 Infrared Sensors

The micromouse utilizes three Sharp GP2D120 proximity sensors. All three sensors are utilized to detect maze walls, and the right and left mounted sensors serve as inputs to the micromouse error correction algorithm. Each sensor requires a supply voltage, ground connection, and output voltage as shown in Figure 3.17. The Dragonflybot prototype board provides the required +5V supply voltage, ground connection, and input pins for the sensors. The analog sensor reading signals are converted to discrete numbers using the HCS12 microcontroller's analog-to-digital converters. It is also important to note that the GP2D120 proximity sensor takes 48 ms to take one distance reading. This time should be accounted for in the micromouse software algorithm.

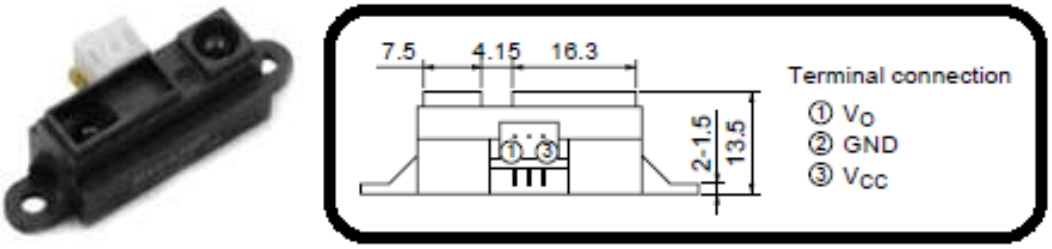


Figure 3.17: Sharp GP2D120 Proximity Sensor and Terminal Connections **[21]**

3.2.3 Stepper Motors

The micromouse utilizes two bipolar stepper motors to move the micromouse through the maze. Each stepper motor requires four wires to be driven by the HCS12 microcontroller. The port pins on the HCS12 microcontroller can only supply a maximum of 25 mA and the stepper motors require 1 A. This causes the need to introduce SN754410 half-bridge motor drivers into the stepper motor circuitry. The SN754410 half-bridge (H-bridge) is equipped with four half-H drivers whereas each driver provides 1 A output-current capability [31]. The H-bridge is utilized by turning on switches in pairs to allow current to flow through the motor. The H-bridge is enabled using enable lines and consists of a single input-output connection for each motor connection. Figure 3.18 shows a simplified diagram showing a typical H-bridge circuit.

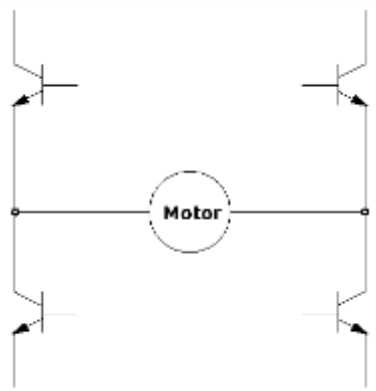


Figure 3.18: Half-Bridge Circuit

Each microcontroller output pin is thus connected to one of the four half-H drivers, and the output of the driver is connected to the respective stepper motor input pin. The Dragonflybot prototype board is equipped with two 16-pin connectors (U5 and U7) for the SN754410 motor drivers, and jumpers are available to configure the prototype board in this configuration for stepper motor operations. No additional wiring is required between the microcontroller, drivers, and stepper motors. Figure 3.19 provides a block diagram showing the connections from the microcontroller to one of the stepper motors.

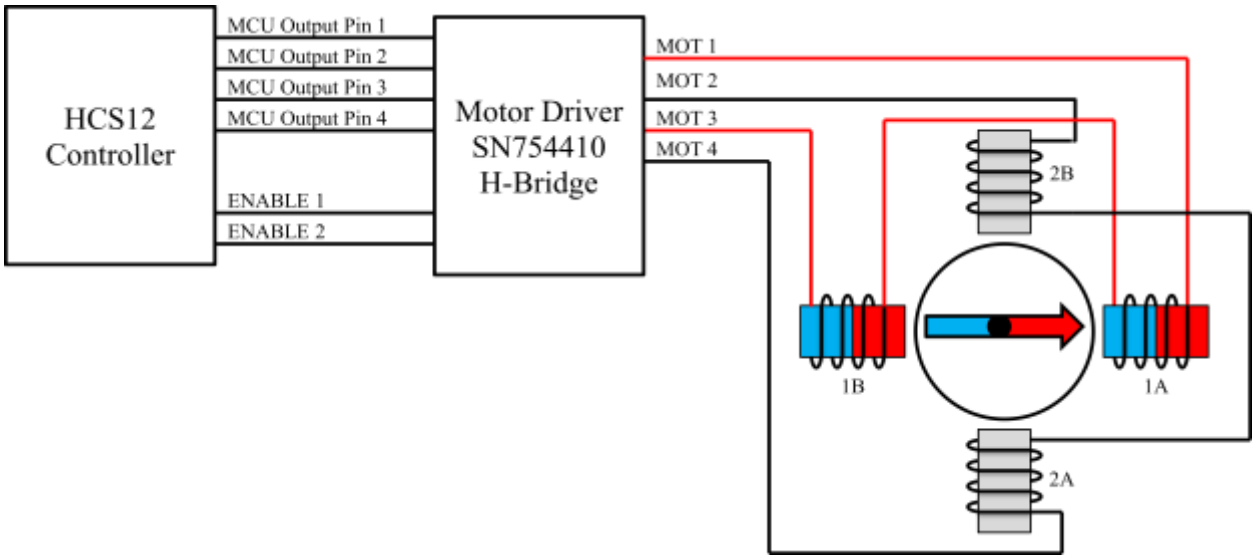


Figure 3.19: Microcontroller Interface to Stepper Motors

The stepper motors are rotated by properly energizing the motor coils in the correct sequence. Table 3.3 provides the energizing pattern used to energize the micromouse stepper motor coils. This energizing pattern energizes one phase at a time and consumes the least amount of power compared to other energizing patterns [32].

Table 3.3: Stepper Motor Energizing Pattern [33]

MOT 1	MOT 2	MOT 3	MOT 4
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0

A design limitation of the stepper motor circuitry was encountered during long duration micromouse runs. When the micromouse operated for long periods of time, the SN754410 motor drivers would often overheat. To mitigate the overheating of the drivers, each motor driver required a large heat sink to dissipate the heat. This solution was effective; however, it did not completely resolve the problem, and in future tests, the motor drivers continued to overheat. In order to mitigate this problem, a separate pulse was sent to the micromouse stepper motors in order to reduce the amount of current flowing through the drivers. Figure 3.19 provides an illustration of the original motor input (black) with the separate overlaid (gold).

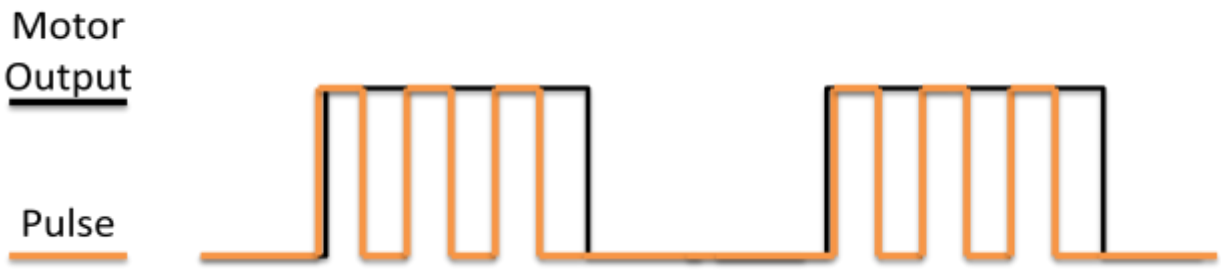


Figure 3.20: Motor Driver Input Pulse

Pulse width modulation (PWM) can be utilized to generate the separate pulse. The inclusion of PWM enhances the learning capabilities of the micromouse platform. This provides the opportunity to demonstrate PWM's ability to simplify the task of waveform generation [34].

Finally, the introduction of generated noise from the motors should also be considered when designing a micromouse robot. Noise is introduced by the motors when the current applied is switched on and off [19]. The simplest mechanism to limit the effects of noise is to utilize two separate power supplies, one for the motors and one for the remaining circuitry; however, the disadvantage with this strategy is that it increases the total weight of the robot by requiring additional batteries [19]. A more appropriate mechanism for the micromouse application is the utilization of capacitor circuits to reduce the noise effects. A filtering capacitor should be placed near batteries and any other sources of noise [19], and decoupling capacitors should be wired across power and ground on any Integrated Circuits [35]. A voltage regulator can also be utilized to keep the voltage used by the processor constant.

3.3 Software Platform

3.3.1 Software Development Environment and Summary

The micromouse software application was written in the C programming language (standard ANSI C) and was compiled, linked, and loaded onto the HCS12 target device using the Freescale Codewarrior Integrated Development Environment version 5.9.0 with Processor Expert plug-in version 3.02. The Freescale interface provides basic IDE programmer windowing and functionality. Processor Expert is a rapid application development tool that provides easy-to-use component-based application development [36]. Processor Expert utilizes software components (beans) to encapsulate a set of functionality that is commonly reused such as an input-output (IO) component; and Processor Expert utilizes methods for each component based on the functionality needed such as output to an IO component.

The microcontroller is programmed using an on-board HCS12 serial monitor. Using the serial monitor, one can connect a personal computer to the Dragonflybot board using a standard USB cable. Before programming, one must also ensure that the software application is configured to enable the phase lock loop (PLL) clock; set the clock frequency to 8 MHz; and set the internal bus clock to 24 MHz. Once the proper configurations are made, the Freescale CodeWarrior IDE makes it easy to debug, build, and load applications onto the target device.

The software was developed in a modular and iterative fashion. The software application includes six sets of Processor Expert beans and five user application files. The beans include all required internal peripherals to support CPU, LCD, Sensors, PID, Motor1, and Motor2 functionality. The user models include the main function and global variable declarations, interrupt service routines, motor code, maze solving algorithms, and LCD functions. Each user application was developed and tested as an individual module to ensure that each module worked properly and later integrated to create an example micromouse application. It is also important to note that some of the code modules were re-engineered from the code base of the spring 2009 micromouse robotics team [20].

3.3.2 Micromouse Application Overview

The micromouse application is composed of several integrated components to allow the mouse to solve an unknown maze. The main components within the software application are the LCD (for debugging and displaying information), motors (1 and 2); sensors (1, 2, and 3); Proportion-Integral-Derivative (PID) code, and maze solving code. Each of these components utilizes both Processor Expert methods as well as user-defined functions to implement the necessary functionality.

The general execution of the micromouse application is captured in Figure 3.21. At power-on, the hardware is initialized and all necessary variables and maze initializations are defined. Once all initializations are completed, the mouse enters a looping condition that is only completed when the mouse enters one of the center cells. This loop involves reading the proximity sensors, updating the wall map based on wall positions, correcting mouse alignment, determining the next logical move, and then moving to the desired cell. The most challenging aspect of the micromouse software is implementing the desired functionality in a small amount of memory (2K RAM and 32K Flash available). The software algorithm and data must be implemented and stored in an effective and efficient manner. Due to the limited resources, the modified flood fill algorithm was chosen as the maze solving algorithm as typically it only requires about 20 kB in program size and 2 kB of data [14]. More complex algorithms such as Dijkstra and A* require more resources. The following sections describe the micromouse software functions as shown in Figure 3.21. Appendix C includes a complete list of all the

Freescale Processor Expert methods utilized by the micromouse application. Appendix D includes a copy of the commented source code for the micromouse application.

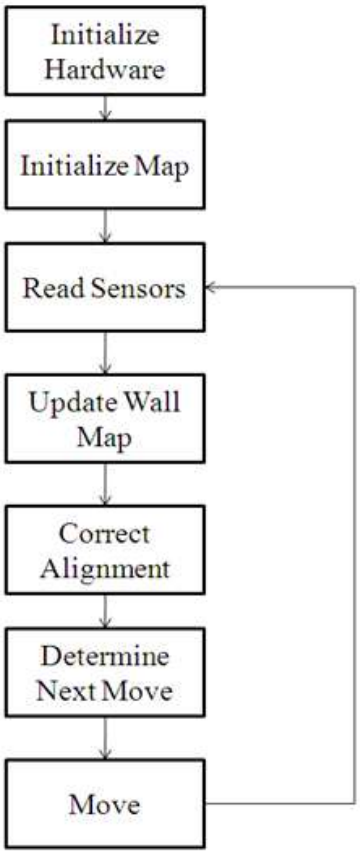


Figure 3.21: Micromouse Application Overview

3.3.2.1 Initialize Hardware

Initialize Hardware involves initialization of the Processor Expert generated code and the initialization of the LCD. The Processor Expert auto-generated code is initialized by calling the PE_low_level_init function which initializes beans and provides common register initialization. This initialization is utilized by the Processor Expert only. The LCD initialization routine turns on the 8x2 display, and initializes the LCD to operate in 4-bit mode with 2 lines and a blinking cursor. The LCD initialization routine also clears the LCD. The LCD code is only executed one time during start-up.

3.3.2.2 Initialize Map

Initializing the maze requires two straightforward functions. The first function initializes the maze distance values in a one-dimensional array as defined by the Modified Flood Fill Algorithm. The array created is of the size of the maze to be solved. In the case with the micromouse, the maze is a 6x13 array. As a result, the maze array is initialized as follows:

```
maze[] =  7 6 5 4 3 2 2 2 3 4 5 6 7
          6 5 4 3 2 1 1 1 2 3 4 5 6
          5 4 3 2 1 0 0 0 1 2 3 4 5
          5 4 3 2 1 0 0 0 1 2 3 4 5
          6 5 4 3 2 1 1 1 2 3 4 5 6
          7 6 5 4 3 2 2 2 3 4 5 6 7
```

The second function required to initialize the maze is to setup a wall map that will be used to by the modified flood fill algorithm to track the location of the walls. Since some of the wall locations are known beforehand (outer walls and first maze cell), the wall map will include a set of initial values. The values in the wall map (single dimension array) can be stored as byte values where each bit (0, 1, 2, 3) is used to indicate wall position. Bit 0 set means there is a southern wall; bit 1 set means that there is an eastern wall; bit 2 set means that there is a northern wall; and bit 3 set means that there is a western wall. Using this mapping, the wall map array is initialized as follows:

```
walls[] = 3 2 2 2 2 2 2 2 2 2 2 2  6
          1 0 0 0 0 0 0 0 0 0 0 0  4
          1 0 0 0 0 0 0 0 0 0 0 0  4
          1 0 0 0 0 0 0 0 0 0 0 0  4
          1 0 0 0 0 0 0 0 0 0 0 0  4
          9 8 8 8 8 8 8 8 8 8 8 8 12
```

3.3.2.3 Read Sensors

The proximity sensors on the micromouse serve two purposes. The first purpose of the sensors is to determine if the robot is getting too close to the maze walls. This information is provided using the left and right facing sensors only and serves as inputs to the error correction algorithm. The second purpose is to map the maze by determining the location of the walls (all three sensor readings are utilized).

In order to read the proximity sensors, the micromouse initializes an analog-to-digital converter (ADC) embedded bean to change the continuous signal into discrete digital numbers. The ADC is initialized with a 10-bit resolution, right justified, performs 8 conversions, and allows 20 microseconds per conversion. Within the application, the sensors are read at a regular interval as the main loop executes. The sensor values are stored in global variables to make them easily accessible to all other coding modules within the application.

3.3.2.4 Update Wall Map

The wall map contains the location of the walls within the maze. As the micromouse traverses the maze; the wall map must continuously be updated as specified by the modified flood fill algorithm. Once the maze is solved, the mouse will be located at the center of the maze and the distance value will be zero. As you move away from the maze each cell position, the distance value will be increased by one. The wall map is stored in an array of bytes where each bit represents the presence of a existing wall as described in Section 3.3.2.2.

3.3.2.5 Correct Alignment

Although the stepper motors are quite precise in their movements, as the micromouse traverses the maze, the mouse will veer from the center of each cell. As a result, the software application must provide a means to monitor and correct the micromouse's position. This functionality is accomplished using a proportional-integral-derivative error (PID) controller [37]. The PID controller is a control loop feedback mechanism used in many control systems and are widely used since they are easy to tune. Figure 3.22 illustrates the basic PID algorithm. The weighted sum of three controller parameters (proportional, integral, and derivative terms) is used to adjust the process.

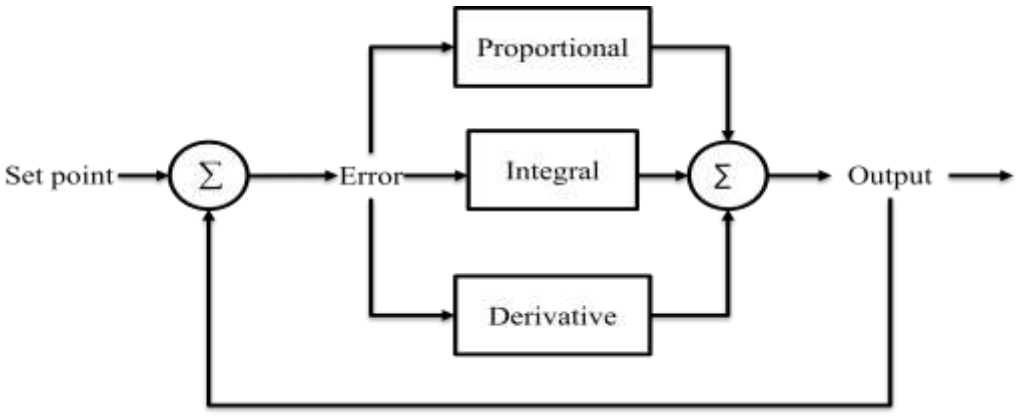


Figure 3.22: PID Controller Block Diagram **[37]**

The PID controller is tuned using three separate parameters: proportional, integral, and derivative. The proportional parameter (K_p) determines the controller's reaction to the current error, the integral parameter (K_i) determines the reaction based on the sum of the recent errors, and the derivative parameter (K_d) determines the reaction based on the rate at which the error has been changing [37]. K_p, K_d, and K_i, are the gains for each term. Table 3.4 captures the effects of each of the controller parameters. Figure 3.23 provides an example of response curves and captures the effects of the introduction of the control parameters.

Table 3.4: Effects of Controller Parameter

Controller Parameter	Rise Time	Overshoot	Settling Time	Steady State Error
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	No Change

For the micromouse application, the parameters are tuned to keep the micromouse in the center of the maze and avoid hitting the maze walls. The left and right infrared sensors are used as a feedback mechanism and flag when the micromouse gets too close to the wall. Additionally, the infrared sensors take approximately 55 microseconds per reading. This delay must be accounted for to ensure that the PID algorithm is not executing too quickly. As a result, a PID control loop of 100 milliseconds is utilized. Figure 3.24 illustrates the micromouse control loop algorithm.

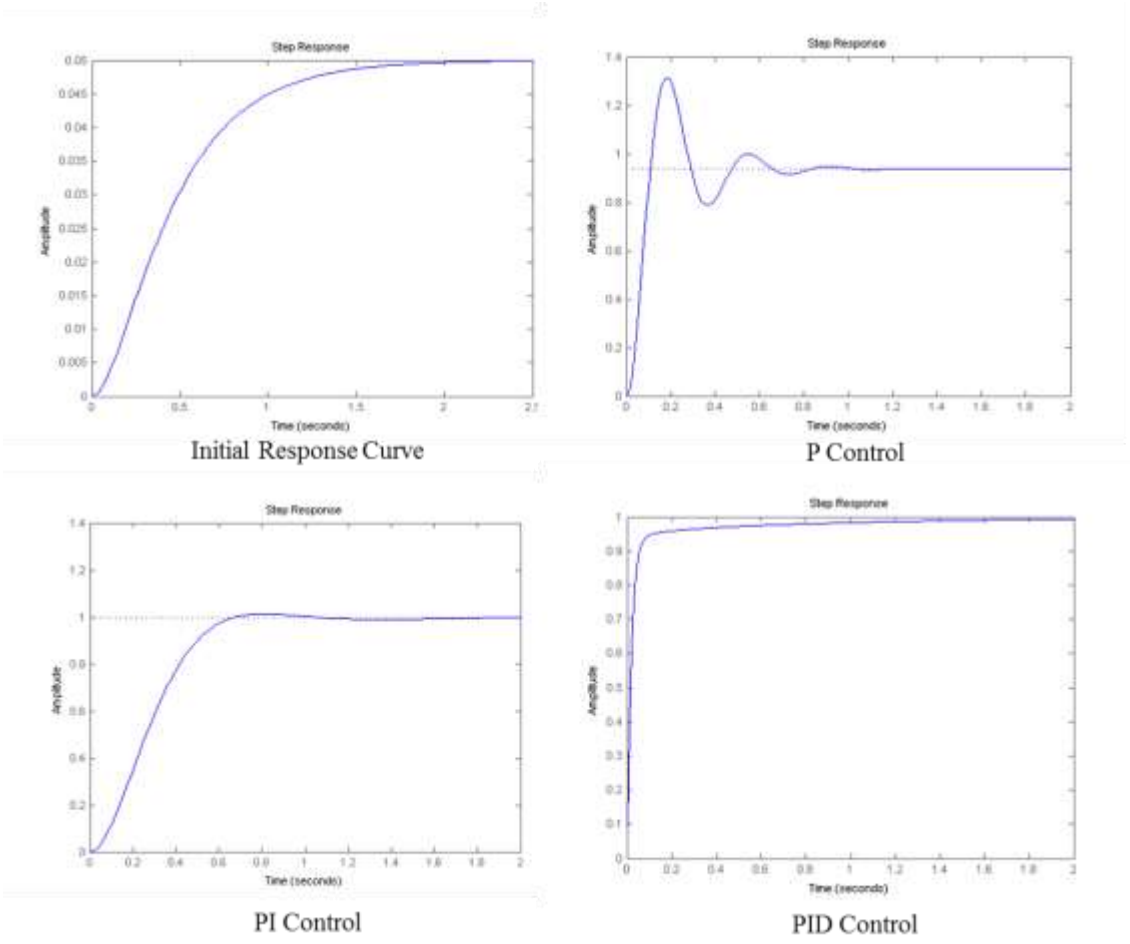


Figure 3.23: P, PI, and PID Control Responses **[38]**

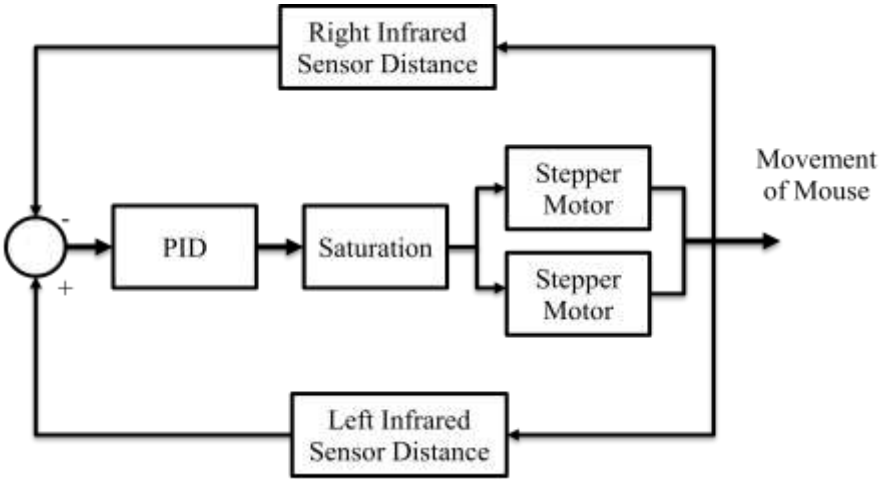


Figure 3.24: Micromouse Control Loop Diagram **[37]**

The micromouse PID algorithm was developed by selecting an appropriate set point for the infrared sensor readings. The micromouse set point was calculated by taking multiple sensor readings from both the left and right sensors with the micromouse placed in center of a maze cell in multiple locations within the maze. From these readings, average set point values were calculated for both the right and left sensors. The set point was utilized to calculate the error in the sensor readings by subtracting the actual value from the measured value. The higher the error, the greater the control adjustment needed.

In development of the micromouse PID algorithm, the proportional term was introduced first by implementing an immediate response to a movement error. The proportional term can be used solely to correct errors for the micromouse application; however oftentimes the proportional term causes steady state error and overshoot. The introduction of the integral term takes the sensor reading errors over time and makes adjustments based on the errors over time. The integral term improves the steady state error but cause addition overshoot. A proportional-integral (PI) controller was typically sufficient for the micromouse application. However if the overshoot was too high or settling time needed decreased, the derivative term was introduced to make adjustments to future error corrections.

It is recommended to keep the controller as simple as possible. Some successful micromouse applications have successfully traversed the maze with solely a proportional controller while others implement a PI or PD controller.

3.3.2.6 Determine Next Move

To determine the next move, the micromouse application checks all of the potential next moves. The potential next moves are *row+1, row-1, column+1,* and *column-1* (as long as there are no walls in the way). The micromouse now is ready to move into the cell with the lowest distance value (distance from the maze center).

3.3.2.7 Move Micromouse Forward

The final step in the application is to have the micromouse move into the correct cell. This functionality is accomplished using timer interrupts and global variables to keep track of how many steps to move. The approximate number of steps to move forward one cell is 378 steps.

Appendix C includes a complete list of all the Freescale Processor Expert methods that are utilized by the micromouse application. Appendix D includes the source code for all the major user-defined functions.

3.4 Bill of Materials

The micromouse robot bill of materials is provided in Table 3.5. The most expensive items were the Dragonflybot single board computer platform and stepper motors.

Table 3.5: Bill of Materials

Part	Qty	Price	Total
Dragonflybot Board	1	$78.00	$78.00
Stepper Motors	2	$15.00	$30.00
Proximity Sensors	3	$13.95	$41.85
Battery	1	$18.95	$18.95
Screws and Nuts	1	$7.20	$7.20
Banebot Wheels	2	$2.75	$5.50
Chassis & Mount Materials	1	$4.00	$4.00
Custom ball canister mounts	2	$1.75	$3.50
Sensor Connectors	3	$1.05	$3.15
Toggle Switch	1	$1.95	$1.95
4-40 x 1/4” standoffs	4	$0.20	$0.80
3 mm PVC spacers	2	$0.10	$0.20
		TOTAL	**$195.10**

The associated bill of materials’ costs demonstrate that the micromouse platform provides a framework to support the development of a rich set of seamless curriculum that can be integrated into undergraduate engineering curriculum with minimal associated costs.

Chapter 4

Results

Eleven micromouse robots were built to support the West Virginia University Lane Department of Computer Science and Electrical Engineering (LCSEE) microcomputer structures and interfacing undergraduate laboratory. Overall, the micromouse performed as well, and it provided a low-cost, durable educational robotics platform. A seamless set of micromouse curriculum was developed that included six laboratory modules that introduces students to basic microcontroller principles and programming and concludes with a tangible, operational product. An additional ten supplementary laboratory modules were developed to highlight specific topics such as I2C communications, digital sensors, servo motor operations, keypad interfacing, DC bidirectional motor control, basic input and output, and serial communications interface, etc. Appendix E includes copies of the six core laboratory modules as well as the micromouse final project description, and Appendix F includes copies of the ten additional lab modules.

The first part of this chapter details the six core micromouse laboratory modules and micromouse project curriculum. The second part of this chapter provides an assessment of the curriculum and its contributions and value to the microcomputer structures and interfacing laboratory.

4.1 Micromouse Curriculum Results

4.1.1 The Micromouse and Modified Flood Fill

The objective of the micromouse and modified flood fill laboratory is to introduce the micromouse platform, the micromouse competition rules, and the modified flood fill algorithm. The laboratory primarily provides an introduction to C programming. Introductory C programming topic areas are covered such as header and source files, conditional statements, for and while loop, arrays, global variables, use of extern, type casting, bit-wise versus logic operators, and recursion. The Freescale CodeWarrior Integrated Development Environment (IDE) is introduced. Students gain insight into the development environment that is utilized throughout the course of the semester and learn how to compile, link, and build HCS12 microcontroller projects.

Both the flood fill and modified flood fill searching algorithms are introduced in this module. As a result, several functions are implemented to update wall map, update distance values, determine the neighbor cell with the lowest distance value, and move to the neighboring cell with the lowest distance value. Students are encouraged to implement the algorithm as well using MatLab.

4.1.2 Microcontroller Basics

The objective of the microcontroller basics laboratory is to introduce the HCS12 microcontroller architecture and capabilities. A four bit counter is utilized to meet the lab's objectives. The module introduces the Dragonflybot project board, Dragonfly12 module, and the HCS12 microcontroller family and presents the documentation set for the MC9S12C controller, including the MC9S12C-family block diagram. This provides a high level understanding of its features such as voltage regulator, timer module, PWM module, CPU, memory, SPI, SCI, etc.

In addition, more details are provided on the CodeWarrior development environment. Details on using the Processor Expert plug-in are introduced. Processor Expert implements reusable software components to encapsulate the functionality of basic elements such as the CPU core, on-chip peripherals, and pure software algorithms. Specially, the CPU and basic IO reusable components are initialized and utilized.

4.1.3 LCD Interfacing

The LCD interfacing laboratory's objective is to develop a set of functions to communicate with an 8x2 LCD using the HCS12 microcontroller. The purpose of this module is to provide a set of reusable functions that can be utilized throughout the course of the semester to debug future modules as well as debug the micromouse application. The LCD datasheet is provided and students are encouraged to utilize it to develop the software functions versus using the handout. The following five functions are implemented for this module.

1. Initialize 8x2 LCD
2. Send single character to LCD
3. Send command to LCD
4. Clear the LCD
5. Display Numerical Values

The module also introduces additional topic areas such as bit shifting, basic input-output functions, and time delays.

4.1.4 Proximity Sensors

The purpose of the sensors module is to develop a set of functions to read the GP2D120 infrared sensor and utilize the returned values to determine the distance from an object (maze wall). The initial topic covered is the infrared sensor, its properties, electrical connections, and operations. The module requires the utilization of three analog-to-digital converter (ADC) channels to change the continuous sensor output signal to discrete numbers. ADC topics such as resolution, step size, accuracy, and conversion times are introduced. The module also introduces other types of sensors such as ultrasonic, touch, and sound and their operations and applications.

4.1.5 Stepper Motor Operation

The stepper motor operation module's purpose is to develop the functionality to control two stepper motors and essentially control the speed and direction of the micromouse. The module introduces the basics of stepper motors and their operation. Differences in unipolar and

bipolar motors are realized and driving stepper motors using timer interrupts. Pulse Width Modulation (PWM) is introduced as an option for simplification of waveform generation. Specifically, the following functions or some combination of these functions are implemented.

1. Move the micromouse forwards and backwards
2. Turn the micromouse left
3. Turn the micromouse right
4. Start/enable the stepper motors
5. Stop the stepper motors

4.1.6 PID Controller

The objective of the PID controller module is to implement a PID algorithm that can be utilized for the micromouse application. The module details an overview of the PID controller and walks through each of the controller parameters (proportional, integral, and derivative terms). The module encourages students to develop the controller in MatLab Simulink.

4.1.7 Micromouse Final Project

Upon successful completion of the core laboratory modules, students are prepared to implement their own micromouse application. The seamless set of curriculum is designed to provide the students three weeks to implement, test, and improve their micromouse application. The micromouse final project utilizes a subset of the IEEE Region 2 Student Activities Conference Rules. Appendix E provides a copy of the micromouse project description.

4.1.8 Micromouse Curriculum Summary

Table 4.1 maps the learning objectives for the undergraduate computer engineering microcomputer structures and interfacing laboratory to both the core micromouse laboratory modules and supplementary laboratory modules. The core micromouse laboratory modules directly map and satisfy all nine learning objectives. The introduction of the micromouse platform and curriculum enhances the current set of learning objectives to include control algorithms such as PID and wireless communications (IEEE 802.15.4 protocol XBee modules).

Table 4.1: Micromouse Curriculum

Objective	Module
1. Draw a detailed architecture diagram of the HCS12 microcontrollers	Core: Microcontroller Introduction and Basics
2. Write a program for HCS12 microcontrollers, i.e. know all addressing modes and full instruction of the HCS12 microcontrollers; and how to write a program using HCS12 assembler and debugging tools	All Laboratory Modules (Core and Supplementary)
3. Design and use interrupt as an integral part of the microcontroller	Core: Proximity Sensors Core: Implementing PID Controller Supplementary: Servo Motor Control using a Phototransistor
4. Design and use programmable timers as an integral part of the microcontroller	Core: Microcontroller Introduction and Baiscs Core: Stepper Motor Operation Supplementary: Alarm Clock Supplementary: Square Wave Frequency Calculation
5. Design and use input/output ports as an integral part of the microcontroller	Core: Microcontroller Introduction and Basics Core: Micromouse Final Project Supplementary: Bidirectional DC Motor Control
6. Interface and develop software to output a wide-range of process control signal (DC, AC, high power, variety of voltage/current compatible levels, I2C, Keyboards, LCD)	Core: LCD Interfacing Core: Micromouse Final Project Supplementary: Keypad Interfacing Supplementary: Alarm Clock Supplementary: I2C and DS1624 Digital Thermometer
7. Design and use Pulse Width Modulation (PWM) as an integral part of the microcontroller	Core: Micromouse Final Project Supplementary: Bidirectional DC Motor Control
8. Understand the basic operation and use of Analog-to-Digital Converter (ADC) and Digital-to-Analog converter with microcontroller	Core: Proximity Sensors Supplementary: ADC Using the Potentiometer Supplementary: XBee Wireless Module Interfacing
9. Understand the basic operation of commonly used sensors/transducers	Core: Proximity Sensors Supplementary: I2C and DS1624 Digital Thermometer
** Enhancement to microcomputer structures and interfacing laboratory: Common Feedback Controller	Core: Implementing PID Controller Supplementary: XBee Wireless Module Interfacing

Table 4.2 provides a list of the supplementary laboratory modules. The supplementary modules (servo motor operation and using photodetectors, bidirectional DC motor operations, and I2C communications and DS1624 temperature sensors) were developed to enhance specific learning objectives. These supplementary modules can optionally be integrated into the

micromouse curriculum to enable specific topics to be given an increased amount of coverage on an as-needed basis. Appendix F includes copies of the supplementary modules.

Table 4.2: Supplementary Modules

#	Optional Module	Objectives	Topic Areas
1	Keypad Interfacing	Interface a 4x4 keypad using HSC12 microcontroller	Keypad initialization, basic IO, and CPU generated time delays. Introduction to reusable software components with Freescale Code Warrior IDE
2	Serial Communications Interface (SCI)	Introduce SCI communications device by configuring microcontroller to communicate with HyperTerminal application through RS-232 port	SCI communications, baud rate, parity checking
3	Alarm Clock	Develop an alarm clock using a timer interrupt, buzzer, and HyperTerminal display	MCU Project Board, buzzer, timer interrupts, working with interrupts in Freescale Code Warrior (events.c)
4	I2C and DS1624 Digital Thermometer	Implement Inter Integrated Circuit (I2C) external communications to read a DS1624 digital thermometer	I2C communications, master/slave logic, digital input device (DS1624)
5	Square Wave Frequency Calculation	Utilize the HCS12 timer capture function to calculate the period and frequency of a square wave	Function generator, square wave, timer capture, frequency and period calculations
6	Controlling DC motor and Servo Motors using Pulse Width Modulation (PWM)	Control the speed and rotation of a DC motor using PWM and control servo motors with PWM	PWM, DC motor control and operation, L293B push-pull four channel driver, bypass capacitor, basic IO, switches, servo motor operations, diodes
7	ADC Using the Potentiometer	Utilize a potentiometer to generate an output voltage than can be read by an analog-to-digital converter. Display the digital voltage reading.	ADC, step size, resolution, accuracy, potentiometers
8	XBee Wireless Module Interfacing	Develop custom applications with the Xbee wireless module. Demonstrate communications between two Xbee modules	UART, ADC, step size, resolution, accuracy, network topologies (point-to-point, point-to-multipoint, peer-to-peer), X-CTU software application
9	Bidirectional DC Motor Control	Create user interface using keypad and LCD to control a bidirectional motor and lamp dimmer	Basic IO, keypad, LCD, PWM, DC motor, L293 driver, bypass capacitor, lamp dimmer, diodes, SPI
10	Servo Motor Control using a Phototransistor	Utilize two servo motors, parabolic mirror, and phototransistor to locate the position of a flashlight	Basic IO, keypad, LCD, PWM, servo motor, SPI, ADC, shift register, phototransistor

4.2 Micromouse Curriculum Assessment

The micromouse curriculum was implemented during the spring 2010 and 2011 semesters for the West Virginia University Computer Engineering 313 laboratory. Twenty-eight students enrolled and completed the laboratory during the spring 2010 semester, and forty-six students enrolled and completed the laboratory during the spring 2011 semester. These students' grades/performance and motivation form the basis of these results. The results from the 2010-2011 semesters are gauged against prior semesters from 2007-2009 where traditional-based curriculum was utilized (2007-21 students; 2008-24 students; and 2009-20 students).

Table 4.3 provides the details of the average student scores, standard deviation of the scores, and number of students that dropped the course in 2010 and 2011 (micromouse) compared to the student statistics from 2007-2009. Note that the average scores are higher and the standard deviation is smaller in the semester that the micromouse curriculum was implemented. During the 2010 and 2011 semesters, it was also noted that no students dropped the course during the semester as in previous semesters. This is most likely due to other factors but in future semesters, in which the micromouse curriculum is implemented, it may be worth monitoring.

Table 4.3: Laboratory Statistics (2007 – 2011)

Semester	2007	2008	2009	2010	2011
No. of Students	21	24	20	28	46
Mean	89.89	81.19	85.57	91.71	89.84
Standard Deviation	7.63	26.44	9.71	5.42	8.22
Dropped*	2	1	1	0	0

*Dropped is defined as the number of students that dropped out of the course after the beginning of the semester and before the semester ended.

Student motivation is the area in which the implementation of the micromouse curriculum demonstrated the most noticeable results. Students during the semester of the micromouse curriculum were consistently more motivated than students from previous semesters. Students not only put forth additional effort to complete the micromouse project but spent considerable more time than in previous semesters to improve their design and

implementation. In order to rate the student's motivation, three categories are defined: excellent, average, and below average. An excellent rating is defined as an extremely motivated student which put forth considerable effort to engineer and re-engineer final project. An average rating is defined as a student that spent the allocated lab hours in additional to some extra time to complete the final project. A below average rating is defined as a student that spent an inappropriate amount of time to complete the final project. Table 4.4 provides the student motivation ratings from 2007-2011.

Table 4.4: Student Motivation Ratings (2007 – 2011)

Semester	2007	2008	2009	2010	2011
Below Average	1	2	0	1	4
Average	18	20	18	21	33
Excellent	2	2	2	6	9

Examining Table 4.4 from 2007-2009, 6 out of 65 (9.23%) students demonstrated excellent motivation towards the project curriculum and final project whereas in 2010 and 2011 (micromouse curriculum), 15 out of 74 (20.27%) students demonstrated excellent motivation towards the project curriculum and final project. Student motivation was a factor 2.20 higher when the micromouse curriculum was implemented versus previous semesters when it was not implemented. Student motivation and the benefits of introducing the micromouse curriculum were also demonstrated in many of the comments that were found in the students' final projects.

> "This entire lab series presented a number of interesting challenges which encouraged critical thinking and clever solutions. It presented a large spectrum of problems while still managing to keep them all related and tied to a greater theme. That and the ability to actually see your effort accomplish something real,..."
> ~ Kaur, Costa

> I had a good time working on this project; though we didn't achieve every goal, the satisfaction of knowing how much was learned and accomplished is enough to relish the class as a whole.
> ~ Blosser, Thomson

"project brought together all the labs that we had done during the semester"
~ Cox, Justice

Overall, it was definitely the most cohesive set of labs that I have ever taken and definitely the best lab experience I have ever had. Thanks for going above the call of duty to create such a wonderful lab!
~ Dilello, Britt-Cane

"This setup resulted in a more positive learning experience and environment in the lab. The purpose and necessity of each lab was clearly visible which in turn results in a more motivated group."
~ Boustany, Glymph, Wight

"Topics that were discussed in class were demonstrated and exemplified in our lab exercises. This provided for a deeper understanding of the fundamentals."
~ Walters, Crawford, Smith

"Getting to see the micromouse come to life over the course of the semester, bit by bit, was definitely the best part."
Kelly, Perhinschi, Tucker

"Overall, this was a great project to teach us about the practical uses of microcontrollers such as the HCS12. We were introduced to a wide variety of topics including LCDs, beans, analog to digital converters, sensors, stepper motors, PID controllers, and C coding. Not only did we learn a lot while working on this project, we also had fun."
~Bowman, Hamilton

The seamless set of micromouse platform and curriculum has been developed, implemented, and improved upon through the course of this effort. These results demonstrate that the introduction of the micromouse platform and curriculum has significantly enhanced the WVU LCSEE microcomputer structures and interfacing undergraduate laboratory by expanding upon the courses learning objectives, introducing more complex topic areas, and motivating students to not only meet the lab objectives but also strive to perfect their individual micromouse application.

Chapter 5

Conclusion & Future Work

5.1 Conclusion

This thesis resulted in the development of a low-cost micromouse robotic platform. The platform was developed by researching and testing alternative solutions for the various micromouse subsystems (mechanical, electrical, and software). The mechanical subsystem was driven by maze dimensions. Factors such as mobility and microcontroller board were taken into account to design and build the micromouse. The mechanical subsystem resulted in a durable and flexible robotics chassis capable of supporting multiple sensors and motors. The electrical subsystem included the Dragonflybot board, three proximity sensors, two stepper motors, and a NiMH battery. The Dragonflybot was selected for the micromouse application and was equipped with convenient connectors for the proximity sensors and stepper motors. The proximity sensors were ideal for the micromouse application. The stepper motors provided precision control and were operated with limited circuitry. The NiMH battery provided a safe and effective power source. The software subsystem implemented the various algorithms and controls necessary to operate the micromouse. The most significant accomplishment in the software subsystem was the demonstration to implement the micromouse algorithms in a small memory space.

The results of this thesis provide a rich set of curriculum that supports and enhances the learning objectives of the microcomputer structures and interfacing laboratory. The core set of curriculum includes learning modules that are centered on the micromouse robot and the IEEE Region II Micromouse Competition Rules. In addition to the core set of micromouse curriculum, additional topic-specific lab modules were developed to supplement the theoretical learning from the lecture portion of the CPE 313 laboratory (microcomputer structures and interfacing). These topics enhance the undergraduate computer engineering curriculum. The specific topics developed under this research include I2C communications, XBEE wireless communications, timer capture, DC motor theory and operations, and servo motor operations.

The integration of the micromouse platform into the microcomputer structures and interfacing laboratory transformed a modular-based lab environment into a seamless set of laboratory sessions. Traditional lab sessions introduced a new learning module each week, and there was no overlap or connection between the set of labs. The micromouse curriculum provided students with a clear set of objectives throughout the semester. The integrated curriculum made it transparent to students the importance of every lab session, and as a result, students were more motivated than in previous semesters to ensure that they understood the content.

The micromouse provides a flexible, low-cost robotics platform that can be expanded to be utilized in middle school and high school science, technology, engineering, and mathematics courses and robotics teams. General topics such as problem solving, environmental factors, design parameters, constraints, and trade-off analysis are inherent to the micromouse problem. Specific topics such as analog-to-digital conversion (ADC), stepper motor operation, control algorithms, maze algorithms, and proximity sensors can also be realized with this platform.

The following contributions are made by this thesis,

1. A low-cost robotics platform with accompanying curriculum was designed, developed, and utilized to accomplish the learning objectives of the West Virginia University's Lane Department of Computer Science and Electrical Engineering Microcomputer Structures and Interfacing Laboratory, an undergraduate computer engineering course. Evaluation of the deployment of developed curriculum provided evidence that competitive-based learning increases motivation and performance for a subset of students.

2. The developed robotics platform identifies a low-cost robotics platform that could be utilized in high school science, technology, engineering, and mathematics courses and robotics teams. The platform has the potential to supplement current robotics programs in the state of West Virginia.
3. Additional topics with supporting hands-on modules to supplement theoretical learning were introduced into the core set of the undergraduate computer engineering curriculum including I2C communications, and XBee wireless communications.

5.2 Future Work

The micromouse performed well and fully satisfies the requirements of the microcomputer structures and interfacing laboratory. The platform subsystems (mechanical, electrical, and software) were effective and efficient; however, additional improvements can be made to better the platform's flexibility and utility for both mobile robotics research and curriculum development. Regarding the mechanical platform, the author recommends that the micromouse dimensions be reduced to provide more leeway for error detection and correction during maze traversal. Future versions of the as-designed micromouse should be taller (in height) and smaller in width and length. This will provide the micromouse more room for error correction handling and maneuvering the maze. Regarding the electrical platform, future versions of the sensor mounts should provide students with the option to mount sensors in different locations as well as provide the capability to mount additional sensors to provide sensor redundancy. Regarding the electrical platform, DC motors and drivers should be available to support hands-on experimentation of stepper motor operation versus DC motor operation. Ideally, a second set of micromouse robots would be designed and built to include DC motors. In regards to the sensors, it is recommended to explore the addition of I2C and/or SPI digital range sensors to the micromouse platform to enhance its capabilities.

It is to consider migrating from the current microcomputer structures and interfacing hardware platform to a different platform such as Raspberry Pi. The Raspberry Pi is a very small single board computer that consists of much more resources that the Dragonflybot. The Raspberry Pi is equipped with 256 MB or 512 mB RAM, USB port(s) and an optional Ethernet port that can be purchased for $25-$35 [39]. Additionally, configuration options should be added to the micromouse framework such as wireless communications. This addition would not

be suitable for the micromouse competition; however, it would improve the capabilities and benefits of the micromouse framework.

The evidence of increased student motivation makes a strong case for more integrated laboratories. The results of this effort serve as a starting point for future investigation of robotics based curriculum and platforms in the engineering students. Parallel efforts to the micromouse platform should be undertaken in other laboratory sessions as appropriate.

The micromouse platform also demonstrated to be a low-cost, effective solution to introduce engineering concepts such as digital input-output, analog-to-digital converters, pulse width modulation, etc. Considering the currently available robotics programs in the state of West Virginia, the micromouse platform has the potential to serve as an ideal platform for robotics education in middle school and high school-aged children. The primary current program available to these students is FIRST Robotics. FIRST Robotics has demonstrated to be an excellent robotics program for students; however, the costs associated with the FIRST Robotics Program when compared to the micromouse platform are considerably higher. Costs based on materials, software, tools, and travel to FIRST Robotics competition are estimated around $15,000/team. The price of participation significantly reduces the ability of the FIRST Robotics Program to reach many potential engineering and scientists. The FIRST Tech Challenge Program provides a lower-cost solution than the FIRST Robotics Program; however, this program has not gained significant traction in West Virginia. Future research is recommended to pilot the micromouse platform to provide a low-cost, flexible robotics platform to support science, technology, engineering, and mathematics education.

References

[1] "FIRST," US FIRST, http://www.usfirst.org/roboticsprograms/resourcecenter.aspx?id=16935

[2] N. Chen, H. Chung, and Y. Kwon, "Integration of Micromouse Project with Undergraduate Curriculum: A Large-Scale Student Participation Approach," in *IEEE Transactions on Education:* 1995, Vol. 38.

[3] "FLL Team Resource," FIRST Lego League, http://www.usfirst.org/roboticsprograms/fll/content.aspx?id=14158, 2010.

[4] "CMU Robotics Education Academy," CMU, http://www.education.rec.ri.cmu.edu/, 2010.

[5] "Robotics Alliance Project Goals," NASA, http://robotics.nasa.gov/goals.php, 2010.

[6] "NURC," NURC, http://www.h2orobots.org/zindex.htm.

[7] "BEST Robotics," BEST Robotics, Inc, http://best.eng.auburn.edu/b_about_best.php.

[8] "The FIRST Lego League: Around the World," US FIRST Lego League, http://www.usfirst.org/uploadedFiles/Robotics_Programs/FLL/Communications_Resource_Center/Flyers/FLL_Growth_FNL.pdf, 2012.

[9] S. Garcia-Vergara, J. Pabon-DeLeon, and Y.Diaz-Mercado, Dr. E. Ortiz-Rivera, "An Integrated Undergraduate Research Experience in Control, Power Electronics, and Design Using a Micromouse," *IEEE* Washington, DC, USA, 2010, Session T3D.

[10] "Maze Solver Heats Maze," Micromouse UK, http://micromouse.cs.rhul.ac.uk/events/2000/mm2000/mazes.html, 2004.

[11] "Maze Solving Robot – AIRAT 2," Active Robots, http://www.active-robots.com/products/robots/maze-details-1.shtml.

[12] "UK Micromouse Site," Technology Innovation Centre, http://www.tic.ac.uk/micromouse/history.asp.

[13] "Temple University. Region 2 Student Activities Conference," Temple University IEEE Student Branch, College of Engineering, http://www.temple.edu/students/ieee/SAC/hotelinfo.html.

[14] "Micromouse Handbook," T. Auyeung, Ph.D., http://www.drtak.org/teaches/UCD/book/book/node1.html, 2003.

[15] "Micromouse Info," MicromouseInfo, http://www.micromouseinfo.com/introduction/mfloodfill.

[16] "Materials," Society of Robots, http://www.societyofrobots.com/materials.shtml.

[17] "TinyDragon Project Board," EVB Plus, http://www.evbplus.com/TinyDragon_9s12/TinyDragon_9s12.html.

[18] "DF12-BBU DragonflyBot Board with USB port," EVB Plus, http://www.evbplus.com/c32_modules/bbu_hc12_68hc12_9s12_hcs12.html.

[19] T. De, "The Inception of Chedda," in *Honors Thesis I*, UNLV, 2004.

[20] J. McCarty, "Spring 2009 Micromouse Project," in *Graduate Robotics*, WVU, 2009.

[21] "Sharp IR Ranger Information," Acroname Robotics, http://www.acroname.com/robotics/info/articles/sharp/sharp.html#e2.

[22] "GP2D120 Optoelectronic Device," Sharp, http://www.sharpsma.com/webfm_send/1205.

[23] "Robot Batteries," Society of Robots, http://www.societyofrobots.com/batteries.shtml.

[24] "Battery Type Comparison Chart," Rechargeable Batteries, http://www.rechargeablebatt.com/.

[25] S. Mishra and P. Bande, "Maze Solving Algorithms for Microm Mouse," in *IEEE Computer Society*, 2008, Vol. 2008.

[26] "Introduction to A*," stanford.edu, http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html#algorithms.

[27] "Micromouse Algorith," Robotix, http://www.robotix.in/rbtx09/tutorials/m4d4.

[28] "Dijkstra's Algorithm for Shortest Paths," CProgramming.com, http://www.cprogramming.com/tutorial/computersciencetheory/dijkstra.html.

[29] "Stepping Motors," Shinano Kenshi, http://www.shinano.com/stepping-motors.php.

[30] "Tenergy 8.4 V 1400 mAh Hi-Power NIHM Battery Pack", All-Battery.com, http://www.all-battery.com/84v1400mahhi-powernimhbatterypackforfirebirdrcplane.aspx.

[31] "Quadruple H-Bridge Driver," Texas Instruments, http://www.ti.com/lit/ds/symlink/sn754410.pdf, 2008.

[32] "Stepper Motors and Control," Stepperworld, http://www.stepperworld.com/Tutorials/pgBipolarTutorial.htm.

[33] H. Huang, “Parallel Ports,” in *The HCS12/9S12: An Introduction to Software & Hardware Interfacing, 1st* ed. New York: Thomas Delmar Learning, 2005, ch. *7,* section 12, pp. 312-319.

[34] H. Huang, “Timer Functions,” in *The HCS12/9S12: An Introduction to Software & Hardware Interfacing, 1st* ed. New York: Thomas Delmar Learning, 2005, ch. 8, section 10, pp. 379-392.

[34] G. McComb, “The Robot Builder’s Bonanza,” New York: McGraw-Hill, 2001.

[35] “Processor Expert”, Freescale Semiconductor, http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=PROCESSOR-EXPERT.

[36] A. Kyoto, “PID Control,” in *Control Systems, Robotics, and Automation,* Vol II.

[37] “Introduction: PID Controller Design”, University of Michigan, http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID#14.

[38] “Raspberry Pi FAQ,” Raspberry Pi Foundation, http://www.raspberrypi.org/faqs.

Appendix A: 2010 Region 2 Micromouse Rules [13]

The following rules will be observed for the Region 2 Micromouse competition which will take place at the Youngstown State University during the annual Region 2 Student Activities Conference. These rules are based on a set of rules used by Region 6, which in turn were adapted from the 1986 *Official Rules for North American Micromouse Contest*.

A. Objective

In this contest the contestant or team of contestants design and build an autonomous robotic "mouse" that negotiates a maze of standard dimensions from a specified corner to its center in the shortest time.

B. Contest Eligibility

1. All contestants must be an undergraduate IEEE student member at a Region 2 school from within the Area of Region 2 in which contest they will compete at the time of entry in the micromouse contest. Any student who graduates anytime during the Fall-Spring academic year in which the contest is held is eligible to enter the contest. A student graduating after competing in the contest still remains eligible to compete in succeeding Area, Region, and higher contests as an undergraduate student. Up to two graduate students per team are also allowed as stated in **Rule B.5** below, providing they meet all other requirements.
2. All contestants must be an IEEE Student Member or must have submitted an application for membership (and have it accepted by their Student Branch Counselor) prior to entry in the Student Branch and/or Chapter Contest.
3. The contestant(s) will make a brief presentation of their mouse design prior to the competition (5 minutes max) if time allows.
4. The micromouse entry may be the effort of an individual or a team. In the case of a team it should be possible to demonstrate that each individual made a significant contribution and that they are all IEEE members.

5. A team may consist of up to five people. A team of four or five people may include no more than two graduate students. A team of two or three people may have no more than one graduate student. A team consisting of a single graduate student is not allowed.
6. All entrants to the contest must declare their intention to enter the contest at least 4 weeks before the date of the regional contest. This notice must be submitted to the current Student Activities Coordinator, appropriate Area, Region 2, by mail, email, or phone (see the names and addresses at the end of this document). If the total number of declared mice, from all schools, is less than the number of eligible schools to compete in that Area, all shall be eligible to compete in the area contest. Two or more mice of near identical design from the same school are not allowed. If more mice than the number of eligible schools to compete are entered in the contest (ie., four mice from the same school), a qualifying competition will be held in the morning. A qualifying contest might involve, for example, having the mice transverse a specific numbers of cells.

C. Rules for the Micromouse

1. A micromouse shall be self-contained (no remote controls). A micromouse shall not use an energy source employing a combustion process.
2. A micromouse shall not leave any part of its body behind while negotiating the maze.
3. A micromouse shall not jump over, fly over, climb, scratch, cut, burn, mark, damage, or destroy the walls of the maze.
4. A micromouse shall not be larger either in length or in width, than 25 centimeters. The dimensions of a micromouse that changes its geometry during a run shall not be greater than 25 cm x 25 cm. There are no restrictions on the height of a micromouse.
5. Any violation of these rules will constitute immediate disqualification from the contest and ineligibility for the associated prizes.

D. Rules for the Maze

1. The maze is composed of multiples of an 18 cm x 18 cm unit square. The maze comprises 16 x 16 unit squares. The walls of the maze are 5 cm high and 1.2 cm thick (**assume 5% tolerance for mazes**). The outside wall encloses the entire maze.

2. The sides of the maze walls are white, the tops of the walls are red, and the floor is black. The maze is made of wood, finished with non-gloss paint.
    a. WARNING: Do not assume the walls are consistently white, or that the tops of the walls are consistently red, or that the floor is consistently black. Fading may occur; parts from different mazes may be used. Do not assume the floor provides a given amount of friction. It is simply painted plywood and may be quite slick. The maze floor may be constructed using multiple sheets of plywood. Therefore there may be a seam between the two sheets on which any low-hanging parts of a mouse may snag.
3. The start of the maze is located at one of the four corners. The start square is bounded on three sides by walls. The start line is located between the first and second squares. That is, as the mouse exits the corner square, the starts. The destination goal is the four cells at the center of the maze. At the center of this zone is a post, 20 cm high and each side 2.5 cm. (This post may be removed if requested.) The destination square has only one entrance.
4. Small square zones (posts), each 1.2 cm x 1.2 cm, at the four corners of each unit square are called lattice points. The maze is so constituted that there is at least one wall at each lattice point.
5. Multiple paths to the destination square are allowed and are to be expected. The destination square will be positioned so that a wall-hugging mouse will NOT be able to find it.

E. Rules for the Contest

1. Each contesting micromouse is allocated a total of 10 minutes of access to the maze from the moment the contest administrator acknowledges the contestant(s) and grants access to the maze. Any time used to adjust a mouse between runs is included in the 10 minutes. Each run (from the start cell to the center zone) in which a mouse successfully reaches the destination square is given a run time. The minimum run time shall be the mouse's official time. First prize goes to the mouse with the shortest official time. Second prize to the next shortest, and so on. **NOTE**, again, that the 10-minute timer continues even between runs. Mice that do not enter the center square will be ranked by the maximum

number of cells they consecutively transverse without being touched. All mice who enter the center square within their 10 minute allotment are ranked higher than those who do not enter the center square.

2. Each run shall be made from the starting square. The operator may abort a run at any time. If an operator touches the micromouse during a run, it is deemed aborted, and the mouse must be removed from the maze. If a mouse has already crossed the finish line, it may be removed at any time without affecting the run time of that run. If a mouse is placed back in the maze for another run, a one-time penalty of **30 seconds** will be added to the mouse's next run time.
3. After the maze is disclosed, the operator shall not feed information on the maze into the micromouse however, switch positions may be changed.
4. The illumination, temperature, and humidity of the room shall be those of an ambient environment. (40 to 120 degrees F, 0% to 95% humidity, noncondensing).
    a. **BEWARE**: Do not make any as unit square clockwise. The finish line is at the entrance to the destination square.
5. Every time the mouse leaves the start square, a new run begins. If the mouse has not entered the destination square, the previous run is aborted.
6. For example, if a mouse re-enters the start square (before entering the destination square) on a run, that run is aborted, and a new run will be deemed begun, with a new time that starts when the starting square is exited.
7. The mouse may, after reaching the destination square, continue to navigate the maze, for as long as their total maze time allows.
8. If a mouse continues to navigate the maze after reaching the destination square, the time taken will not count toward any run. Of course, the 10-minute timer continues to run. When the mouse next leaves the start square, a new run will start. Thus, a mouse may and should make several runs without being touched by the operator. It should make its own way back to the beginning to do so.
9. The judges reserve the right to ask the operator for an explanation of the micromouse. The judges also reserve the right to stop a run, declare disqualification, or give instructions as appropriate (e.g., if the structure of the maze is jeopardized by continuing operation of the mouse).

10. A contestant may not feed information on the maze to the micromouse. Therefore, changing ROMs or downloading programs is NOT allowed once the maze is revealed. **However, contestants are allowed to**:
    a. Change switch settings (e.g. to select algorithms)
    b. Replace batteries between runs
    c. Adjust sensors
    d. Change speed settings
    e. Make repairs
11. However, a contestant may not alter a mouse in a manner that alters its weight (e.g. removal of a bulky sensor array or switching to lighter batteries to get better speed after mapping the maze is not allowed). The judges shall arbitrate.
12. There is only one official IEEE micromouse contest each year in each Area or Region. All mice, whether or not they have competed in previous contests, compete on an equal basis. All mice must be presented to the judges by the original design team, which must meet all other qualifications. First prize will go to that mouse which travels from the start square to the destination square in the least amount of time. Second and third prizes will be awarded to the second and third fastest respectively.
13. As stated in **Rule E.1**, mice that do not enter the center square will be ranked by the maximum number of cells they consecutively transverse without being touched.
14. A rotating trophy is awarded to the first place mouse. Verbal recognition and certificates will be given to the top three mice among those who are competing for the first time. If you and your mouse are first-time contestants be sure to so stipulate when you register for the contest and notify the contest judge at the time of the contest.
15. If requested, a break will be provided for a mouse after any run if another mouse is waiting to compete. The 10-minute timer will stop. When the mouse is re-entered, the 10-minute timer will continue. The judges shall arbitrate on the granting of such breaks.

Appendix B: Dragonflybot Pin-outs

Table B.1 provides a list of the pin numbers and numbers utilized on the HCS12 microcontroller and to what it is connected to on the micromouse platform.

Table B.1: Dragonflybot board pin-outs

Pin Number	Pin Name	Connection
1	Gnd	
2	Vcc	
3	/Reset	
4	PT0	Motor 1 Enable 1
5	PT1	Motor 1 Enable 2
6	PT2	
7	PT3	
8	PT4	Motor 1 Orange
9	PT5	Motor 1 Red
10	PT6	
11	PT7	
12	PE1	
13	PE0	Motor 2 Enables
14	PE4	
15	PM0	Motor 1 Brown
16	PM1	Motor 1 Yellow
17	BKGN	
18	PE7	LCD Enable Bit
19	CAN_HIGH	
20	CAN_LOW	
40	PP5	
39	PM2	LCD Data Bit 0 / Motor 2 Brown
38	PM3	LCD Data Bit 1 / Motor 2 Yellow
37	PM4	LCD Data Bit 2 / Motor 2 Orange
36	PM5	LCD Data Bit 3 / Motor 2 Red
35	PS1/TXD	
34	PS0/RXD	
33	VRH	
32	AN7/PAD07	
31	AN6/PAD06	
30	AN5/PAD05	
29	AN4/PAD04	
28	AN3/PAD03	
27	AN2/PAD02	ADC Input (Right Sensor Input)
26	AN1/PAD01	ADC Input (Left Sensor Input)
25	AN0/PAD00	ADC Input (Front Sensor Input)
24	XTAL	CPU Clock Output Pin
23	EXTAL	CPU Clock Input Pin
22	PA0	LCD RS Bit
21	PB4	

Appendix C: Processor Expert – Software Components

The Freescale CodeWarrior software includes an integrated development environment that auto-generates source code for Freescale applications. The software components and descriptions of their utilization in the micromouse application are provided in Table C.1.

Table C.1: Processor Expert Software Components

Software Components	Method Prototype	Description
CPU	void Cpu_Delay100US (input)	Creates software delay. Length of delay is 100 microseconds * input parameter
ADC	byte Sensors_GetChanValue (byte channel, void *Values)	Reads ADC channel into Values variable and returns an error value.
ADC	byte Sensors_Measure (bool WaitforResult)	Performs one measurement on all channel
Timer	void PID_OnInterrupt(void)	Instantiates PID controller code every 55 milliseconds
Timer	Motor1_Timer1_OnInterrupt	Moves Motor 1 forward or backward
Timer	Motor1_Timer2_OnInterrupt	Toggles Motor 1 output to reduce effective current
Timer	Motor2_Timer1_OnInterrupt	Moves Motor 2 forward or backward
Timer	Motor2_Timer2_OnInterrupt	Toggles Motor 2 output to reduce effective current
Timer	Motor1_Timer1_SetPeriodUS	Sets the period of the timer interrupt for Motor 1
Timer	Motor2_Timer1_SetPeriodUS	Sets the period of the timer interrupt for Motor 2
Timer	void PID_OnInterrupt(void)	Instantiates PID controller code every 55 milliseconds
Timer	Motor1_Timer1_OnInterrupt	Moves Motor 1 forward or backward
Timer	Motor1_Timer2_OnInterrupt	Toggles Motor 1 output to reduce effective current
Timer	Motor2_Timer1_OnInterrupt	Moves Motor 2 forward or backward
Timer	Motor2_Timer2_OnInterrupt	Toggles Motor 2 output to reduce effective current
Timer	Motor1_Timer1_SetPeriodUS	Sets the period of the timer interrupt for Motor 1
Timer	Motor2_Timer1_SetPeriodUS	Sets the period of the timer interrupt for Motor 2

Appendix D: Micromouse Software

Appendix D includes a copy of the source code of all major software functions.

Appendix D.1: Main Program (main.c)

```
/** ###################################################################
**     Filename  : Micromouse.C
**     Project   : Micromouse
**     Processor : MC9S12C32MFA25
**     Version   : Driver 01.13
**     Compiler  : CodeWarrior HC12 C Compiler
**     Date/Time : 1/5/2010, 11:33 PM
**     Abstract  :
**         Main module.
**         This module contains user's application code.
** ###################################################################*/

/* MODULE Micromouse */

/* Including used modules for compiling procedure */
#include "Cpu.h"
#include "Events.h"
#include "PID.h"
#include "Sensors.h"
#include "Motor2_Enable1234.h"
#include "Motor1_Timer1.h"
#include "Motor1_Timer2.h"
#include "Motor2_Timer1.h"
#include "Motor2_Timer2.h"
#include "Motor1_Enable12.h"
#include "Motor1_Enable34.h"
#include "Motor1_Brown.h"
#include "Motor1_Orange.h"
#include "Motor1_Red.h"
#include "Motor1_Yellow.h"
#include "DataBits.h"
#include "ENBit.h"
#include "RSBit.h"
/* Include user modules */
#include "LCD.h"
#include "Motor.h"
#include "Maze.h"
/* Include shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"

//*****************************************************************
// Motor 1 Global Variables
// Wired Motor1 - Brown (A)
// Databits = M0, M1, T4, T5 and Enable12=Port T0 and Enable34=Port T1
```

```
// steps1 = keeps track of the number of steps for Motor1
// direction1:  0-forward and 1-reverse
//*********************************************************************

char Motor1Step[NUM_OF_STATES] =
      {0x01, 0x09, 0x08, 0x0C, 0x04, 0x06, 0x02, 0x03};
volatile char next_state1 = 0;
volatile bool toggle1 = 0;
unsigned int steps1 = 0;
volatile bool direction1 = 0;

//*********************************************************************
// Motor 2 Global Variables
// Wired Motor5 - Brown (A), Motor6 -
// Databits = M2, M3, M4, M5 and Enable12=Port T2 and Enable34=Port T3
// steps2 = keeps track of the number of steps for Motor2
// direction2:  0-forward and 1-reverse
//*********************************************************************

char Motor2Step[NUM_OF_STATES] =
      {0x01, 0x09, 0x08, 0x0C, 0x04, 0x06, 0x02, 0x03};
volatile char next_state2 = 0;
volatile bool toggle2 = 0;
unsigned int steps2 = 0;
volatile bool direction2 = 0;

//*********************************************************************
// Sensor values
//*********************************************************************

word valueR, valueL, valueF;

//*********************************************************************
// Micromouse Position Variables
//*********************************************************************

byte currentR, currentC;
byte lastR, lastC;

byte walls[ROWS*COLS];
byte maze[ROWS*COLS];

//*********************************************************************
// To change Motor Timer Values based on PID
//*********************************************************************

volatile word timer1, timer2;
volatile byte Motor1, Motor2;
volatile int rev1, rev2;

//*********************************************************************
// PID Global Variables
//*********************************************************************
volatile byte reset = 0;
```

```
//*********************************************************************
// Main Program
//*********************************************************************

void main(void)
{
  /* Write your local variable definition here */
  byte Error = 0;
  bool initial = 0;
  byte value = 0;
  byte lowValue = 0x00;
  byte direction = 1;
  byte newdirection = 1;
  int i = 0;

  /* Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! */
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.***/

//*********************************************************************
// Initialize LCD
//*********************************************************************
  LCD_Init();

//*********************************************************************
// Implementation of Modified Flood Fill Algorithm requires two arrays
// 1. Flood-fill Numbers - stored in maze[] array
// 2. Wall Information - stored in walls[] array
//*********************************************************************

  InitializeMaze();
  SetUpWalls();

//*********************************************************************
// Initialize currentC and currentR values
//*********************************************************************

  currentC = 0;
  currentR = 0;

  reset = 0;

  for (;;){
    MoveForward(400);
    Error = Sensors_Measure(1);
    Error = Sensors_GetChanValue(0, &valueL);
    Error = Sensors_GetChanValue(1, &valueF);
    Error = Sensors_GetChanValue(2, &valueR);
    display_pulseWidths(valueL, valueR);

    // Update wall map
    UpdateWallMap(direction);
    display_pulseWidths(valueF, valueR);

    // Update distance values as necessary
    UpdateMaze(currentR, currentC);
```

```
    // Store value of current cell
    lastR = currentR;
    lastC = currentC;

    // Determine which neighbor has lowest value
    // lowValue = upper nibble=row, lower nibble=col
    lowValue = NextMove(currentR, currentC);

    // Update position based on NextMove function results
    currentC = (lowValue & 0x0F);
    currentR = ((lowValue & 0xF0) >> 4);

    // Move to the new current cell
    newdirection = MoveMouse(direction);
    direction = newdirection;

    Cpu_Delay100US(5000);
 }
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
  /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/
}
```

Appendix D.2: Interrupt Routines (events.c)

```
/** ###################################################################
**     Filename  : Events.C
**     Project   : Micromouse
**     Processor : MC9S12C32MFA25
**     Beantype  : Events
**     Version   : Driver 01.04
**     Compiler  : CodeWarrior HC12 C Compiler
**     Date/Time : 1/5/2010, 11:33 PM
**     Abstract  :
**         This is user's event module.
**         Put your event handler code here.
** ###################################################################*/

/* MODULE Events */

#include "Cpu.h"
#include "Events.h"
#include "Motor.h"
#include "float.h"

#pragma CODE_SEG DEFAULT

extern char Motor1Step[];
extern volatile char next_state1;
extern volatile bool toggle1;
extern unsigned int steps1;
extern volatile bool direction1;

extern char Motor2Step[];
extern volatile char next_state2;
extern volatile bool toggle2;
extern unsigned int steps2;
extern volatile bool direction2;

extern word valueR, valueL;
volatile word Motor1_Timer = 5000, Motor2_Timer = 5000;

extern volatile byte reset;

//*****************************************************************
// Motor 1 Interrupt Service Routine - Step Motor 1
//*****************************************************************
void Motor1_Timer1_OnInterrupt(void)
{
  byte Error = 0;
  if (next_state1 > 7) {
    next_state1 = 0;
  }
  if (next_state1 < 0){
    next_state1 = 7;
  }

  Motor1_WriteBits(Motor1Step[next_state1]);
```

```
  if (direction1 == 0){
    next_state1++;
  }
  else {
    next_state1--;
  }

  steps1++;
}

//*********************************************************************
// Motor 2 Interrupt Service Routine - Step Motor 2
//*********************************************************************
void Motor2_Timer1_OnInterrupt(void)
{
  if (next_state2 > 7) {
    next_state2 = 0;
  }
  if (next_state2 < 0){
    next_state2 = 7;
  }

  DataBits_PutVal(Motor2Step[next_state2]);
  if (direction2 == 0){
    next_state2++;
  }
  else {
    next_state2--;
  }
  steps2++;
}

//*********************************************************************
// Motor 1 Interrupt Service Routine - Pulse to reduce effective current
//*********************************************************************
void Motor1_Timer2_OnInterrupt(void)
{
  if (toggle1){
    Motor1_WriteBits(0);
    toggle1 = 0;
  }
  else {
    Motor1_WriteBits(Motor1Step[next_state1]);
    toggle1 = 1;
  }
}

//*********************************************************************
// Motor 2 Interrupt Service Routine - Pulse to reduce effective current
//*********************************************************************
void Motor2_Timer2_OnInterrupt(void)
{
  if (toggle2){
    Motor2_WriteBits(0);
    toggle2 = 0;
  }
  else {
```

```
    Motor2_WriteBits(Motor2Step[next_state2]);
    toggle2 = 1;
  }
}

//*****************************************************************
// Event=PID_OnInterrupt (module Events)
// PID Interrupt Service Routine (P Controller Implementation)
//*****************************************************************
void PID_OnInterrupt(void)
{
 static int olderrorL = 0, olderrorR = 0;
 float Kp = 0.625;
 int error = 0;
 word setpoint = 400;
 float correct;
 float P=0, D=0;

  // Left Sensor Value
  error = setpoint - valueL;

  P = error*Kp;


  correct = P;

  Motor1_Timer += correct;

  if (Motor1_Timer >= 5000)
    Motor1_Timer = 5000;
  if (Motor1_Timer <= 2000)
    Motor1_Timer = 2000;

  Motor1_Timer1_SetPeriodUS(Motor1_Timer);

  olderrorL = error;

  // Right Sensor Value
  error = setpoint - valueR;

  P = error*Kp;
  //D = (error - olderrorR)*Kd;

  correct = P;

  Motor2_Timer += correct;
  if (Motor2_Timer >= 5000)
    Motor2_Timer = 5000;
  if (Motor2_Timer <= 2000)
    Motor2_Timer = 2000;

  Motor2_Timer1_SetPeriodUS(Motor2_Timer);
  olderrorR = error;
  reset = 0;

}
```

Appendix D.3: Motor Functionality (motor.c)

```
/** ###################################################################
**     Filename  : Motor.C
**     Project   : Project_44
**     Processor : MC9S12C32CFA25
**     Compiler  : CodeWarrior HC12 C Compiler
**     Date/Time : 10/24/2009, 10:39 PM
**     Contents  :
**         User source code
** ###################################################################*/

/* MODULE Motor */
#include "Cpu.h"
#include "Events.h"
#include "Motor.h"
#include "Maze.h"
#include "LCD.h"
#include "Motor1_Brown.h"
#include "Motor1_Orange.h"
#include "Motor1_Red.h"
#include "Motor1_Yellow.h"
#include "DataBits.h"
#include "float.h"
#include "math.h"
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"

extern volatile unsigned int steps1;
extern volatile unsigned int steps2;
extern volatile bool direction1;
extern volatile bool direction2;

extern word valueR, valueL, valueF;
extern byte currentR, currentC;
extern byte lastR, lastC;

extern volatile byte Motor1, Motor2;
extern volatile word timer1, timer2;
extern volatile int rev1, rev2;

//*****************************************************************
// Motor 1 - Function to drive stepper motor 1 pins
//*****************************************************************
void Motor1_WriteBits (byte value) {

  // Write values to appropriate pins
  Motor1_Brown_PutVal ((value & 0x08) && 0x08);
  Motor1_Yellow_PutVal((value & 0x04) && 0x04);
  Motor1_Orange_PutVal((value & 0x02) && 0x02);
  Motor1_Red_PutVal   ((value & 0x01) && 0x01);

}
```

```
//*********************************************************************
// Motor 2 - Function to drive stepper motor 2 pins
//*********************************************************************
void Motor2_WriteBits (byte value){
  DataBits_PutBit(3, (value & 0x08) && 0x08);
  DataBits_PutBit(2, (value & 0x04) && 0x04);
  DataBits_PutBit(1, (value & 0x02) && 0x02);
  DataBits_PutBit(0, (value & 0x01) && 0x01);
}

//*********************************************************************
// Move the micromouse forward by number of steps
//*********************************************************************
void MoveForward (unsigned int steps){
  // 400 steps = 1 rotation
  // 1.8 degree / step = 200 steps and 400 1/2 steps
  direction1 = 0;
  direction2 = 0;

  StartMotors();
  while ((steps1 < steps) && (steps2 < steps)){} // Do nothing

  StopMotors();
  steps1 = 0;
  steps2 = 0;
}

//*********************************************************************
// Move the micromouse backward by number of steps
//*********************************************************************
void MoveBackward (unsigned int steps) {
  direction1 = 1;
  direction2 = 1;

  StartMotors();
  while ((steps1 < steps) && (steps2 < steps)){} // Do nothing

  StopMotors();
  steps1 = 0;
  steps2 = 0;

}

//*********************************************************************
// Turn the micromouse to the left
//*********************************************************************
void MoveLeft () {
  direction1 = 1;
  direction2 = 0;

  StartMotors();
  while ((steps1 < 200) && (steps2 < 200)){} // Do nothing

  StopMotors();
  steps1 = 0;
  steps2 = 0;
}
```

```
//*********************************************************************
// Turn the micromouse to the right
//*********************************************************************
void MoveRight (){
  direction1 = 0;
  direction2 = 1;

  StartMotors();
  while ((steps1 < 200) && (steps2 < 200)){} // Do nothing

  StopMotors();
  steps1 = 0;
  steps2 = 0;
}

//*********************************************************************
// Turn the micromouse around
//*********************************************************************
void TurnAround (){
  direction1 = 0;
  direction2 = 1;

  StartMotors();
  while ((steps1 < 315) && (steps2 < 315)){} // Do nothing

  StopMotors();
  steps1 = 0;
  steps2 = 0;
}

//*********************************************************************
// Enable both stepper motors
//*********************************************************************

void StartMotors(){
  Motor1_Timer1_EnableEvent();
  Motor2_Timer1_EnableEvent();

  // This was the PWM code that was used to reduce motor current
   Motor1_Timer2_EnableEvent();
   Motor2_Timer2_EnableEvent();
}

//*********************************************************************
// Stop/Disable both stepper motors
//*********************************************************************
void StopMotors (){
  Motor1_Timer1_DisableEvent();
  Motor2_Timer1_DisableEvent();

  // This was the PWM code that was used to reduce motor current
   Motor1_Timer2_DisableEvent();
   Motor2_Timer2_DisableEvent();
}
```

```
//*********************************************************************
// Moves the mouse to the next cell based on its current facing direction
//*********************************************************************
byte MoveMouse (byte direction){
  byte forwardCount=0;
  unsigned int onecell = 378;

  switch(direction){
    case NORTH:
    if (currentR < lastR){
      MoveLeft();
      forwardCount = 0;
      direction = WEST;
      writeData4('1');
    }
    else if (currentR > lastR){
      MoveRight();
      forwardCount = 0;
      direction = EAST;
      writeData4('2');
    }
    else if (currentC < lastC){
      TurnAround();
      direction = SOUTH;
      writeData4('3');
    }
    else{
      forwardCount++;
      writeData4('4');
    }
    break;

    case EAST:
      if (currentC > lastC){
        MoveLeft();
        forwardCount = 0;
        direction = NORTH;
      }
      else if (currentC < lastC){
        MoveRight();
        forwardCount = 0;
        direction = SOUTH;
      }
      else if ( currentR < lastR){
        TurnAround();
        direction = WEST;
      }
      else{
        forwardCount++;
      }
      break;

    case SOUTH:
      if (currentR > lastR){
        MoveLeft();
        forwardCount = 0;
        direction = EAST;
```

```
      }
      else if (currentR < lastR){
        MoveRight();
        forwardCount = 0;
        direction = WEST;
      }
      else if (currentC > lastC){
        TurnAround();
        direction = NORTH;
      }
      else{
        forwardCount++;
      }
      break;

    case WEST:
      if (currentC < lastC){
        MoveLeft();
        forwardCount = 0;
        direction = SOUTH;
      }
      else if ( currentC > lastC){
        MoveRight();
        forwardCount = 0;
        direction = NORTH;
      }
      else if ( currentR > lastR){
        TurnAround();
        direction = EAST;
      }
      else{
        forwardCount++;
      }
      break;
  }

  Cpu_Delay100US(2000);
  MoveForward(onecell);
  return direction;
}
```

Appendix D.4: Maze Solving Algorithm (maze.c)

```
/** ###################################################################
**     Filename  : Maze.C
**     Project   : Project_44
**     Processor : MC9S12C32CFA25
**     Compiler  : CodeWarrior HC12 C Compiler
**     Date/Time : 10/25/2009, 8:40 PM
**     Contents  :
**         User source code
** ###################################################################*/

#include "Cpu.h"
#include "Events.h"
#include "Maze.h"
#include "float.h"
#include "math.h"
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"

extern byte walls[ROWS*COLS];
extern byte maze[ROWS*COLS];
extern byte currentR, currentC;
extern word valueL, valueR, valueF;


//*****************************************************************
// Initializes maze values --> Modified Flood Fill Algorithm
// Puts values in maze[] array
// Example of 6x13 array
// maze[] = 7 6 5 4 3 2 2 2 3 4 5 6 7
//          6 5 4 3 2 1 1 1 2 3 4 5 6
//          5 4 3 2 1 0 0 0 1 2 3 4 5
//          5 4 3 2 1 0 0 0 1 2 3 4 5
//          6 5 4 3 2 1 1 1 2 3 4 5 6
//          7 6 5 4 3 2 2 2 3 4 5 6 7
//*****************************************************************
void InitializeMaze()
{
      byte i,j;

      for (i=0; i< COLS; i++)
            for (j=0; j< ROWS; j++)
            {
                  if (((i == COLS/2) || (i == COLS/2 -1)) && (((j == ROWS/2)
                  || (j == ROWS/2 -1))))
                              maze[j*COLS + i] = (byte)0;
                  else if (i <= COLS/2 -1)
                  {
                        if (j <= ROWS/2 -1)
                              maze[j*COLS + i] = (byte)(fabs(((COLS/2 -1) -
                              i)) + fabs(((ROWS/2 -1) - j)));
                        else
```

```
                              maze[j*COLS + i] = (byte)(fabs(((COLS/2 -1) -
                              i)) + fabs(((ROWS/2) - j)));
                  }
                  else
                  {
                        if (j <= ROWS/2 -1)
                              maze[j*COLS + i] = (byte)(fabs(((COLS/2) - i))
                              + fabs(((ROWS/2 -1) - j)));
                        else
                              maze[j*COLS + i] = (byte)(fabs(((COLS/2) - i))
                              + fabs(((ROWS/2 ) - j)));
                  }
            } // close for


}


//*****************************************************************
// Sets up the return maze path as follows
// maze[] = 0 1 2 3 4  5  6  7  8  9  10 11 12
//          1 2 3 4 5  6  7  8  9  10 11 12 13
//          2 3 4 5 6  7  8  9 10  11 12 13 14
//          3 4 5 6 7  8  9 10 11  12 13 14 15
//          4 5 6 7 8  9 10 11 12  13 14 15 16
//          5 6 7 8 9 10 11 12 13  14 15 16 17
//*****************************************************************
void SetUpReturn(int currentRow, int CurrentCol, int StartRow, int StartCol)
{
      int i, j;
      for (i=0; i< COLS; i++)
            for (j=0; j< ROWS; j++)
            {
                  if ((i == StartCol) && (j == StartRow))
                        maze[j*COLS + i] = (byte)0;
                  else
                  {
                        maze[j*COLS + i] = (byte)(i+j);
                  }
            }
}


//*****************************************************************
// Update Maze
// Function updates the maze's distance values - stores in maze[] array
//    Pull a cell from the stack
//    Is the distance value of this cell = 1 + the minimum value of its open
      neighbors?
//    No -> Change the cell to 1 + the minimum value of its open neighbors
      and
//    push all of the cell's open neighbors onto the stack to be checked
//    Yes -> Do nothing
// Example:
// If no wall to the South
//   If south's cells "d" value is less than current cell's value
//     Yes, this is OK
```

```
//     No, this needs fixed
//*********************************************************************
void UpdateMaze(byte r, byte c)
{
      byte toFix[(ROWS*COLS)];
      int pointer=1;
      bool fixed, change;
      byte element;
      byte val;

      toFix[pointer-1] = (((byte)r << 4) | (byte)c);

      while (pointer > 0)                              // Outer Loop
      {
            element = toFix[pointer-1];
            r = (byte)(element >> 4);
            c = (byte)(element & 0x0F);
            pointer--;
            fixed = 0;
            change = 0;
            val = maze[r*COLS+c];
            if (val != 0)  // Haven't reached the center
            {
                  do
                  {
                        if((walls[r*COLS+c] & 0x01)  == 0 && c > 0)
                        {
                              if(maze[r*COLS + c-1] < val)
                                          fixed = 1;
                        }
                        if((walls[r*COLS+c] & 0x02)  == 0 && r > 0)
                        {
                              if(maze[(r-1)*COLS + c] < val)
                                    fixed = 1;
                        }
                        if((walls[r*COLS+c] & 0x04)  == 0 && c < COLS -1)
                        {
                              if(maze[r*COLS + c+1] < val)
                                    fixed = 1;
                        }
                        if((walls[r*COLS+c] & 0x08)  == 0 && r < ROWS -1)
                        {
                              if(maze[(r+1)*COLS + c] < val)
                                    fixed = 1;
                        }

                        if(!fixed)
                        {
                              val+=1;
                              change = 1;
                        }

                }
                  while (!fixed);

                  maze[r*COLS+c] = val;
```

```
                  if (change)
                  {
                        if((walls[r*COLS+c] & 0x01) == 0 && c > 0)
                        {
                              pointer++;
                              toFix[pointer-1] = (((byte)r << 4) | (byte)(c-
                              1));
                        }
                        if((walls[r*COLS+c] & 0x02)  == 0 && r > 0)
                        {
                              pointer++;
                              toFix[pointer-1] = (((byte)(r-1) << 4) |
                              (byte)c);
                        }
                        if((walls[r*COLS+c] & 0x04)  == 0 && c < COLS -1)
                        {
                              pointer++;
                              toFix[pointer-1] = (((byte)r << 4) |
                              (byte)(c+1));
                        }
                        if((walls[r*COLS+c] & 0x08)  == 0 && r < ROWS -1)
                        {
                              pointer++;
                              toFix[pointer-1] = (((byte)(r+1) << 4) |
                              (byte)c);
                        }
                  }
            }
      }
}


//*****************************************************************
// Function sets up the number of walls in the maze - puts in walls[]
// walls[] = 3 2 2 2 2 2 2 2 2 2 2 2  6
//           1 0 0 0 0 0 0 0 0 0 0 0  4
//           1 0 0 0 0 0 0 0 0 0 0 0  4
//           1 0 0 0 0 0 0 0 0 0 0 0  4
//           1 0 0 0 0 0 0 0 0 0 0 0  4
//           9 8 8 8 8 8 8 8 8 8 8 8 12
//*****************************************************************
void SetUpWalls()
{
  byte r=0,c=0;
      byte wls;

      for(r=0; r < 6; r++)
      {

        for(c =0; c < 13; c++)
        {
          wls = 0x00;
      if (r == 0)
      {
          wls = (wls | 0x02);
      }
      if (c == 0)
```

```
      {
          wls = (wls | 0x01);
      }
      if (r == ROWS - 1)
      {
          wls = (wls | 0x08);
      }
      if (c == COLS - 1)
      {
          wls = (wls | 0x04);
      }
      walls[r * COLS + c] = wls;
        }
      }

}


//*****************************************************************
// Updates Maze Values to Robot's New Position within the Maze
// Checks all of the potential next moves to determine where to move next
// Checks:  row+1, row-1, col-1, col+1
// Return value is the cell of the next move
// upper nibble is the row value; lower nibble is the column value
//*****************************************************************
byte NextMove(byte currentRow, byte CurrentCol)
{
      byte move =0;
      byte r = ROWS+1,c = COLS+1;
      byte value = maze[currentRow*COLS + CurrentCol];
      byte next;

      //Check Row+1
      if (currentRow < ROWS -1)
            if (maze[(currentRow+1)*COLS + CurrentCol] < value &&
               (walls[(currentRow)*COLS + CurrentCol] & 0x8) == 0x00)
            {
                  r = currentRow+1;
                  c = CurrentCol;
                  value = maze[(currentRow+1)*COLS + CurrentCol];
            }
      //Check Row-1
      if (currentRow > 0)
            if (maze[(currentRow-1)*COLS + CurrentCol] < value &&
               (walls[(currentRow)*COLS + CurrentCol] & 0x2) == 0x00)
            {
                  r = currentRow-1;
                  c = CurrentCol;
                  value = maze[(currentRow-1)*COLS + CurrentCol];
            }
      //Check Column+1
      if (CurrentCol < COLS -1)
            if (maze[(currentRow)*COLS + CurrentCol+1] < value &&
               (walls[(currentRow)*COLS + CurrentCol] & 0x4) == 0x00)
            {
                  r = currentRow;
                  c = CurrentCol+1;
```

```
                  value = maze[(currentRow)*COLS + CurrentCol+1];
            }
      //Check Column-1
      if (CurrentCol > 0)
            if (maze[(currentRow)*COLS + CurrentCol-1] < value &&
               (walls[(currentRow)*COLS + CurrentCol] & 0x1) == 0x00)
            {
                  r = currentRow;
                  c = CurrentCol-1;
                  value = maze[(currentRow)*COLS + CurrentCol-1];
            }

            //upper half will be row, lower col
            next = ((byte)r << 4) | (byte)c;

            return next;
}


//*****************************************************************
// Insert Wall -
// walls matrix when called
//*****************************************************************
void InsertWall(byte wallSide)
{
   walls[(int)(currentR*COLS + currentC)] = walls[(int)(currentR*COLS +
   currentC)] | wallSide;

   /*switch (wallSide)
   {
    case 1:
      if (currentC > 0)
        walls[(int)((currentR)*cols + (currentC-1))] =
        walls[(int)((currentR)*cols + (currentC-1))] | 0x04;
      break;
    case 2:
      if (currentR > 0)
        walls[(int)((currentR-1)*cols + currentC)] = walls[(int)((currentR-
        1)*cols + currentC)] | 0x08;
      break;
    case 4:
      if (currentC < cols -1)
        walls[(int)((currentR)*cols + (currentC+1))] =
        walls[(int)((currentR)*cols + (currentC+1))] | 0x01;
      break;
    case 8:
      if (currentR < rows -1)
        walls[(int)((currentR+1)*cols + (currentC))] =
        walls[(int)((currentR+1)*cols + (currentC))] | 0x02;
      break;
   } */
}


//*****************************************************************
// UpdateWallMap
// Updates the wall map based on the direction the mouse is facing
```

```
// NORTH=1, EAST=2, SOUTH=3, WEST=4
//*********************************************************************
void UpdateWallMap (byte direction){
    switch (direction){
      case NORTH:
       if(valueL > WALLSENSE)
          {
            InsertWall(0x02);
          }
       if(valueR > WALLSENSE)
          {
            InsertWall(0x08);
          }
       if(valueF > FRONTSENSE)
          {
            InsertWall(0x04);
          }
       break;
      case EAST:
        if(valueL > WALLSENSE)
          {
            InsertWall(0x04);
          }
        if(valueR > WALLSENSE)
          {
            InsertWall(0x01);
          }
        if(valueF > FRONTSENSE)
          {
            InsertWall(0x08);
          }
        break;
      case SOUTH:
        if(valueL > WALLSENSE)
          {
            InsertWall(0x08);
          }
        if(valueR > WALLSENSE)
          {
            InsertWall(0x02);
          }
        if(valueF > FRONTSENSE)
          {
            InsertWall(0x01);
          }
        break;
      case WEST:
        if(valueL > WALLSENSE)
          {
            InsertWall(0x01);
          }
        if(valueR > WALLSENSE)
          {
            InsertWall(0x04);
          }
        if(valueF > FRONTSENSE)
          {
```

```
            InsertWall(0x02);
          }
        break;

    }  // End Switch Statement
}


//*********************************************************************
// UpdateAll calls updateMaze in order to loop through all the elements as
appropriate
//*********************************************************************

void UpdateAll()
{
      byte i, j;
      for (i=0; i < COLS; i++)
            for (j=0; j < ROWS; j++)
            {
                  UpdateMaze(j,i);
            } // close for
}
```

Appendix D.5: LCD Initialization (lcd.c)

```
/** ###################################################################
**     Filename  : LCD.C
**     Project   : Project_44
**     Processor : MC9S12C32CFA25
**     Compiler  : CodeWarrior HC12 C Compiler
**     Date/Time : 10/25/2009, 12:13 PM
**     (c) Copyright UNIS, spol. s r.o. 1997-2008
** ###################################################################*/

/* MODULE LCD */
#include "Cpu.h"
#include "Events.h"
#include "DataBits.h"
#include "ENBit.h"
#include "RSBit.h"
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"

byte out_hundreds, out_tens, out_ones = 0x00;
byte final_out_hundreds, final_out_tens, final_out_ones;


//*****************************************************************
// This routine sends a command to the LCD Module
// The routine operates in 8-bit mode.  The upper nibble is
// sent, and the lower nibble is ignored.  For commands, the
// RS bit is always set equal to 0
//*****************************************************************
void writeCom8(byte command){

  // Start of Command Sequence
  // Transmit upper nibble with Enable = 0
  ENBit_PutVal(0);
  RSBit_PutVal(0);
  DataBits_PutVal(command >> 4);

  // Write Command
  // Transmit upper nibble with Enable = 1
  ENBit_PutVal(1);
  RSBit_PutVal(0);
  DataBits_PutVal(command >> 4);

  // End of Command Sequence
  // Transmit upper nibble with Enable = 0
  ENBit_PutVal(0);
  RSBit_PutVal(0);
  DataBits_PutVal(command >> 4);

  Cpu_Delay100US(1);
}
```

```
//*********************************************************************
// This routine sends a command to the LCD Module
//
// The routine operates in 4-bit mode.  The upper nibble is
// sent, then the lower nibble is sent.  For commands, the
// RS bit is always set equal to 0
//*********************************************************************
void writeCom4(byte command){

  // Start of Command Sequence (Upper Nibble)
  // Transmit upper nibble with Enable = 0
  ENBit_PutVal(0);
  RSBit_PutVal(0);
  DataBits_PutVal(command >> 4);

  // Write Command (Upper Nibble)
  // Transmit upper nibble with Enable = 1
  ENBit_PutVal(1);
  RSBit_PutVal(0);
  DataBits_PutVal(command >> 4);

  // End of Command Sequence (Upper Nibble)
  // Transmit upper nibble with Enable = 0
  ENBit_PutVal(0);
  RSBit_PutVal(0);
  DataBits_PutVal(command >> 4);

  // Start of Command Sequence (Lower Nibble)
  // Transmit lower nibble with Enable = 0
  ENBit_PutVal(0);
  RSBit_PutVal(0);
  DataBits_PutVal(command);

  // Write Command (Lower Nibble)
  // Transmit lower nibble with Enable = 1
  ENBit_PutVal(1);
  RSBit_PutVal(0);
  DataBits_PutVal(command);

  // End of Command Sequence (Lower Nibble)
  // Transmit lower nibble with Enable = 0
  ENBit_PutVal(0);
  RSBit_PutVal(0);
  DataBits_PutVal(command);

  Cpu_Delay100US(1);

}

//*********************************************************************
// This routine sends a single character to the LCD Display
//
// The routine operates in 4-bit mode.  The upper nibble is
// sent, then the lower nibble is sent.  For data, the RS bit
// is always set equal to 1
//*********************************************************************
void writeData4(byte data){
```

```
  // Start of Data Sequence (Upper Nibble)
  // Transmit upper nibble with Enable bit = 0
  ENBit_PutVal(0);
  RSBit_PutVal(1);
  DataBits_PutVal(data >> 4);

  // Write Data (Upper Nibble)
  // Transmit upper nibble with Enable bit = 1
  ENBit_PutVal(1);
  RSBit_PutVal(1);
  DataBits_PutVal(data >> 4);

  // End of Data Sequence (Upper Nibble)
  // Transmit upper nibble with Enable bit = 0
  ENBit_PutVal(0);
  RSBit_PutVal(1);
  DataBits_PutVal(data >> 4);


  // Start of Data Sequence (Lower Nibble)
  // Transmit lower nibble with Enable bit = 0
  ENBit_PutVal(0);
  RSBit_PutVal(1);
  DataBits_PutVal(data);

  // Write Data (Lower Nibble)
  // Transmit lower nibble with Enable bit = 1
  ENBit_PutVal(1);
  RSBit_PutVal(1);
  DataBits_PutVal(data);

  // End of Data Sequence (Lower Nibble)
  // Transmit lower nibble with Enable bit = 0
  ENBit_PutVal(0);
  RSBit_PutVal(1);
  DataBits_PutVal(data);

  Cpu_Delay100US(1);
}

//*****************************************************************
// This routine initializes the 8x2 LCD Display
//
// The routine begins in 8-bit Mode, and requires the command
// 0x30 to be sent three consecutive times before setting up
// the LCD.  After communication is established, the routine
// switches over to 4-bit mode to continue setting up the LCD
//*****************************************************************
void LCD_Init(void){

  // Wait 20ms for initial Power-On (15ms minimum)
  Cpu_Delay100US(200);

  // COMMAND #1
  // Send 0x30 the first time (8-bit Mode)
  writeCom8(0x30);
```

```
  // Wait 5 ms (4.1 ms minimum)
  Cpu_Delay100US(50);

  // COMMAND #2
  // Send 0x30 the second time (8-bit Mode)
  writeCom8(0x30);

  // Wait 200 us (100us minimum)
  Cpu_Delay100US(2);

  // COMMAND #3
  // Send 0x30 the third time (8-bit Mode)
  writeCom8(0x30);

  // COMMAND #4: Switches to 4-bit Mode w/ Function Set Command
  // Send 0x20 (Still in 8-bit Mode)
  writeCom8(0x20);

  // **********  NOW IN 4-BIT MODE  **********
  // COMMAND #5: FUNCTION SET -> [0 0 1 DL N$ RE/F 0 #]
  //                   Result:   [0 0 1 0 1 0 0 0]  = 0x28
  // Write 0x28 (Now in 4-bit Mode: Send upper nibble then lower nibble)
  writeCom4(0x28);

  // COMMAND #6: DISPLAY ON ->  [0 0 0 0 1 D C B]
  //                   Result:  [0 0 0 0 1 1 1 1]  = 0x0F
  //
  //Write 0x0F (In 4-bit Mode: Send upper nibble then lower nibble)
  writeCom4(0b00001100);

  // COMMAND #7: DISPLAY CLEAR ->  [0 0 0 0 0 0 0 1]
  //                     Result:   [0 0 0 0 0 0 0 1]  = 0x01
  //
  // Write 0x01 (In 4-bit Mode: Send upper nibble then lower nibble)
  writeCom4(0x01);
  // Additional delay required when clearing screen & returning home
  Cpu_Delay100US(35);
  // COMMAND #8: ENTRY MODE SET ->  [0 0 0 0 0 1 I/D S]
  //                     Result:    [0 0 0 0 0 1 1 0]  = 0x06
  //
  // Write 0x06 (In 4-bit Mode: Send upper nibble then lower nibble)
  writeCom4(0x06);

  // Delay to allow initialization to complete before writing data (10 ms)
  Cpu_Delay100US(100);

  writeCom4(0x80);
  Cpu_Delay100US(100);
}

//****************************************************************
// The routine sends the command to clear the screen then
// delays for the appropriate amount of time
//****************************************************************
void clrLCD(void){
```

```
  writeCom4(0x01);
  Cpu_Delay100US(35);
}

//*****************************************************************
// This routine displays the sensor readings on the LCD
//*****************************************************************
void display_sensors(byte *sensors)
{
  byte hundreds1, hundreds2, hundreds3, hundreds4;
  byte tens1, tens2, tens3, tens4;
  byte ones1, ones2, ones3, ones4;

  // Convert Rear Sensor Values for LCD Display
  hundreds1 = (sensors[0] / 100);
  tens1 = (sensors[0] - hundreds1 * 100) / 10;
  ones1 = (sensors[0] - hundreds1 * 100 - tens1 * 10);

  // Convert Front side Sensor Values for LCD Display
  hundreds2 = (sensors[1] / 100);
  tens2 = (sensors[1] - hundreds2 * 100) / 10;
  ones2 = (sensors[1] - hundreds2 * 100 - tens2 * 10);

  // Convert Front Sensor Values for LCD Display
  hundreds3 = (sensors[2] / 100);
  tens3 = (sensors[2] - hundreds3 * 100) / 10;
  ones3 = (sensors[2] - hundreds3 * 100 - tens3 * 10);

  // Convert Sensor Values for LCD Display
  hundreds4 = (sensors[3] / 100);
  tens4 = (sensors[3] - hundreds4 * 100) / 10;
  ones4 = (sensors[3] - hundreds4 * 100 - tens4 * 10);


  // DO WRITE COMMANDS HERE!
  writeCom4(0x80);
  writeData4(hundreds1 + 48);
  writeData4(tens1 + 48);
  writeData4(ones1 + 48);
  writeData4(0x20);
  writeData4(0x20);
  writeData4(hundreds3 + 48);
  writeData4(tens3 + 48);
  writeData4(ones3 + 48);
  writeCom4(0xA8);
  writeData4(hundreds2 + 48);
  writeData4(tens2 + 48);
  writeData4(ones2 + 48);
  writeData4(0x20);
  writeData4(0x20);
  writeData4(hundreds4 + 48);
  writeData4(tens4 + 48);
  writeData4(ones4 + 48);
}

/* END LCD */
```

Appendix E: Micromouse Curriculum Set

The micromouse curriculum is broken into six mandatory modules/laboratories. This appendix includes copies of the laboratory modules and micromouse final project description.

Appendix E.1: Laboratory #1: The Micromouse & Modified Flood Fill

Goals

The objective of this lab is to introduce the Micromouse competition and to introduce one of the many algorithms that can be utilized by a Micromouse to solve an unknown maze, the modified flood fill algorithm. At the completion of this lab, students should be able to write a program to implement the modified flood fill algorithm.

Background

Micromouse and Micromouse Competition

The Micromouse is an autonomous robotic mouse whose objective is to solve a 16x16 maze. The Micromouse is provided with a predetermined starting position and must correctly traverse an unknown maze and determine when it has reached the goal cell. Once the correct route has been identified, the mouse should return to the starting position, and then run the route in the shortest possible time. *Figure 1* provides a few images of some common Micromouse designs.

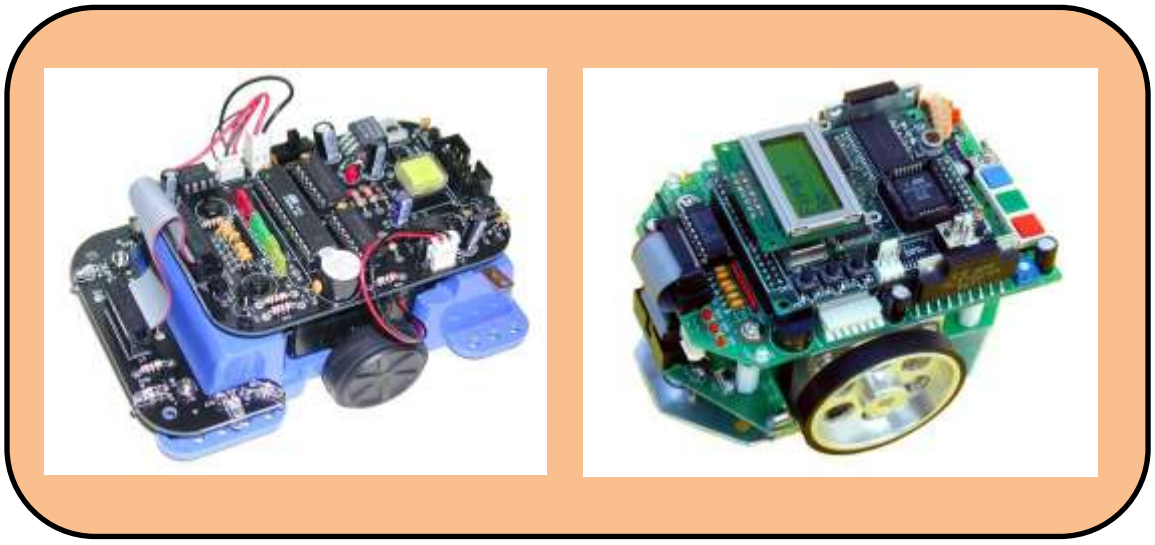


Figure 1: Micromouse Designs (Active Robots)

A Micromouse robot is generally composed of some combination of motors (DC or stepper) and sensors (proximity or distance), power source, display mechanism, chassis, wheels, and switches. A Micromouse robot typically is controlled using a microcontroller which reads sensors, controls

Micromouse movement, implements error detection and correction algorithms, and implements various searching algorithms.

The Micromouse framework is often utilized for robotics competitions. Micromouse competitions are generally targeted for student IEEE members (college undergraduates). The Micromouse competition was created by the Institute of Electronics and Electrical Engineers in 1977, and the first competition was held in New York in 1979 (IEEE) (Technology Innovation Centre). In a matter of a few years, the competition went global, and the first world competition was held in Japan in 1985 (Technology Innovation Centre). Competitions today are held across the United States and the world.

IEEE Micromouse Competitions are held based on Region. Region 2 includes West Virginia, Delaware, District of Columbia, Maryland, Southern New Jersey, Ohio (except Toledo), Pennsylvania, and Northern Virginia. The 2010 Region 2 Micromouse Competition was hosted by Temple University and held on April 17, 2010 (Temple University IEEE Student Branch). *Appendix A* includes a copy of the 2010 Region 2 Micromouse Competition Rules.

The Micromouse framework and competition serve as the foundation for this laboratory. Students will learn many features of the HSC12 microprocessor family by building, programming, and debugging a Micromouse robot.

Searching Algorithms

In order to solve an unknown maze, a Micromouse can implement a variety of different algorithms such as the flood fill algorithm, modified flood fill algorithm, A*, etc. In this lab, we will implement the modified flood fill algorithm but additional algorithms are encouraged to be explored by students throughout the course of the semester.

Flood Fill Algorithm

To first introduce the modified flood fill algorithm, let's first take a look at the flood fill algorithm. The flood fill is a simple algorithm that determines the area connected to a given

node in a multi-dimensional array. It sets the goal values (destination cells) to zero and "floods" the surrounding cells with radiating, increasing values. *Figure 2* provides an example.

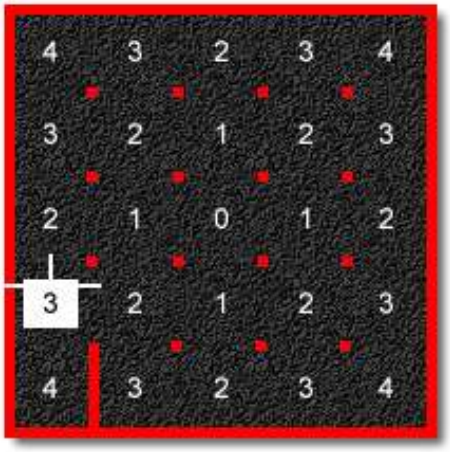


Figure 2: Flood Fill Algorithm [1]

Examining *Figure 2,* notice that the center value is zero and all other cells are filled with values corresponding to their distance from the goal cell. After "flooding" the maze with values, the algorithm then searches the adjacent nodes for the smallest value to determine which cell to travel. It then continues to follow the values in descending order until it has reached the center. This algorithm is rather simple and provides a reliable method to finding the center of the maze; however, the modified flood fill algorithm can find the center more quickly so we will introduce this algorithm as well.

Modified Flood Fill Algorithm

The modified flood fill algorithm is similar to the flood fill algorithm for it also uses distance values to navigate the maze. The primary difference is that the modified flood fill algorithm does not "flood" the entire maze with values. It modifies only the values that need to be changed. For example, if a wall is encountered and the robot is not in the destination cell, it updates the value of that cell to ***1 + the minimum value of its open neighbors.*** *Figure 3* provides an example of this process.

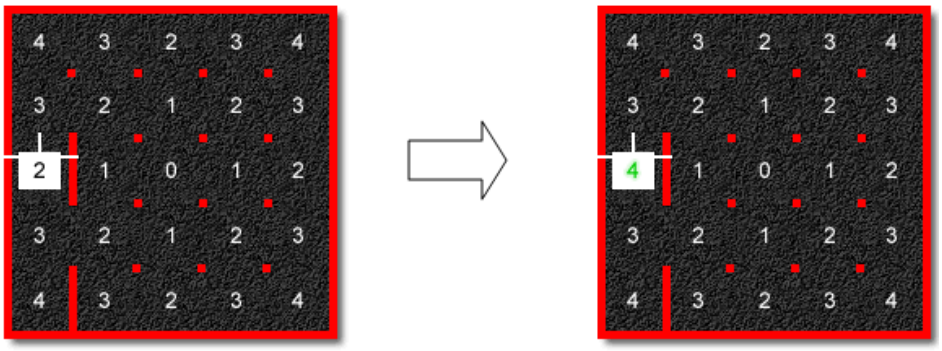


Figure 3: Modified Flood Fill Algorithm [1]

Examining *Figure 3*, when the robot encounters a wall to the east and can only move north or south. The north and south cells (open neighbors) are checked and we find that the current cell's new value is ***1 + the minimum value of its open neighbors*** or 1+3=4.

Once the robot has found the destination cell, it can return to the beginning of the maze using the distance values. On return, it is often a good practice to double check the wall positions. Once the micromouse has returned to the beginning, the maze is solved and the mouse can scamper off for a speed run to the center of the maze.

In summary, the modified flood fill process for **updating the distance values** is:

Update the distance values (if necessary)

Make sure the stack is empty

Push the current cell (the one the robot is standing on) onto the stack

Repeat the following set of instructions until the stack is empty:

{
Pull a cell from the stack
Is the distance value of this cell = 1 + the minimum value of its open neighbors?
No -> Change the cell to 1 + the minimum value of its open neighbors and
push all of the cell's open neighbors onto the stack to be checked
Yes -> Do nothing
}

Figure 4: Updating Distance Values with Modified Flood Fill Algorithm [1]

Using the algorithm from *Figure 4* and the wall map, the modified flood fill algorithm now becomes the following and should be executed every time the mouse enters a new cell.

1. Update the wall map (using the sensor readings)
2. Update the distance values (only if necessary)
3. Determine which neighbor cell has the lowest distance value
4. Move to the neighboring cell with the lowest distance value

Procedure

Develop C or MatLab functions to implement the Modified Flood Fill Algorithm described in the *Background* section. Note: There are several on-line references and video tutorials describing the modified flood fill algorithm on-line as well. Recommendations and more details on how to implement the algorithm are provided below.

The first thing that the Micromouse must keep track of is the location of the walls in the maze. One manner to store this information is by using an array. In order to store this information, we must first define the directions within the maze and assign a value to each direction as shown in *Figure 5*. Let's look at an example. If we are at the starting position of a 16x16 maze and we have three walls surrounding us: wall to the west, wall to the south, and wall to the east, the wall array would have the value = 0x02 + 0x01 + 0x08 = 0x0B or walls[0] = 0x0B.

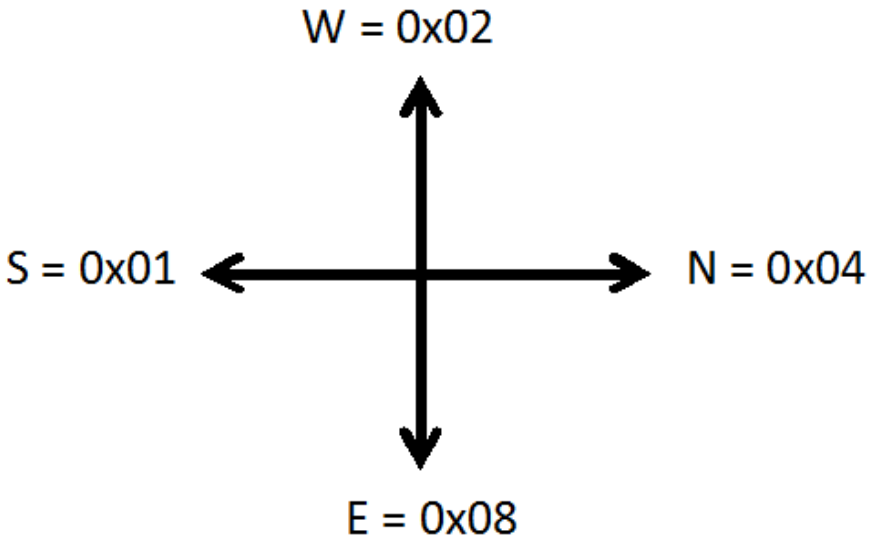


Figure 5: Direction Definitions

So, as you quickly realize, the walls array say of a 6x13 maze can already be defined with the positions of the outer walls and the starting cell (which always has 3 surrounding walls per competition rules).

B	2	2	2	2	2	2	2	2	2	2	2	6
1	0	0	0	0	0	0	0	0	0	0	0	4
1	0	0	0	0	0	0	0	0	0	0	0	4
1	0	0	0	0	0	0	0	0	0	0	0	4
1	0	0	0	0	0	0	0	0	0	0	0	4
9	8	8	8	8	8	8	8	8	8	8	8	C

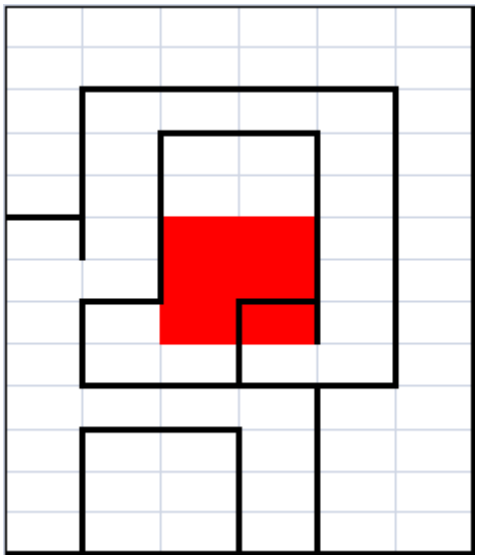
The second value you must store for the Micromouse to successfully traverse the maze are called distance values. Distance values (as mentioned earlier) store a value indicating the number of moves you are from the center of the maze. Taking the 6x13 maze, the distance value array should be initialized as shown below.

```
Initializes maze values --> Modified Flood Fill Algorithm
Puts values in maze[] array
Example of 6x13 array
maze[] = 7 6 5 4 3 2 2 2 3 4 5 6 7
         6 5 4 3 2 1 1 1 2 3 4 5 6
         5 4 3 2 1 0 0 0 1 2 3 4 5
         5 4 3 2 1 0 0 0 1 2 3 4 5
         6 5 4 3 2 1 1 1 2 3 4 5 6
         7 6 5 4 3 2 2 2 3 4 5 6 7
```

Besides distance values and wall values, there are two other items that must be accounted for to solve the maze: the Micromouse's current row, column position and which direction the Micromouse is facing. These items can be stored in several different manners and will be left up to you to implement.

Finally, you may want to write a function that checks all the possible next moves to determine which neighboring cell has the lowest distance value. You will only have to check the following: (row – 1, row + 1, column + 1, and column – 1).

Demonstrate that your algorithm can solve the following maze to your Teaching Assistant. If unable to complete this lab before the allotted time, you must demonstrate your completed algorithm prior to the fifth week of the semester.



Lab Questions / Notebook

1. Using the C programming language, write the code necessary to create two integer variables, A and B, initializing A to 10 and B to 20. Add A and B and store result in A.
2. What is the difference between logical and bit-wise operators?
3. Include a brief description of what was accomplished in this lab. Be sure to include all functions with comments.
4. Identify three additional algorithms besides the flood fill and modified flood fill algorithm that are used to solve an unknown maze
5. Read through the C Programming Tutorial on the class website at http://csee.wvu.edu/classes/cpe313/references/C_Tutorial.ppt. Answer the following questions.
    a. Show how to declare and initialize a variable X to 0x33;
    b. Show how to declare and initialize a character array of 10 characters with values a-j
    c. Show how to shift the upper nibble (lower four bits) of a character value "y" to the lower nibble (lower four bits).

References

- http://sites.ieee.org/sb-osu/
- http://www.micromouseinfo.com/introduction/mfloodfill.html

Appendix E.2: Laboratory #2: Microcontroller Introduction and Basics

Goals

The main objective of this lab is to introduce the microcontroller hardware and in particular, understand the process for creating, executing, and debugging application programs for the MC912C32 microcontroller.

1. Develop, execute, and debug a program on the microcontroller using the HCS12 serial monitor.
2. Become familiar with Dragonflybot Board Hardware features
3. Understand the advantages and disadvantages of using the Processor Expert plug-in

All manuals and complete schematics for the MC9S12C-family microcontroller, Dragonflybot board, and Dragonfly12 module are located on the class website at http://www.csee.wvu.edu/classes/cpe313/labs.html.

Background

MC9S12C Microcontroller

The MC9S12C microcontroller features a 16-bit CPU, up to 128K bytes of Flash EEPROM, 4K bytes of RAM, serial communications interface (SCI), serial peripheral interface (SPI), 8-channel 10-bit analog-to-digital converter (ADC), 6-channel 8-bit pulse width modulation (PWM), and a 8-channel 16-bit timer module (TIM). *Figure 1* provides an illustration of the MC9S12C, and *Figure 2* provides the MC9S12C-Family Block Diagram.

Figure 1: MC9S12C Microcontroller

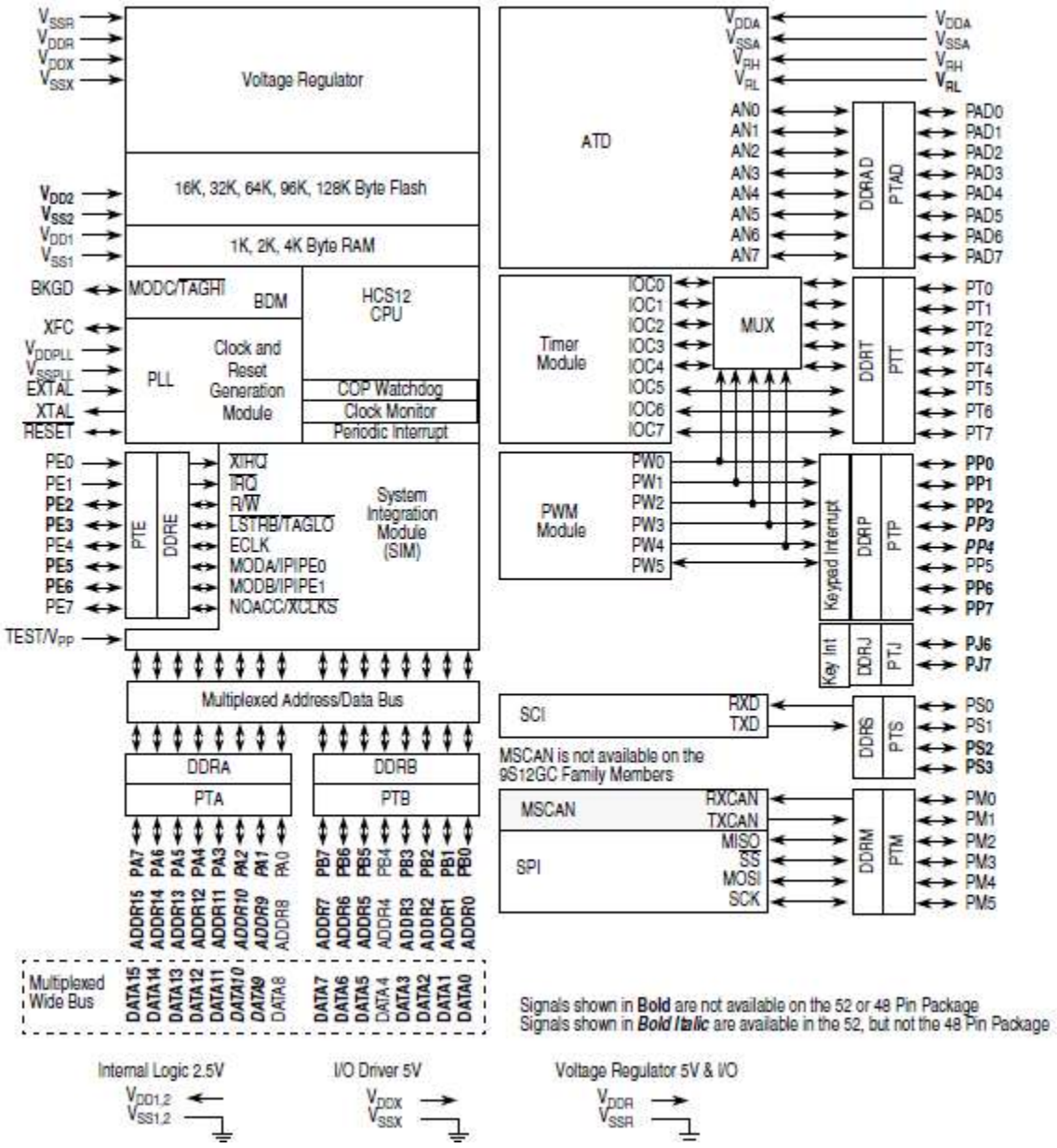


Figure 2 MC9S12C-Family Block Diagram

Dragonfly12 Module

The Dragonfly12 Module provides a convenient prototype platform for engineers who want to design, develop, and prototype new MC9S12C32 applications. The Dragonfly12 module bonds 40 of the MC9S12C32 microcontroller pins to the package. This allows programmers to access the MCS12C32 microcontroller to access its features. *Figure 3* provides a snapshot of the Dragonfly12 module, and *Figure 4* provides its pin-out diagram.

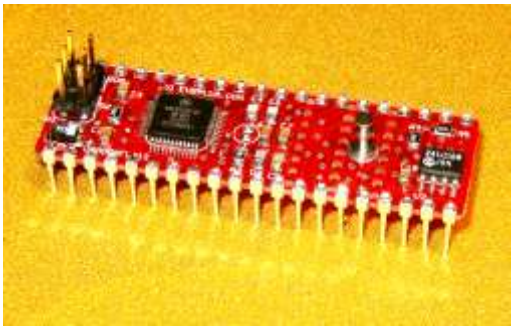


Figure 3 Dragonfly12 Module

1	GND	40	PP5/KWP5
2	VCC	39	PM2/MISO
3	/RESET	38	PM3/SS
4	PT0/IOC0/PW0	37	PM4/MOSI
5	PT1/IOC1/PW1	36	PM5/SCK
6	PT2/IOC2/PW2	35	PS1/TXD
7	PT3/IOC3/PW3	34	PS0/RXD
8	PT4/IOC4/PW4	33	VRH
9	PT5/IOC5	32	AN7/PAD07
10	PT6/IOC6	31	AN6/PAD06
11	PT7/IOC7	30	AN5/RAD05
12	PE1/IRQ	29	AN4/PAD04
13	PE0/XIRQ	28	AN3/PAD03
14	PE4/ECLK	27	AN2/PAD02
15	PM0/RXCAN	26	AN1/PAD01
16	PM1/TXCAN	25	AN0/PAD00
17	BKGD	24	XTAL
18	PE7/XCLKS	23	EXTAL
19	CAN_HIGH	22	PA0
20	CAN_LOW	21	PB4

Figure 4 Dragonfly12 Module Pin-outs (40 pins)

Dragonflybot Board

The Dragonflybot board is a low-cost robot control board made to connect Dragonfly12-DIP modules [1]. Some of the features of the Dragonflybot board include four robot servo controllers, four push button switches, 5V regulator, dual H-bridge for controlling two DC motors or one stepper motor, support 4 GP2-D12 distance measuring sensors, and 20x2 female headers which provides input-output pins of the MC912C32. The Dragonflybot board also includes 28 jumpers. The jumpers are generally used to enable and disable features and change settings. In this lab, we will use the on-board push buttons and LEDs on the Dragonflybot board. *Figure 5* provides the schematic of the component side of the Dragonflybot board.

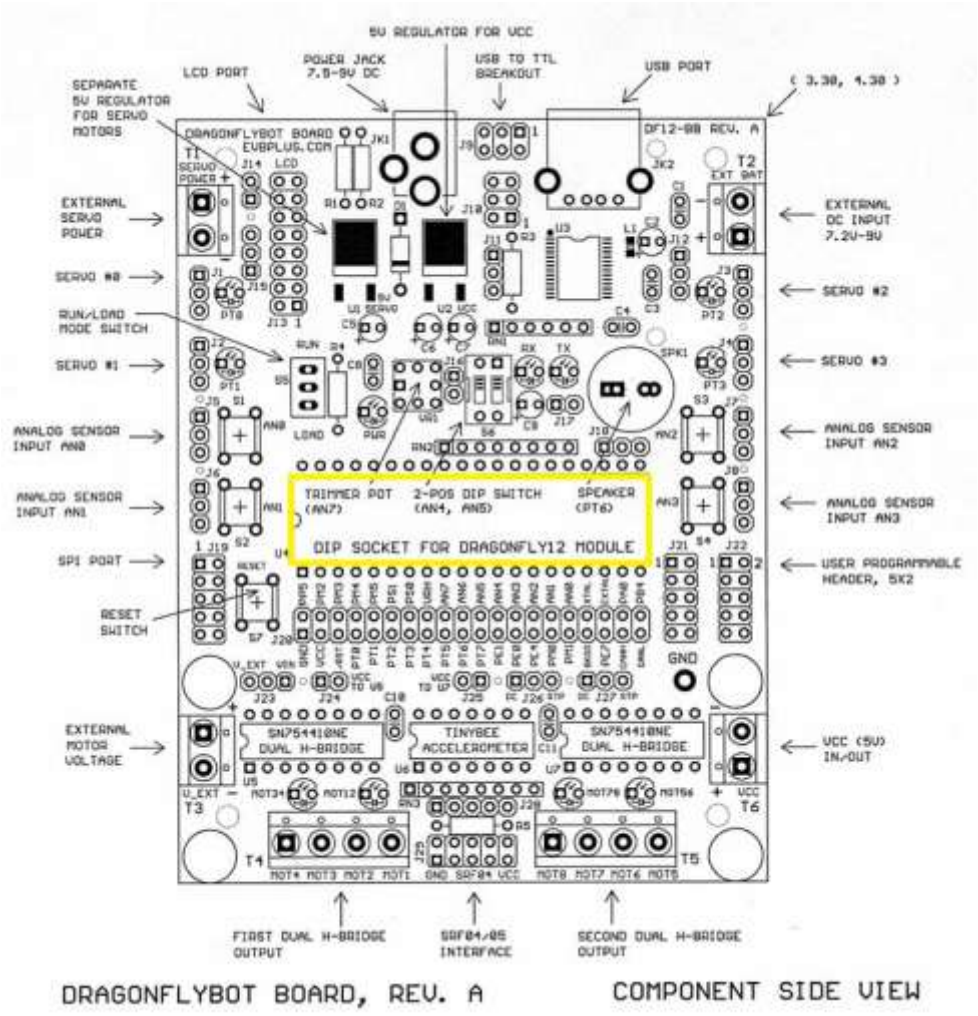


Figure 5: Dragonflybot Board – Component Side Schematic

[http://www.evbplus.com/c32_modules/bbu_hc12_68hc12_9s12_hcs12.html]

Freescale CodeWarrior with Processor Expert

CodeWarrior is a complete Integrated Development Environment (IDE) for developing embedded applications. CodeWarrior integrates the editor, compiler, linker, debugger, and other software modules. The IDE manages the control and execution of the tools. CodeWarrior enables engineers to build efficient HCS12 systems.

Processor Expert is a CodeWarrior plug-in that is designed for rapid application development of embedded applications for a variety of microcontrollers. Processor Expert implements embedded beans, reusable software components, to encapsulate the functionality of basic elements such as the CPU core, on-chip peripherals, and pure software algorithms. The plug-in also consists of a Bean Wizard, which is a graphical user interface specifically designed for the creation and modification of embedded beans [Processor Expert Online].

Setup

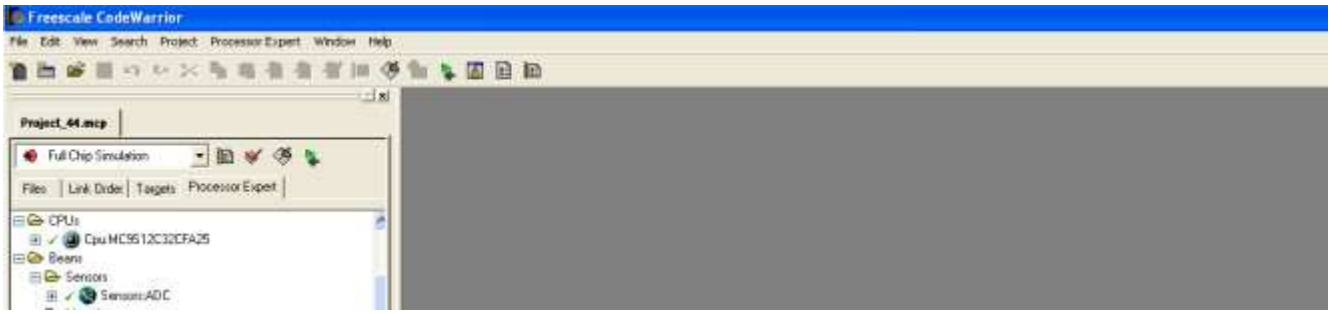
To get started, the first thing you need to do is create a new CodeWarrior project with Processor Expert enabled. To create a new CodeWarrior Project, follow these steps.

Creating a New CodeWarrior Project

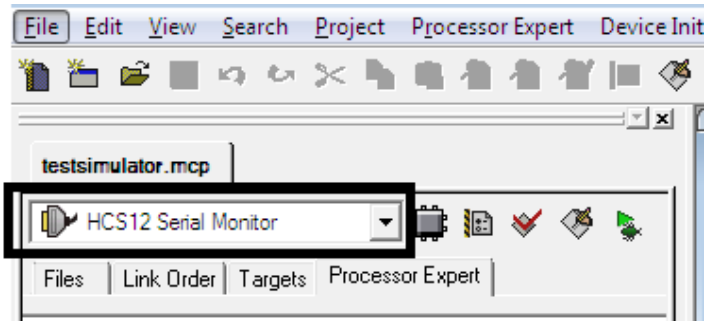
Note: It is very important that you follow these instructions because this project will be used throughout the semester.

1. Select File > New Project
2. Select the HCS12C Family > MC912C32 microcontroller
3. Choose “Full Chip Simulation” as the default connection and make sure that the Serial Monitor is also a selected (as necessary based on software version)
4. Change the project name as desired and make sure you jot down where the project is going to be saved. **When you save a CodeWarrior project, it is recommended that you save the entire workspace.**
5. Make sure the “C” programming language is checked to be supported
6. Select “Processor Expert” for Rapid Application Development Options
7. Choose ANSI startup code, small memory model, and add floating numbers support by selecting the “float is IEEE32, double is IEEE32” option

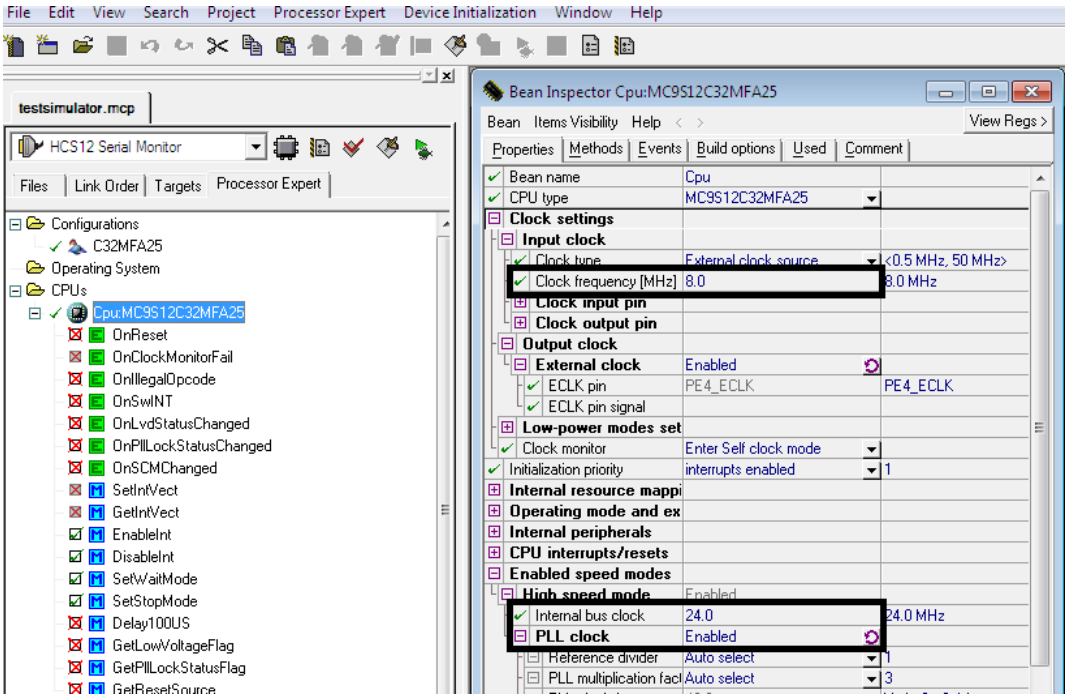
8. Select No to create project setup for PC_lint
9. When prompted by Processor Expert, select the 48-pin MCU (MFA25 will be fine)



Notice in your leftmost window the option to change your hardware connection type or select the simulator. Since we will be connecting board through the serial monitor, go ahead and make this change within your project as shown below. Note: You may have to change the default COM port later before the program will successfully load. Also, you are encouraged to use the simulator to help debug your applications.



Next, you must modify the CPU settings using Processor Expert's bean inspector to match the bus speed of the Dragonflybot board. The default bus speed on the Dragonflybot Board is 8 MHz. You need to enable the PLL clock, set the input clock frequency to 8.0 MHz, and the internal bus clock to 24.0 MHz. These settings can be modified by selecting the CPU in the Processor Expert window and modifying the settings in the Bean Inspector window. If the bean inspector window is not displayed, select Processor Expert > View > Bean Inspector.



In the leftmost window under the Processor Expert tab, you will notice there is a main.c module. This is the location of your main function for your program. You will notice if you double click on the file, you will get an error stating that the main module cannot be found. The reason for this is that since you are using the Processor Expert plug-in, the main module is not generated until you leverage Processor Expert by generating any autogenerated code. Go ahead and select Processor Expert > Generate Code. After the code is generated, you will now be able to open your main module. Now, you are ready to write your application that can be loaded on the microcontroller. Note: As expected, if you add any Processor Expert beans or modify any methods, you will need to regenerate the Processor Expert code.

Procedure:

Activity #1: Write a function that creates a 4-bit counter that counts down from 15-0. When the counter reaches 0 it stops until a pushbutton is pressed to initiate the counter again. For this experiment, you can use pushbutton "S4" and LEDs PT0-PT3 on the Dragonflybot board.

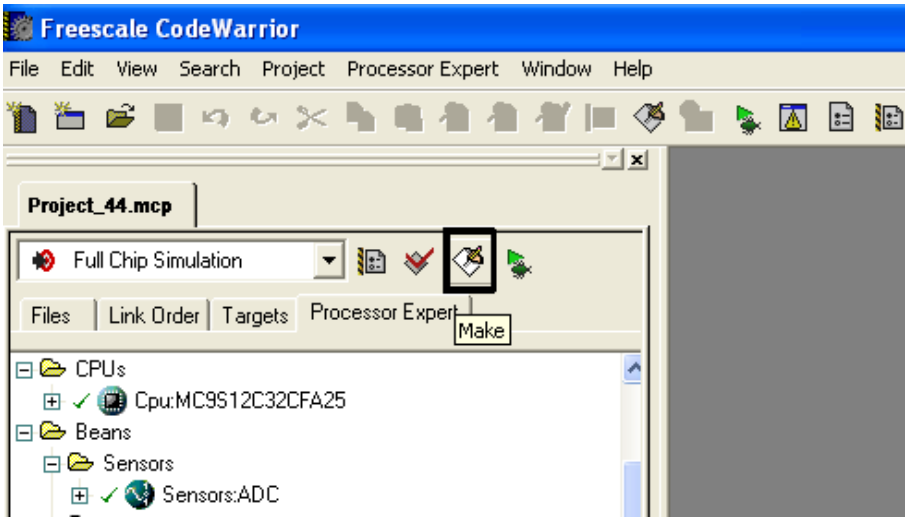
1. In this lab, we will leverage the functionality encapsulated in the Processor Expert beans to use the microcontroller's features.

2. First, use Processor Expert's bean selector (Categories View) and select the appropriate Port IO beans to implement this functionality. You can view descriptions of the beans by right clicking the bean and clicking "Help on Bean". You can add the beans to your project by double clicking them. (You can remove them if needed as well). The added beans will show up in your "Beans" directory. You can view the properties for a given bean in the Bean Inspector Window.
3. Tailor the properties of the beans for the functionality you need to implement. You need 4 bits or a nibble for the LEDs, and you need one bit for the push button switch.
4. Once the beans have been selected and configured, go ahead and Generate the Processor Expert code by selecting "Processor Expert" > "Generate Code".

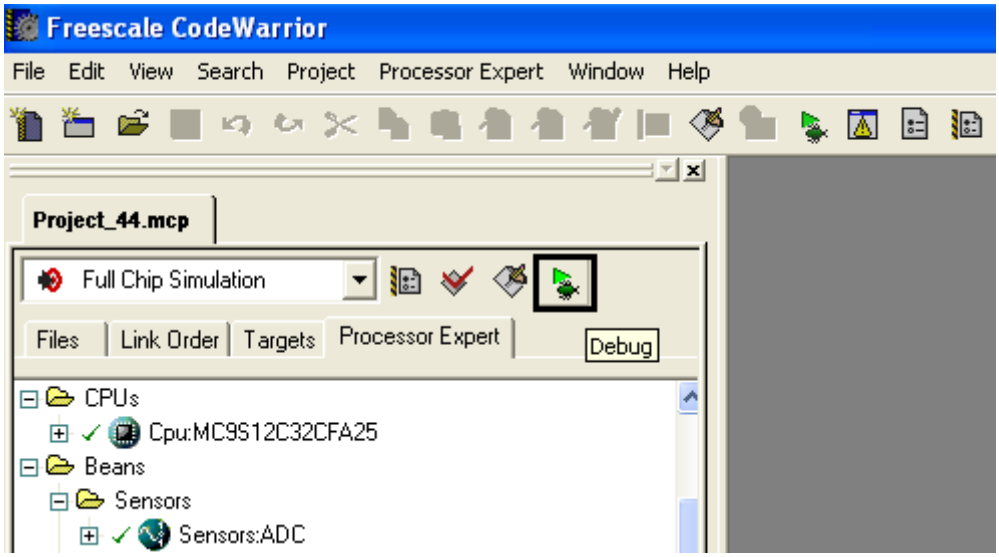
   Note: Processor Expert generates code based on the beans and properties selected. This code resides in the "Generated Modules" directory. The main routine is located in the "User Modules" directory.
5. Write a function that uses the beans to create the 4-bit down counter.
6. Compile, Debug, and Load your code on the microcontroller.
7. Demonstrate the successful execution of your program to your TA.

Compiling, Debugging, and Running

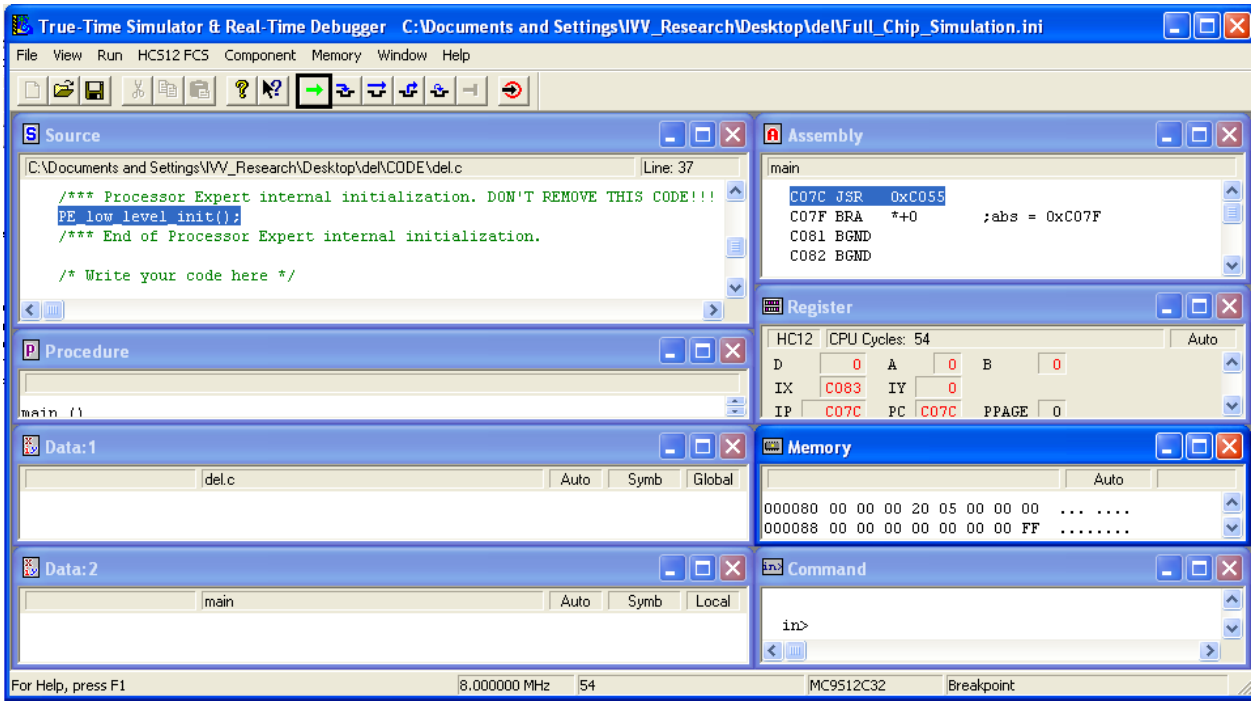
- To compile the project: Select Project > Make *or* Hit Make Button *or* Hit "F7"



- To debug and download the project: Select Project > Debug *or* Hit Debug Button *or* Hit "F5"



- To run the project with the Real-Time Debugger: Select the Run button



Programming the Microcontroller

- In CodeWarrior, Generate Processor Expert code and compile code
- Make sure the HCS12 serial monitor is selected.
- Properly connect the USB cable to the Dragonflybot board and your workstation. The PWR LED should illuminate on the Dragonflybot. If not, check
- At Switch 5 (S5) on the Dragonflybot board, there is a "Run/Load" switch. Make sure the switch is in the "Load" position.
- Press the "Reset" button located at S7.
- In CodeWarrior, download the code.
- Select the appropriate COM port as necessary
    - If unsure of correct COM port, use Windows Device Manager to find it
- You should now see the Real-Time Debugger window
- You can run the code using the debugger

Activity #2: Real-Time Debugger. The real-time debugger is a powerful tool that provides insight into your application. It provides real-time information on global and local variables, assembly language equivalent commands, register values, and memory values.

1. Using the Real-Time Debugger, step through your program.
2. Insert breakpoints into your program and execute your code.
3. Examine the following: start/continue, single step, step into, step out, assembly step, stop, and reset. **Be sure to document what each of these functions do in your lab report.**

Questions

1. Discuss some advantages and disadvantages of Processor Expert
2. Examine the Dragonflybot Board Manual. Identify the jumper connections that allow you to enable and configure the on-board LCD.
3. Read the LCD Manual (8X2 Display).
    a. Determine the amount of wait time required before you can communicate with the LCD after powering on (4-bit interface). How would you create this delay in your program?

b. Examining the Character Font Table, in order to send a decimal 3 to the LCD, you must send the byte 0011 0011. If you wanted to write a function called writeData(3) which would write the value 3 to the LCD, how would you have to manipulate the parameter value inside your function. Does this work for all ASCII values? Note: You don't have to write out the function.

References:

- http://www.evbplus.com/c32_modules/bbu_hc12_68hc12_9s12_hcs12.html
- http://download.datasheet1.com/down/160417_HANTRONIX_HDM08216L.html
- http://www.eng.auburn.edu/~nelson/courses/elec3050/

Appendix E.3: Laboratory #3: LCD Interfacing

Goals

The objective for this laboratory is to learn how to communicate with an 8x2 LCD using a HCS12 microcontroller. In particular, you will have to write five functions to interface with the LCD.

1. Initialize the 8x2 LCD
2. Send a single character to the LCD
3. Send a command to the LCD
4. Clear the LCD
5. Display Numerical Value (Sensor Values)

Background:

Hantronix HDM08216L 8x2 LCD

A LCD is a display device that requires small amounts of electrical power and can prove very useful when debugging robotic applications. This lab uses the Hantronix HDSM0216L 8x2 Liquid Crystal Display (LCD). The 8x2 display connects on the Dragonflybot Board at J13, and the LCD backlight can be enabled at J14. The complete datasheet is at: http://www.csee.wvu.edu/classes/cpe313/references/LCD_Manual.pdf. Be sure to familiarize yourself with the datasheet as it will help you with this experiment.

Figure 1 shows the pin-out diagrams for the Hantronix LCD. Of the 14 pins shown, only have to be concerned with the six pins connected to the microcontroller via the J13 header (RS pin (4), EN pin (7), and the four data pins (DB4-DB7)).

The pinouts of J13 are as follows:

Pin 1	GND	
Pin 2	VCC (5V)	
Pin 3	Via a 1K Ohm resistor to GND	
Pin 4	PA0	RS pin for LCD module
Pin 5	GND	R/W pin for LCD module
Pin 6	PE7	EN pin for LCD module
Pin 7	Not used	
Pin 8	Not used	
Pin 9	Not used	
Pin 10	Mot used	
Pin 11	PM2 or PP5	DB4 pin for LCD module
Pin 12	PM3	DB5 pin for LCD module
Pin 13	PM4	DB6 pin for LCD module
Pin 14	PM5	DB7 pin for LCD module

Figure 1 LCD Pin-outs

Control Register (CG RAM)

- The **Read Select (RS)** line is used to select which register with the LCD Module the data will be going to. When RS = 0, the data is sent to the control register, and when RS = 1, the data is sent to the data screen.

- The **Read/Write (R/W)** line simply used to read from the LCD or write to the LCD. Since we will only be writing to the LCD, the R/W pin is grounded when using the J13 header on the Dragonflybot board.

- The **Enable (EN)** line disables the LCD (0) and enables the LCD (1).

Data Register (DD RAM)

- The **Data Bus (DB4 – DB7)** is used to send data to the LCD.

Examining the IO lines, you can see that to control the LCD, you have 6 lines available, but a character is 8-bits, which would require 8 data lines (DB0-DB7). However to save IO pins, the LCD Module also has a 4-bit mode of operation which only uses 4 data lines. In 4-bit mode, the 8-bit ASCII data is split into 2 nibbles which are sequentially sent to data lines DB4-DB7, each with its own data strobe being presented to the Enable line.

Procedure:

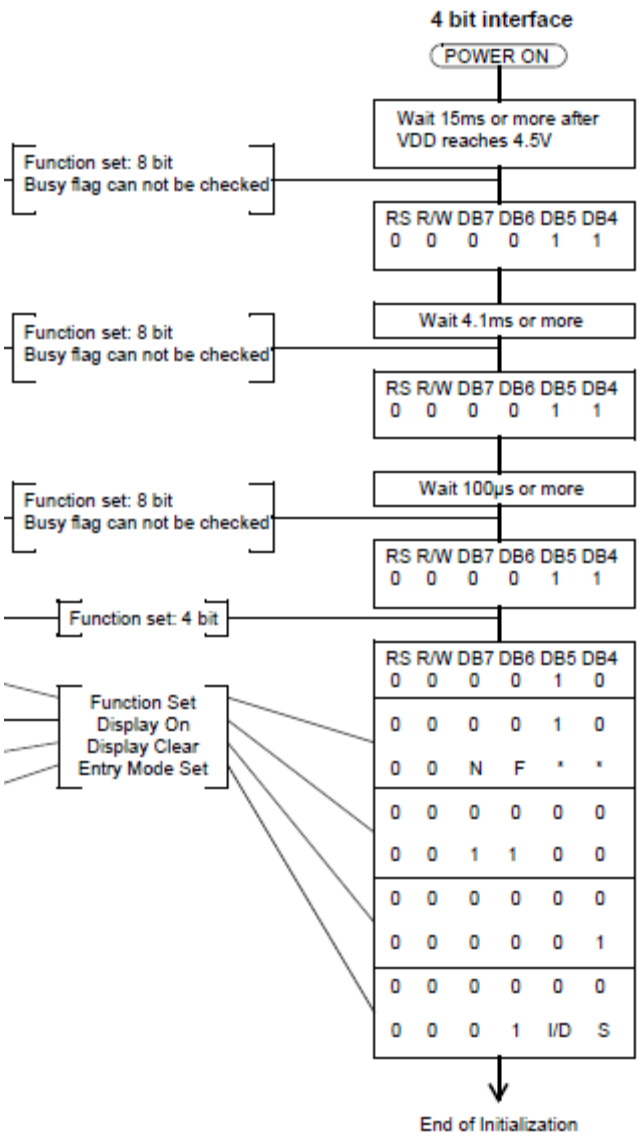
Activity #1: Write a function that can be used to send a Command to the LCD in 4-bit Mode

1. Write a function that accepts a character as a parameter and writes the character to the control register. Function Prototype: void writeCom (byte command)

2. As discussed in the section above, in order to send a command in 4-bit mode, you must strobe the upper four bits and then sequentially strobe the lower four bits. Since commands are only four bits, we only have to worry about the lower nibble in this case.
   Example: Strobe Lower Four Bits.
   - Send the lower four bits (to be passed into the command parameter) with the EN=0 and the correct RS value
   - Send the lower four bits (to be passed into the command parameter) with the EN=1 and the correct RS value
   - Send the lower four bits (to be passed into the command parameter) with the EN=0 and the correct RS value

Activity #2: Initialize the LCD. Use the function that you wrote in Activity #1.

1. Using the function from *Activity 1*, write a function to initialize the LCD in 4-bit mode. *Figure 2* provides the steps from the LCD Manual to accomplish this task. The values for N, D, S, and I/D can be found in the LCD manual are provided below as well. Select the appropriate values. You can utilize the cpu method CPUDelay100US to create the needed delay times. Function Prototype: void InitializeLCD()
2. Compile, Load, and Run the Program. For this lab, power the Dragonflybot Board from the USB cable and not the battery.
3. Demonstrate to your TA that the LCD initializes properly by showing a blinking cursor on the LCD Module.



I/D = 1: Increment I/D = 0: Decrement
S = 1: Accompanies display shift.
S/C= 1: Display shift S/C = 0: cursor move
R/L= 1: Shift to the right. R/L= 0: Shift to the left.
DL = 1: 8 bits DL = 0: 4 bits
N = 1: 2 lines N = 0: 1 line
RE = 1: Ext. Reg. Ena. F = 0: 5 x 7 dots
BF = 1: Busy BF = 0: Can accept data
 # Set to 1 on 24x4 modules
$ With KS0072 is Address Mode.

Figure 2 4-bit LCD Initialization Routine

Activity #3: Write a character to the LCD

1. Write a function that accepts a character and writes that character to the LCD screen. This function should be similar to function written in *Activity 1* except for the RS line. With the writeData function, you will need to send the upper nibble first and then the lower nibble (be sure to strobe the enable bit for each nibble).
   Function Prototype: void writeData (byte data).
2. Demonstrate this function to your TA. Write an 'a' to the LCD by calling your function in the main routine.

Activity #4: Clear the LCD

1. Write a function that clears the LCD using the commands from the LCD manual.
   Function Prototype: void clearLCD()

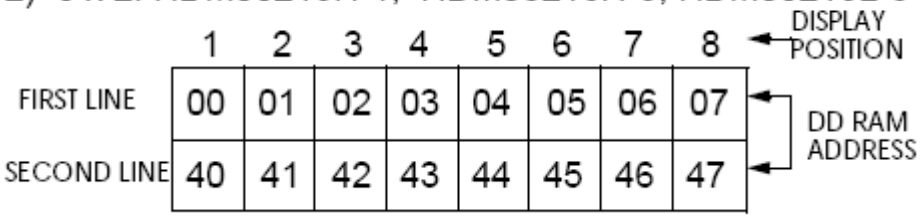
Command	Code										Description	Execution Time
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears the display and returns the cursor to the home position (address 0).	82μs~1.64ms

Activity #5: Write Sensor Value to the LCD

In an upcoming lab and for your final project, you will be reading numerical values from the sensors mounted on your Micromouse. It will be helpful for debugging purposes to have a function which can write these values to the LCD.

1. Write a function which accepts two unsigned integer values and displays the values on the LCD. The first value should be displayed on the first line of the LCD, and the second value should be displayed on the second line of the LCD.
   Function Prototype: void displaySensors (unsigned int value1, unsigned int value2)
2. Display two numerical values – 1024 and 255. Demonstrate your function to the TA. To accomplish this task, you need to reference the LCD datasheet and implement the appropriate command(s) to send the values to the correct positions. For the LCD we are using, the 8x2 character position and respective character address of each position is shown below.

2) 8 x 2: HDM08216H-1, HDM08216H-3, HDM08216L-3



Questions

1. Examine the GP2D120 Proximity Sensor datasheet located on the class website. http://www.csee.wvu.edu/classes/cpe313/references/gp2d120.pdf
    a. What is the detecting distance of the GP2D120? (5 points)
    b. The Proximity Sensor has three connectors. How would you wire up these connectors?
2. Provide some examples of analog devices and digital devices
3. How would the function that you wrote in *Activity #5* change if you were using a 16x2 Hantronix LCD?
4. Why did we have to toggle the enable bit in this experiment to successfully send a character to the LCD?

References

- http://download.datasheet1.com/down/160417_HANTRONIX_HDM08216L.html
- http://www.hobbyengineering.com/specs/gp2d120.pdf

Appendix E.4: Laboratory #4: Proximity Sensors

Goals

In this lab, you will learn how to read the GP2D120 proximity sensors and use the returned values to determine the distance from an object.

Background

GP2D120 Proximity Sensor

The GP2D120 proximity sensor *(Figure 1)* is an infrared sensor by Sharp which has a detecting range of 4 cm to 30 cm. An infrared sensor consists of an infrared transmitter and infrared receiver. The transmitter sends out an invisible beam of light into the environment and the receiver absorbs the light that is reflected back. The angle of the reflected beam indicates the distance of the receiver to the object that is reflecting the light [1].

Figure 1 GP2D120 Proximity Sensor

The sensor requires three connections: supply voltage, ground, and output values, as shown in *Figure 2*. This sensor takes about 48 ms to get one distance reading. This factor is important so that we use the sensor appropriately in our programs.

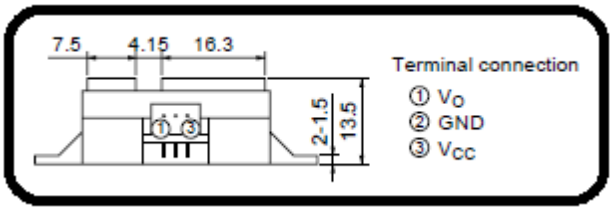


Figure 2 GP2D120 Proximity Sensor Terminal Connection

The micromouse configuration has three proximity sensors (left, front, and right). The left sensor is connected to pin AD0, the front sensor is connected to pin AD1, and the right sensor is connected to AD2. All of the sensors will be later used to detect the maze walls, and the right and left sensors can be used as inputs to the micromouse control algorithm.

Micromouse Application of Proximity Sensors

The proximity sensors serve two purposes for our micromouse application. The first purpose is to ensure that the robot detects errors while it's moving through the maze. Thus, the sensors can be use to correct the robot's path and ensure that the robot does not crash into a wall. The left and right sensors will be used for this purpose. The second purpose is to map the maze by determining the location of the walls. All three sensors will be used for this purpose.

Analog-to-Digital Converter (ADC)

An analog-to-digital converter changes a continuous signal (output from the proximity sensor) to discrete digital numbers. The digital output can then be represented in any number base representation such as binary, hexadecimal, decimal, etc. The **resolution** of an ADC is the number of bits in the output conversion. The microcontroller's ADC can be configured to perform either 8-bit or 10-bit conversions. The possible values that can be used to represent the input voltage for an ADC are $2^N - 1$, so for an 8-bit ADC, the possible values are 2^8-1 = 255. For example, if you have an analog input voltage with a range from 0-5 V and an 8-bit ADC, you will have an ADC voltage resolution or step size = $(5\text{-}0)/2^8$ = 0.0195 V. In conclusion, your analog signal can be represented by 256 discrete values with 0.0195 V steps. It is also important to note that if you increase the resolution, then you increase the accuracy of the ADC.

Procedure

Activity #1: Read proximity sensor values and display values on LCD.

1.) Open up your Freescale Project.
2.) Insert the ADC embedded bean into project.
3.) Initialize the ADC bean
    a. Use a 10-bit resolution ADC

b. Modify the number of conversions to 8. Increasing the number of conversions decreases the probability of sensor read errors.
c. Select a conversion time of 20 us. This value represents the amount of time for each conversion.
d. Select the appropriate number of channels and ensure that the appropriate pins are selected

4.) Generate the processor expert code.
5.) Select the appropriate method(s) to read the sensor values.
6.) Display the sensor values on the LCD as follows: display left sensor, delay, display right sensor, delay, display front sensor, delay.

Activity #2: Convert the sensor values into centimeters and display the values on the LCD.

1.) Write the necessary code to convert the sensor values into centimeters.

Questions

1. Sensors are used to provide information to the robot about its environment. Robots process the information from sensors and then react in a predetermined way. We will use the proximity sensors (a type of light sensor) to detect the surrounding walls in the maze to keep track of the path to the center and to keep the robot aligned within the maze. Describe how some other types of sensors work and describe some applications for each type of sensor.
    - Ultrasonic Sensor
    - Touch Sensor
    - Sound Sensor
2. If an ADC has a resolution of 16 bits with reference voltages of -5V to +5V, what is the step size for this setup? What would an analog input of -1.24 V be converted to if the output was in binary code?
3. What are the advantages and disadvantages of using stepper motors for a micromouse application?

References

- http://www.physics.unlv.edu/~bill/ecg497/Drew_Tondra_report.pdf

Appendix E.5: Laboratory #5: Stepper Motor Operation

Goals

This lab provides an introduction of how to control a stepper motor using the HCS12 microcontroller. At the conclusion of this lab, students should be able to effectively control stepper motor speed. Students will also try to develop the fastest Micromouse possible.

Background

Stepper Motor

A stepper motor is a brushless, electric motor that can divide a full rotation into a large number of steps [1]. This allows the motor's position to be controlled precisely without any type of feedback mechanism. Stepper motor's have multiple electromagnets arranged around a central gear. The electromagnets can be energized by an external device such as an HCS12 microcontroller. To make the stepper motor move one step, power the first electromagnet to make the gear's teeth to align to the first electromagnet. As current is applied to electromagnets and they are energized in sequence, the motor turns. *Figure 1* provides a snapshot of a stepper motor's internal components.

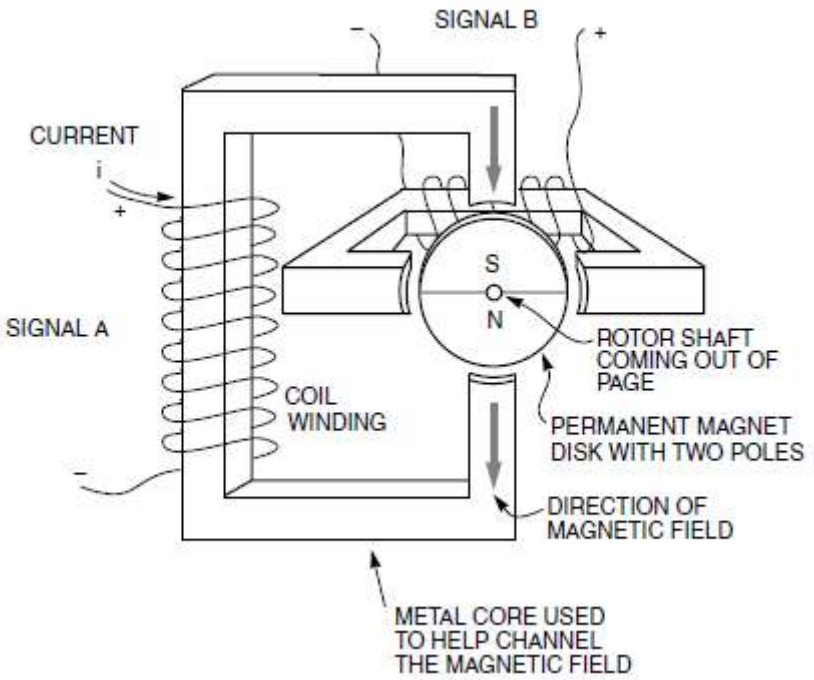


Figure 1: Stepper Motor Internal Components [2]

Driving a Stepper Motor

Stepper motors have a set of input pins that allow current from a supply source (in our case, a microcontroller) into the coil windings of the motor [2]. Sending pulsed waveforms in the correct sequence can be generated to create the electromagnetic fields needed to drive the motor. There are a few different techniques to choose from when selecting a waveform to drive a stepper motor. Two of these techniques are full stepping and half stepping. Standard motors have 200 rotor teeth or 200 full steps per revolution of the motor shaft. Dividing 200 steps into 360 degrees per rotation equals 1.8 degrees full step angle. Thus in full step mode, one digital input is equivalent to one step. Half stepping means that the motor is now rotating at 400 steps per revolution and that for each digital input, the motor only rotates half a step or 0.9 degrees. In this lab, you will drive the stepper motors in half stepping mode.

Driving a Stepper Motor using a HCS12 Microcontroller

The HCS12 microcontroller can be programmed to interface with many different types of stepper motors [2]. Microcontrollers can generate the appropriate waveforms to make stepper motors rotate. The micromouse stepper motors used in this lab are Shinano Kenshi stepper motors. Each motor has 4 input pins. The input voltage of the motor is 3.15 V with a typical current of 1 amp. To control the four pins of the motor, the microcontroller needs four output pins that are capable of sinking somewhere between 1 amp out of each pin [2]. The port pins on the HCS12 microcontroller are not suitable for this task. As a result, some additional circuitry (a motor driver) is necessary to drive the stepper motors with the HCS12 microcontroller. As a result, you will notice on the Dragonflybot Board (U5 and U7) that there are two SN754410 drivers (one for each motor) to drive the currents up to 1 amp [4]. *Figure 2* provides the pin-out diagram for the SN754410 drivers. The motor drivers can be enabled on the Dragonflybot Board by appropriately configuring jumpers J24, J25, J26, and J27 for stepper motor operation. When these jumpers are configured correctly, no wiring is required between the microcontroller, the drivers, and the motor terminal blocks T4 and T5. *Figure 3* provides diagrams of the wiring connections of the motor drivers on the Dragonflybot Board.

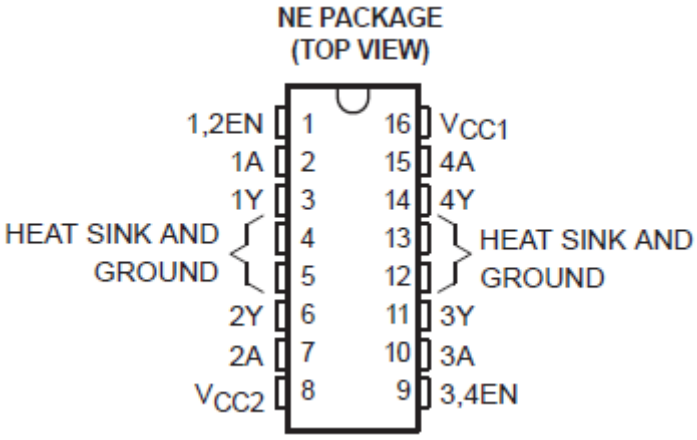


Figure 2: SN754410 Pin-out Diagram [4]

The drivers can be enabled/disabled using the connections at pins 1 and 9. The inputs to the driver at located at pins 2, 7, 10, and 15, and the outputs are located at pins 3, 6, 11, and 14.

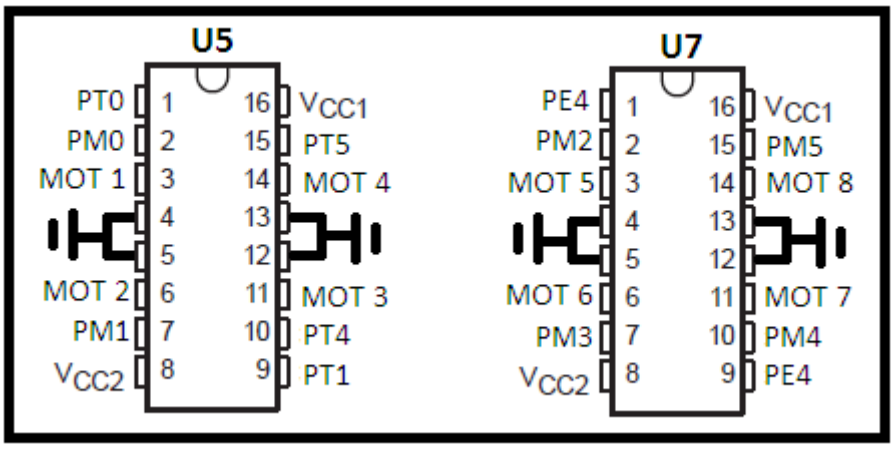


Figure 3: Dragonflybot Motor Driver Wiring Connections

Examining *Figure 3*, there are two items to note before performing this lab experiment. First, pin PE4 is also used as the external clock pin on the microcontroller and will thus have to be disabled so that it can be used to enable and disable the motor driver at U7. Secondly, pins PM2-PM5 are used as the input pins for the right motor. These pins were also used to control the LCD. As a result, you will not have to add any additional control bits to your project but will have to be careful when using these pins in this lab and future labs to when controlling the motor and using the LCD functions.

Figure 4 provides a block diagram of the MCU interface to the stepper motor coils.

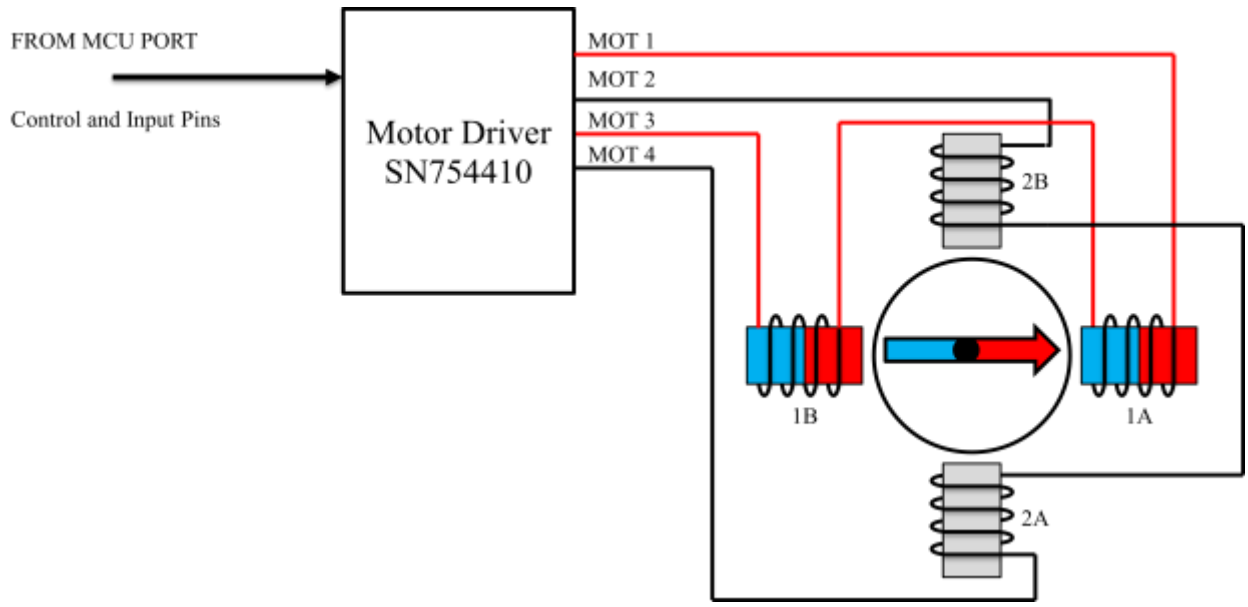


Figure 4: MCU Interface to Stepper Motor Coils [5]

By energizing the coils of the stepper motor in the correct sequence, the stepper motor can be rotated in either the clockwise or counterclockwise direction. *Table 1* provides the pattern used to energize the coils of the stepper motor.

MOT 1	MOT 2	MOT 3	MOT 4
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0

Table 1: Stepper Motor Energizing Pattern [5]

Setup

Before you get started, two functions are provided to reduce the current sent through the motor drivers. To accomplish this task, you will need to create 2 timer interrupts and name them Motor1_Timer2 and Motor2_Timer 2 and select the settings that follow.

1. Set the interrupt priorities to “low priority”
2. Set the interrupt period to 200 microseconds for both timer interrupts
3. Create two global bool variables toggle1 and toggle2
4. In the Events code for each timer interrupt, write the following code

```
//*********************************************************************
void Motor1_Timer2_OnInterrupt(void)
{
  if (toggle1){
    Motor1_WriteBits(0);
    toggle1 = 0;
  }
  else {
    Motor1_WriteBits(Motor1Step[next_state1]);
    toggle1 = 1;
  }
}

//*********************************************************************
void Motor2_Timer2_OnInterrupt(void)
{
  if (toggle2){
    Motor2_WriteBits(0);
    toggle2 = 0;
  }
  else {
    Motor2_WriteBits(Motor2Step[next_state2]);
    toggle2 = 1;
  }
}
```

If you need assistance with this initial code, please ask your teaching assistant.

Procedure

Activity #1: Write two functions (one for each motor) that can be used to send the appropriate energizing sequences to the drive the stepper motors. These functions should move the motor a half step every time they are called.

- Open up your previous project
- Create a Motor.h header file and create a Motor.c user module to write your code in
- Select and initialize the necessary Processor Expert beans
    - To send correct energizing sequences
    - To enable the motor drivers
- Write the two functions using the methods from the beans you selected

- Test your functions by calling them in main in a loop with a 4 millisecond time delay.
- Helpful Hints
    - One method to accomplishing this task is to store your motor sequences in a global array such as char Motor2Seq[8] = {…,…,} and having a variable that keeps track of the current position in the array. You should also have a variable for each motor that keeps track of the motor direction.
    - For Motor #1, you will need six bits to control. 2 for the enables and 4 for control bits. For Motor #2, you will need 1 bit for enable (PE4) and you will notice that you already have bits in place to use the control pins (PM2-PM5) from the LCD code. You'll have to keep track of where you're writing the bits to but should be fine and do not need to add any beans to the project to write to these pins.
- Demonstrate to your TA.

In *Activity #1*, you used a time delay to control when the next step occurred. By changing the length of the delay, you are able to change the speed of the motors. Another technique used in stepper motor control is to use a timer interrupt to control the length of time between steps. A timer interrupt is an interrupt generated from an internal clock. For example, you can use a timer interrupt to toggle an output pin to an LED every 50 milliseconds.

Activity #2: Using a Timer Interrupt

- Insert two "TimerInt" beans from the Bean Inspector (one for each motor)
- Initialize both timer interrupts with a interrupt period of 4 milliseconds
- Using the timer interrupt event routines (in the Events.c file), use your code from *Activity 1* such that each time the timer interrupt occurs, the respective stepper motor will move one step.
- Demonstrate to your TA.

Activity #3: Write the following functions to control your motor.

- Move Forward
- Turn Left (90 degree turn)
- Turn Right (90 degree turn)

- Turn Around (180 degree turn)
- Demonstrate to your TA

Activity #4: At a time specified by your TA, you will have to demo your Micromouse running at full speed. You will run your Micromouse against your classmates to determine which lab group has the fastest mouse.

Questions

1. Develop a schematic showing the connections from the HCS12 microcontroller, motor drivers, and motors.
2. Stepper motors have two basic winding arrangements for the coils in a two phase stepper motor: unipolar and bipolar. Discuss the differences between unipolar and bipolar stepper motors
3. There are two primary options for selecting a motor when building a micromouse robot: a DC motor or stepper motor. Which motor would you choose for the micromouse application? Why?
4. HCS12 microcontrollers can drive small motors without a motor driver. What is the output current of a HCS12 microcontroller pin?
5. Describe how Pulse Width Modulation (PWM) can be utilized rather than timer interrupts.
6. What is a Proportional, Integral, and Derivative (PID) controller? Define each term in PID. In regards to designing a micromouse, why would you need something similar to a PID controller?

References

- http://en.wikipedia.org/wiki/Stepper_motor
- http://www.freescale.com/files/microcontrollers/doc/app_note/AN2974.pdf
- http://www.ams2000.com/stepping101.html
- http://www.datasheetcatalog.org/datasheet/texasinstruments/sn754410.pdf
- http://www.freescale.com/files/microcontrollers/doc/app_note/AN1285.pdf

Appendix E.6: Laboratory #6: Implementing PID Controller

Goals

This lab introduces the PID controller and discusses how to apply the PID controller in regards to the Micromouse application.

Background

PID Controller

Proportional, Integral, Derivative Controller (PID Controller) is a control loop feedback mechanism used in many control systems. PID controllers are one of the most widely used controllers today since it is fairly easy to tune the PID parameters without having much knowledge of control theory. A PID controller calculates the "error" value as the difference between a measured process variable and a desired set point [2]. To minimize the error, the controller adjusts the process control inputs and the controller is tuned according to the nature of the system.

A PID controller consists of three separate parameters: proportional, integral, and derivative. The proportional parameter (K_p) determines the controller's reaction to the current error, the integral parameter (K_i) determines the reaction based on the sum of the recent errors, and the derivative parameter (K_d) determines the reaction based on the rate at which the error has been changing [2]. K_p, K_d, and K_i, are the gains for each term. The weighted sum of these three parameters is used to adjust the process such as the position of a micromouse in a maze. *Figure 1* provides a block diagram of a PID controller.

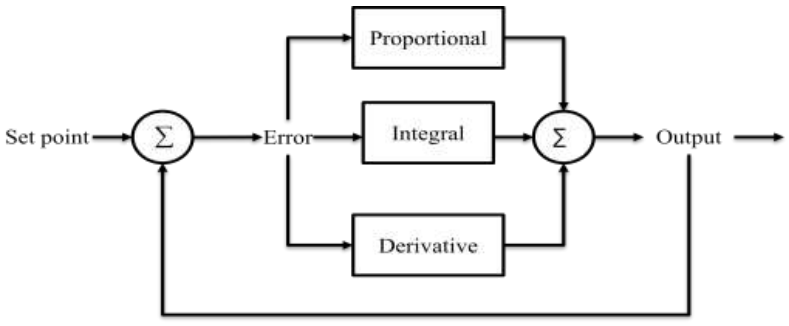


Figure 1: PID Controller Block Diagram

Proportional

The proportional parameter makes a change to the output that is proportional to the current error value. Essentially, you are multiplying a scalar (K_p) values times the error to adjust. So, a high proportional gain results in a large change to the output, and a small gain results in a small output response.

$$P_{\text{out}} = K_p\, e(t)$$

where

P_{out}: Proportional term of output

K_p: Proportional gain, a tuning parameter

e: Error = $SP - PV$

t: Time or instantaneous time (the present)

Figure 2: Proportional Term [2]

Integral

The integral parameter takes into consideration both the magnitude of the error and the duration of the error (by summing the error over time). The integral term with the proportional term increases the movement to the desired state (setpoint) and eliminates the steady-state error that is caused with only a proportional only controller [2]. However, since the integral term takes into consideration past errors, it does invoke the possibility to make the current adjustment overshoot the desired state.

$$I_{\text{out}} = K_i \int_0^t e(\tau)\, d\tau$$

where

I_{out}: Integral term of output

K_i: Integral gain, a tuning parameter

e: Error = $SP - PV$

t: Time or instantaneous time (the present)

τ: a dummy integration variable

Figure 3: Integral Term [2]

Derivative

The derivative parameter involves using the rate of change of the error to reach the destination state. One can take the slope of the error and multiply it by K_d. This term decreases the rate of change of the controller output to reduce overshooting the setpoint.

$$D_{\text{out}} = K_d \frac{d}{dt} e(t)$$

where

D_{out}: Derivative term of output

K_d: Derivative gain, a tuning parameter

e: Error = $SP - PV$

t: Time or instantaneous time (the present)

Figure 4: Derivative Term [2]

The PID controller output then takes each term and sums them together as shown in *Figure 5*.

$$\mathrm{u(t) = MV(t) = } K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{d}{dt} e(t)$$

Figure 5: PID Controller Equation

So, the PID controller can be modified for use in different systems by changing the gains of the different terms described above. *Table 1* captures the effects of increasing each of these parameters.

Table 1: PID Controller – Effects of Increasing Parameters

Effects of *increasing* parameters				
Parameter	**Rise time**	**Overshoot**	**Settling time**	**Error at equilibrium**
K_p	Decrease	Increase	Small change	Decrease
K_i	Decrease	Increase	Increase	Eliminate
K_d	Indefinite (small decrease or increase)[1]	Decrease	Increase	None

It is also important to note that there are several variations of PID controllers. Often times, systems only need to implement a few of the PID parameters to suitably correct the error. So, a PID controller may be called a PI, PD, P, or I controller when one of the respective control actions is not used in the current system. There are also several ways to tune the parameters of a PID controller such as mathematical models, software tools, and manually. Given that each

group only has a limited amount of time to finish this lab and the project, you will implement the PID algorithm and tune it by trial and error.

Micromouse

Due to the quality of the stepper motors, the robot does not have much trouble going in a straight line or even making a near perfect 90 degree or 180 degree turn. The stepper motors will rotate exactly the number of steps you need them to rotate. However, for our application, we saw the need to have some sort of feedback system in place to help find the center of the maze and help the robot go in a straight line (detect error's in its path) while it's moving through the maze. Using the distance measuring sensors as a feedback, the robot can keep track of its position and maintain a straight orientation in the maze to avoid from hitting the walls. To maintain the robot's orientation straight with reference to the wall is accomplished using a controller such as a PID controller. [1]

Examining our micromouse robot, we have two proximity sensors mounted on the left and right front of the robot. These sensors can be used to determine whether or not the robot is too close to the right wall or too close to the left wall. This calculation is the error for our controller and can generate the necessary signal to correct the error [1]. However, looking to the proximity sensor datasheet shows that it takes almost 55 milliseconds to complete the distance measuring operation. This time slows down our ability to fix the error in our control system and increases our chances to run into the wall especially if the micromouse is moving fast. As a result, we tailor the parameters of the PID controller to keep from hitting the walls while keeping in mind the time to perform the distance measuring operation.

A couple other things that you will want to consider in regards to the control problem are the stepper motor dynamics and single sensor readings. You will need to be careful when adjusting your stepper motor speed as stepper motors operate best within a certain range. If you go outside of this range, the stepper motor may become jerky or not respond correctly. In regards to the sensors, since there is only one sensor on each side of the robot, we are prone to sensor reading errors.

Let's take a look at an example scenario.

Suppose the robot begins to move towards the left wall instead of going straight through the maze. Based on the sensor readings, an error signal should be generated. Using this error, the controller should generate a control signal which increases the speed of the left wheel and decreases the speed of the right wheel which will make the robot turn slightly in the desire direction to straighten its path. The amount of the increase and decrease of each wheel is controlled by the magnitude of the control signal [1]. *Figure 6* provides an illustration of how to correct the robot's movement.

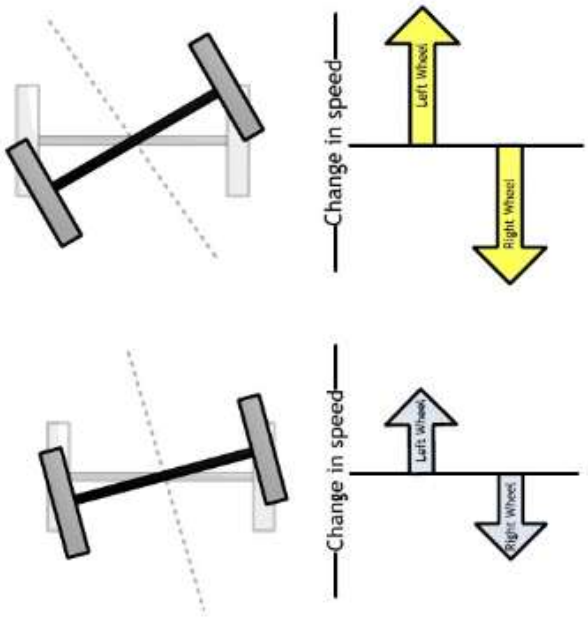


Figure 6: Adjusting Speed when in Error to Straighten Micromouse [1]

Now, pulling all of this together, we have the system as shown in *Figure 7.*

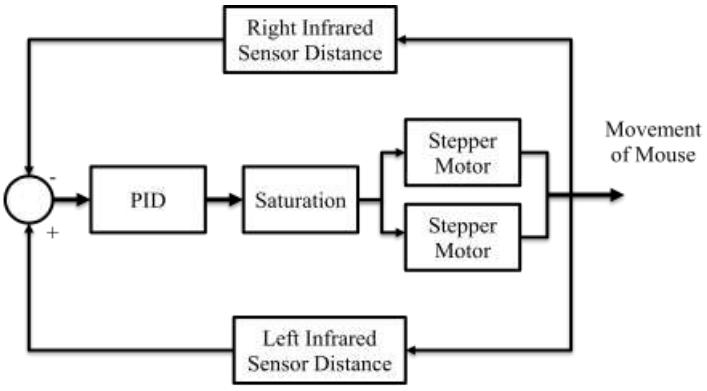


Figure 7: Micromouse Control Loop Diagram [1]

Procedure

Your task is to develop a control loop inside of a timer interrupt subroutine that is executed every 55 milliseconds. The 55 millisecond control loop is limited by the proximity sensors as discussed above. The subroutine should read the sensor data from all three sensors and can optionally convert the data into centimeters. The subroutine should use the distances to make the robot go straight with respect to the walls. For the purposes of this experiment, you will just have to keep the robot between two walls (a straight line traversing 13 cells) – you will not have to worry about turning and your program should be organized similar the flowchart in *Figure 8.*

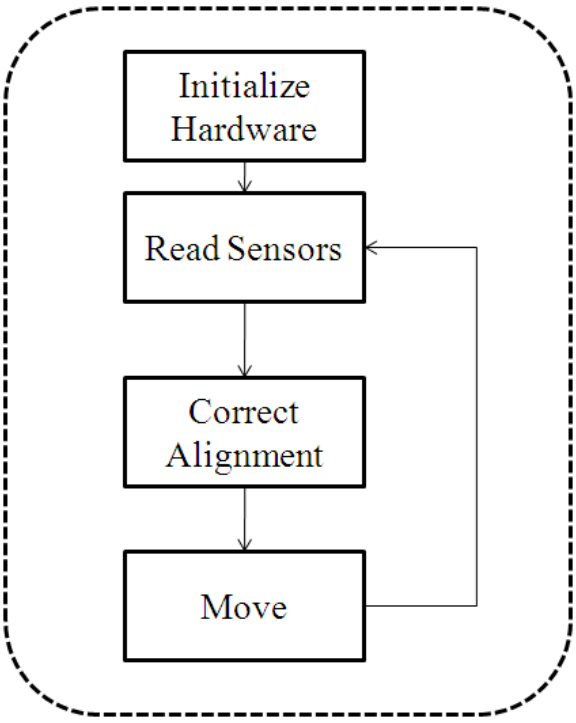


Figure 8: Program Flowchart

- **Correct Alignment Action**
  - Initially, both motors should be driven at the same speed
  - Once the robot is not centered, an error should be sent to the controller
  - Based on the magnitude of the error, the controller should generate a control signal to make the mouse turn slightly left or slightly left. You can make these adjustments by increasing the speed of one wheel which decreasing the speed of the other.

First, implement a Proportional controller (P controller) using your left sensor and right sensor readings. We can calculate the proportional error by performing the following calculation shown in *Figure 9.*

$$Proportional\_error = \frac{Left\ distance - Right\ distance}{2}$$

Figure 9: Proportional Error Calculation [1]

Using this equation, you notice that if the error is negative, the robot is moving towards the left, and if the error is positive, the robot is moving to the right.

Let's take a look at an example.

Looking at *Figure 10*, we note that the left sensor distance is 6" and the right sensor distance is 2", and thus we consider this an error. We can calculate the error by using our equation above.

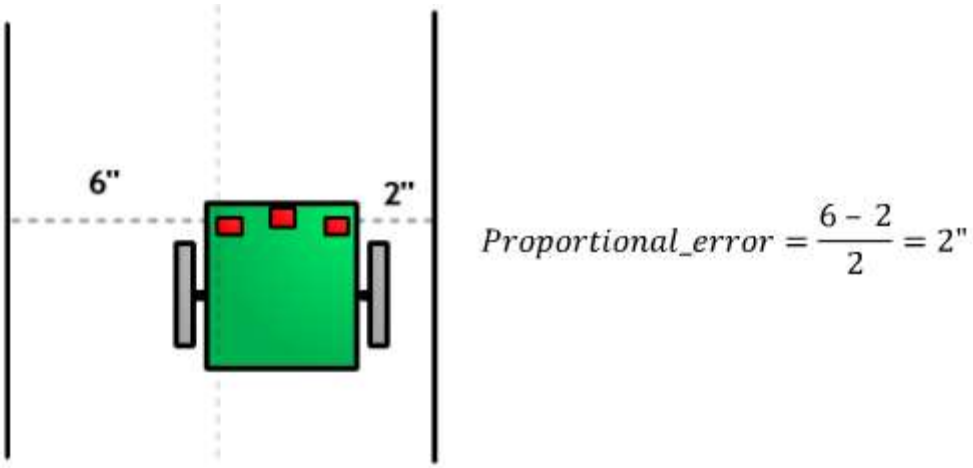


Figure 10: Determining Error using Sensor Values [1]

For the proportional term, you can take this error and multiply it by the proportional gain (K_p). Demonstrate your results to your TA.

Second, now try to improve your controller by adding the integral term to the control algorithm to create a PI controller. The integral term is proportional to the magnitude of the error and the duration of the error. When finished, demonstrate your results to your teaching assistant.

Finally if necessary implement the derivative term to your control algorithm to create a PID controller. Demonstrate your results to your TA.

Questions

1. Clearly indicate your PID code and highlight your values for the gains K_P, K_I, and K_D.
2. What did you notice when you were using just a proportional controller? What are some advantages and disadvantages of a proportional controller?
3. What did you notice when you added the derivative controller? What are some advantages and disadvantages of the proportional-derivative controller?
4. Revisit your code from Lab #1 on initializing the wall map and distance values. Then, research the modified flood fill algorithm and describe how the algorithm works in your own words.
5. By putting together everything you learned throughout the semester, you now have the skills you need to develop your own micromouse. Develop an outline for your final project indicating the following.
    - Functions that will be called to perform each action within the flowchart
    - References for each action within the flowchart
    - Percent Complete for each function within the flowchart
    - Identify the areas that you believe will be the most difficult
    - Keeping in mind the time remaining to complete the final project, come up with a brief schedule of what you need to have accomplished each week to have a working micromouse by the end of the semester.
6. Design your controller in Matlab/Simulink. Turn in your completed code and a brief write-up on your design including all references used.

References

- http://www.scribd.com/doc/16400768/Ishu-Pradhan-Implementing-PD-controller-in-a-Robot-Micro-mouse. A good portion of the text in this lab experiment handout was derived from this reference.

Appendix E.7: MICROMOUSE PROJECT DESCRIPTION

[Modified from IEEE Region 2 Student Activities Conference 2007 Micromouse Competition Rules]

Objective

The objective is to program your robot to negotiate a 6x13 maze from the bottom, left corner of maze to its center in the shortest amount of time.

Micromouse Rules

1. Micromouse must be self-contained (no remote controls).
2. Micromouse shall not leave any part of its body behind while solving the maze.
3. Micromouse shall not jump over, fly over, climb, scratch, cut, burn, mark, damage, or destroy the walls of the maze.

Maze Rules

1. The maze is composed of multiples of 18 cm x 18 cm unit square. The maze comprises of 6x13 unit squares. The walls of the maze are 5 cm high and 1.2 cm thick. Assume 5% tolerance for all maze dimensions. The outside wall encloses the entire maze.
2. The sides of the maze walls are white, the tops of the walls are white (red in actual competition), and the floor is black. The maze is made of wood, finished with non-gloss paint.
    a. Do NOT assume that the walls are consistently white or floor is consistently black. Fading may occur. Do not assume that the floor provides a certain amount of friction. The maze floor is constructed of a single piece of plywood.
3. The start of the maze is located at the bottom left corner of the maze. The start square is bounded on three sides by walls. The start line is located between the first and second squares. That is, as the mouse exits the corner square, the time starts. The destination goal is the six cells at the center of the maze. The destination cell only has one entrance. *Figure 1* provides an illustration showing the starting cell highlighted gray and the destination cells highlighted yellow.

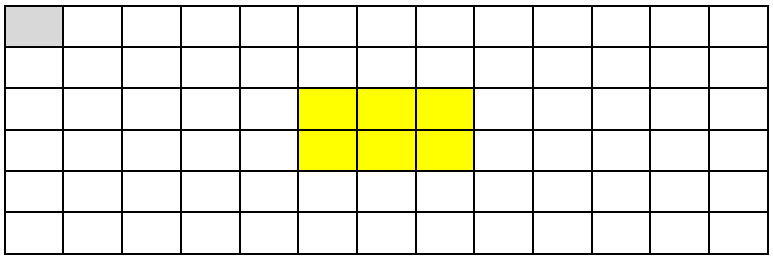


Figure 1: 6x13 Maze

4. Small square zones (posts), each 1.2 cm x 1.2 cm, at the four corners of each unit square are called lattice points. The maze is so constituted that there is at least one wall at each lattice point.
5. Multiple paths to the destination square are allowed.

Contest Rules

1. Each lab group is allocated a total of 10 minutes to access the maze from the moment the teaching assistant acknowledges the lab group and grants access to the maze. Any time used to adjust a mouse between runs is included in the 10 minutes. Each run (from the start cell to the center zone) in which a mouse successfully reaches the destination square will be given a run time. The minimum run time shall be the mouse's official time. First place goes to the mouse with the shortest run time. Mice that do not enter the center square will be ranked by the maximum number of cells they consecutively transverse without being touched. All mice that enter the center square within their 10 minute allotment are ranked higher than those that don't enter the center square.
2. Each run shall be made from the starting square. The operator may abort a run at any time. If the operator touches the micromouse during a run, it is deemed aborted, and the mouse must be removed from the maze. If a mouse has already crossed the finish line, it may be removed at any time without affecting the time of that run. If a mouse is placed back in the maze for another run, a one-time penalty of **30 seconds** will be added to the mouse's next run time.
3. After the maze is disclosed, the operator shall not feed information on the maze into the micromouse; however, switch positions may be changed.

4. The illumination, temperature, and humidity of the room shall be those of an ambient environment.
5. The run timer will start when the front edge of the mouse crosses the start line and stops when the front edge of the mouse crosses the finish line. The start line is at the boundary between the starting unit square and the next unit square clockwise. The finish line is at the entrance to the destination square.
6. Every time the mouse leaves the start square, a new run begins. If the mouse has not entered the destination square, the previous run is aborted.
7. The mouse may, after reaching the destination square, continue to navigate the maze, for as long as their maze time allows.
8. If a mouse continues to navigate the maze after reaching the destination square, the time taken will not count toward any run. Of course, the 10-minute timer continues to run. When the mouse next leaves the start square, a new run will start. Thus, a mouse may and should make several runs without being touched by the operator. It should make its own way back to the beginning to do so.
9. A contestant may not feed information on the maze to the Micromouse. Therefore, changing ROMs or downloading programs is NOT allowed once the maze is revealed. **However, contestants are allowed to**:
    a. Change switch settings (e.g. to select algorithms)
    b. Replace batteries between runs
    c. Adjust sensors
    d. Change speed settings
    e. Make repairs
10. However, lab groups may not alter a mouse in a manner that alters its weight (e.g. removal of a bulky sensor array or switching to lighter batteries to get better speed after mapping the maze is not allowed). The TA shall arbitrate.

Lab Rules

1. Each lab group will program their own micromouse.
2. All maze solving algorithms are allowed even though the one taught in the course is the modified flood fill algorithm.

Appendix F: Additional Laboratory Modules

This appendix includes an additional set of standalone laboratory modules to support the learning objectives of the microcomputer structures and interfacing undergraduate laboratory that were developed as part of this effort for DC motors, Zigbee wireless communications, I2C communications, and servo motors. These modules are intended to be utilized during any week of a given semester (intended to align with the instructional course CPE 312).

Appendix F.1: Keypad Interfacing

Objective

The primary objective for this laboratory is for students to develop an understanding of how to interface with a given keypad using the HSC12 microcontroller. Students will learn how to use the 4x4 keypad and how to communicate with the microcontroller. The students are expected to be able to successfully interface the keypad with the microcontroller and be able to implement it in a practical application.

Background

Using the 4x4 Keypad

The first thing you will notice with the 4x4 keypad is that there are 8 pins. As a result, each key is not directly correlated with only one of the pins. To determine the button that is being pressed you must determine the row and column. *Table 1* illustrates how the rows and columns correspond to the pressed key. *Example: (1,1) corresponds to the first column and the first row of the keypad.*

(1,1)=1	(1,2)=2	(1,3)=3	(1,4)=A
(2,1)=4	(2,2)=5	(2,3)=6	(2,4)=B
(3,1)=7	(3,2)=8	(3,3)=9	(3,4)=C
(4,1)=*	(4,2)=0	(4,3)=#	(4,4)=D

Table 1: Row and Column configuration (Row, Column)

In order to determine the row of a button that is pressed, you must receive input from the low nibble of Port T. *Figure 1* provides the possible received values. To do this you must first set the appropriate direction of Port T (you will only want to receive input from the low nibble). Then, wait 100 us. Finally, you must receive input from the full port and filter out the high nibble (which you will not use).

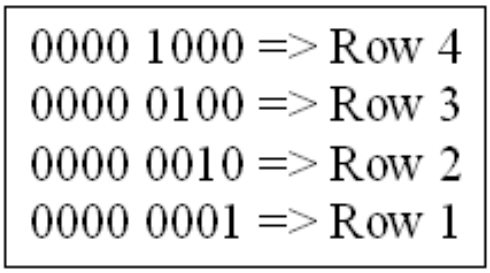


Figure 1: Low Nibble - Possible Received Values

Secondly, in order to determine the column of a button that is pressed, you must receive input from the high nibble of Port T. *Figure 2* provides the possible received values. To do this you must first set the direction of Port T (you will only want to receive input from the high nibble). Then, wait 100 us. Finally, you must receive input from the full port and filter out the low nibble (which you will not use).

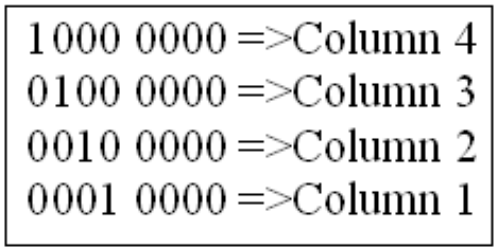


Figure 2: High Nibble - Possible Received Values

In this lab, you must also handle the case when no buttons are pressed. A row or column input of 0 means that no buttons are pressed.

Ports

Port T will be used to interface with the keypad. You will have to define alternate ports to be used for additional inputs or outputs. Refer to the CSM12C32 manual to choose a port.

4x4 Keypad Initialization Code

The 4x4 keypad must be initialized before using it. Be sure that to select pull-down resistors, resistors used to hold the input low when no other component is driving the input (or pull-up

resistors) for the appropriate port when using the keypad. The only power supplied to the keypad is that which is supplied using the output port, thus the pull-down resistors are necessary to ensure that the input port pins don't have floating values. Using pull-down resistors, the input ports will be at "0" when no key is pressed and a "1" written on the output port can be detected. The low nibble initially should be set as an output and the high nibble must be initially set an output. Both nibbles (high and low) should be initialized with high inputs/outputs (use Processor Expert to ensure that PTT = 0xFF).

Delays

In this experiment, you will need to implement delays. To implement a delay in Processor Expert, you can use the Delay100US method found in the Processor Expert window using the Cpu selected for the current project. *Figure 3* provides an illustration of where this is located in CodeWarrior.

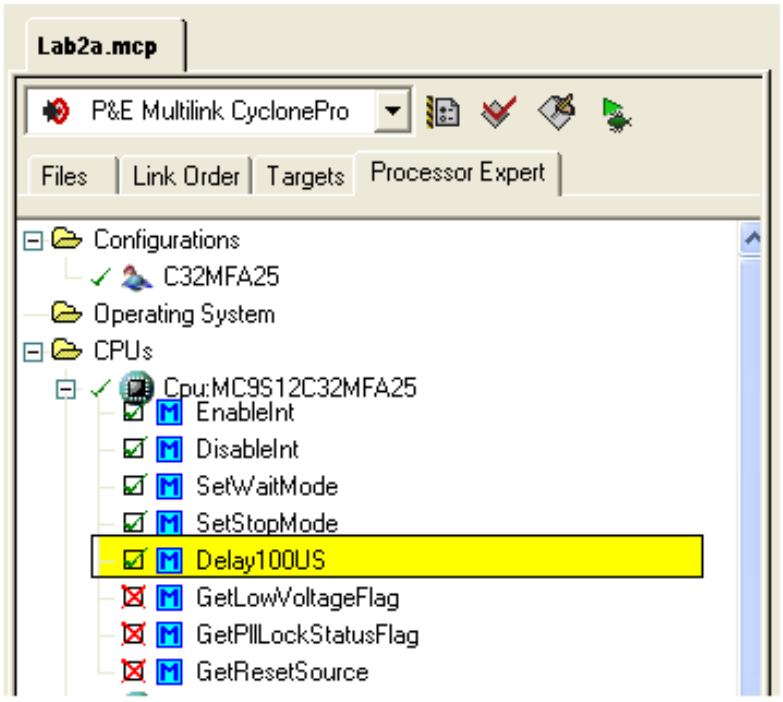


Figure 3: Using a CPU Delay

Experiment

Hexadecimal Representation of the Keypad

Assign 4 bits to be used as output to LEDs. You will have to choose another port besides Port T for these outputs. Represent the hexadecimal value from the keypad using a 4 bit binary representation. Treat the * as E, # as F and 0 as 0. This means that no LEDs will light up when button "0" is pressed.

Questions

1. What is a SPI? What are some applications of a SPI?
2. If an SPI is operating in master mode, meaning that the MSTR bit is set, what values must the MODFEN and SSOE bits have to operate with the slave select as an output. Hint: Refer to the Serial Peripheral Interface Block Description document.
3. What Processor Expert bean and method can be used to generate a delay? What parameter value is required to generate a 250 ms delay?
4. Why is it necessary to use pull-down resistors on the input ports?
5. Please read Part 2 of this lab before next week.

References

1. Processor Expert Online. UNIS, Ltd. 14 January 2007. <http://www.processorexpert.com/>.

Appendix

V_+	1	2	PE1/IRQ*
GND	3	4	RESET*
PS1/TXD	5	6	MODC/BKGD
PS0/RXD	7	8	NC
PP5/KWP5	9	10	NC
PE0/XIRQ*	11	12	NC
PT0/PW0/IOC0	13	14	NC
PT1/PW1/IOC1	15	16	NC
PM4/MOSI	17	18	PAD00/AN00
PM2/MISO	19	20	PAD01/AN01
PM5/SCK	21	22	PB4
PM3/SS*	23	24	PA0
PE4/ELCK	25	26	PM1/TXCAN
PE7/XCLKS	27	28	PM0/RXCAN
PAD02/AN02	29	30	PT2/PW2/IOC2
PAD03/AN03	31	32	PT3/PW3/IOC3
PAD04/AN04	33	34	PT4/PW04/IOC4
PAD05/AN05	35	36	PT5/IOC5
PAD06/AN06	37	38	PT6/IOC6
PAD07/AN07	39	40	PT7/IOC7

Figure A.1 MCU Port Connector

Appendix F.2: Serial Communications Interface

Objective

The purpose of this lab is to introduce students to the serial communications interface (SCI). Students will learn how to use Processor Expert to initialize and implement the SCI. The lab will also require students to use the communication program, HyperTerminal, to emulate a text terminal.

Background

Serial Communications Interface (SCI)

The SCI is a communications device that allows data exchange between a microprocessor and peripherals such as printers, scanners, or mice [3] and can be used to communicate with other microprocessors. Some of the features provided by the SCI are as follows: full-duplex operation, programmable 8-bit or 9-bit data format, separately enabled transmitter and receiver, hardware parity checking, etc. The SCI has three modes of operation: run, wait, and stop. Refer to the SCI Block Description (Chapter 13 in Motorola Datasheet) for more information.

HyperTerminal

HyperTerminal is a communications program which can be configured to connect using a modem or directly over a serial port [2]. HyperTerminal allows the following parameters to be configured: baud rate, parity, stop bits, and flow control. To access HyperTerminal, go to Start > All Programs > Accessories > Communications > HyperTerminal. *Figure 1* provides a screenshot of HyperTerminal.

Figure 1: HyperTerminal

Experiment

Part #1: Echo keypad input on HyperTerminal

1. Create a new processor expert project.
2. Select the AynchroSerial embedded bean using the bean selector. This bean is a CPU Internal Peripheral used for communication.
3. Using the Bean Inspector, configure the bean to operate at 9600 baud, 8 data bits, and 1 stop bit. (HyperTerminal should also be configured using the same parameters as port settings). Make the appropriate changes to generate an interrupt when a character is received.
4. After choosing the appropriate settings, generate the code.
5. Write code in the main routine and OnRxChar event to receive a character from the computer's keyboard and display it back on HyperTerminal.
6. After writing the necessary code, open up HyperTerminal as described in the *Background* and *Step 3*. Run the program. You should be able to enter characters from the computer's keypad, and they should output on HyperTerminal.
7. Now, using ASCII character codes (http://www.asciitable.com/), handle the following cases:
    a. When ENTER is pressed, go to the beginning of a new line
    b. When BACKSPACE is pressed, ring the bell.

Questions

1. If you were going to transmit in 9-bit format, does it matter which data register you read first? (Hint: Refer to SCI Block Guide)
2. What is non-return-to-zero (NRZ) encoding?
3. Illustrate the data format when the following information is sent using the SCI.

    Byte: 10011011, 1 stop bit, 1 stop bit.
4. In *Lab #2*, you were required to display characters from a keypad onto an LCD. Using the functions (writeData, writeCommand, and the CpuDelay100US) from *Lab #2* and *Figures A.2 and A.3* in the *Appendix*, write the necessary code to display text from a computer keyboard to the LCD. Also, write the code to do the following:

a. When ENTER is pressed, clear the LCD screen
b. When BACKSPACE is pressed, delete the previous character and move the cursor backwards.

5. To simplify circuit construction, the MCU Project board has several user features that have been connected to the MCU_PORT through FET switches. The FET switches are controlled by enable signals that are also routed to the MCU_PORT header. This setup allows the user to electronically connect and disconnect each connected feature group. The buzzer and potentiometer on the bottom left of the MCU Project Board are examples of the connected features.
   a. What do you need to do to enable the potentiometer on the MCU Project Board?
   b. What do you need to do to enable the buzzer on the MCU Project Board?

Appendix

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Figure A.1 ASCII Codes [5]

Appendix F.3: Alarm Clock

Objective

The main objective of this lab is to create a simple alarm clock using the MCU features, buzzer, and HyperTerminal. Students will learn how to generate a timer interrupt, gain an understanding of the connected features on the MCU project board such as the buzzer, and gain a further understanding of the Serial Communications Interface (SCI) and related Processor Expert methods.

Background

MCU Project Board Buzzer

The buzzer for the MCU project board 2 is a connected feature of the MCU Project Board and thus is connected to pin 13 via field effect transistors. The buzzer is enabled by connecting the jumper on the MCU Project Board at JP10-1. After enabling the buzzer, you can utilize it by generating a square wave output from Port T, Pin 0 (pin 13) as shown in *Figure 1*.

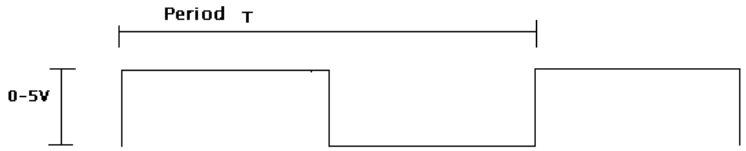


Figure 1: Square Wave with Period T

Example: So if you would like to generate a pitch, you would be concerned with selecting the correct frequency to represent that pitch. The period (T) of your waveform is the inverse of that frequency in Hz or cycles per second. The delay of your function will represent the amount of time you would like to hold a value without changing. Thus, the delay would be approximately half of the period.

Timer Interrupt

A timer interrupt can be used to indicate the need for attention or a synchronous event in the software indicating the need for a change in execution or indicating that something should be

done. In this lab, we can utilize a timer interrupt by using the TimerInt bean. Using this bean, we can enable and disable a timer and generate events after certain time delays.

Experiment

Goal: *Design a simple alarm clock that can be set via HyperTerminal from 1-9 seconds. Once the alarm clock is set, the buzzer should sound until the user turns it off via a switch on the MCU project board.*

Example Scenario:

Execute the program

HyperTerminal Displays "Please enter alarm time (1-9s):"

User enters: 9

9 Seconds pass

The buzzer sounds until turned off via a switch

HyperTerminal Displays "Please enter alarm time (1-9s):"

Questions

1. If you would encounter a hardware or software problem with the lab, please describe the necessary steps that you would take to try to resolve the problem.
2. Look at the SCI Block Guide on the class website. Fill out the following table.

Register	Value for this bit = 0	Value for this bit = 1
SCICR1 bit 7		
SCICR1 bit 6		
SCICR1 bit 5		
SCICR1 bit 4		
SCICR1 bit 3		
SCICR1 bit 2		
SCICR1 bit 1		
SCICR1 bit 0		

Copy each phrase exactly from below into the table above.

- 8 bit data format
- 9 bit data format
- Count idle bits after start bit
- Count idle bits after stop bit
- Disable loop operation (Normal operation)
- Disable SCI in wait mode
- Enable loop operation
- Enable SCI in wait mode
- Even parity
- Odd Parity
- Parity disabled
- Parity enabled
- Receiver connected externally to transmitter
- Receiver internally connected to transmitter output
- Wake on address mark
- Wake on idle line

3. Take a look at the datasheet of the DS1624 thermometer at **http://datasheets.maxim-ic.com/en/ds/DS1624.pdf** and briefly describe its operation.
4. Describe the concept of the Inter-Integrated Circuit Bus (I^2C) and explain how it may be useful when interfacing with microprocessors.

References

1. http://www.freescale.com/files/microcontrollers/doc/app_note/AN2949.pdf
2. MCU Project Board-2→http://www.csee.wvu.edu/classes/cpe313/MCUPB2_SCH_C.pdf

Appendix F.4: I2C and DS1624 Digital Thermometer

Objective

The purpose of this lab is to teach students how to implement Inter Integrated Circuit (I^2C) external communications on the microcontroller unit to read a DS1624 digital thermometer. Students will learn how to use Processor Expert to properly initialize the I^2C embedded bean and use the bean's methods. The lab utilizes the serial communications interface for a display.

Background

Inter-Integrated Circuit Bus (I^2C)

I^2C is a serial and synchronous bus protocol that uses two bidirectional lines, serial data (SDA) and serial clock (SCL). The protocol generally has one master and multiple slaves, and the main difference between the master and slave is that the master generates the clock pulse [1]. The master is usually the microcontroller. I^2C designs can either utilize 7 or 10 bit addressing with some reserved addresses. For example, with 7 bit addressing, there are 16 reserved addresses leaving room for 112 nodes on the same bus. *Figure 1* is a schematic of a typical hardware configuration. In this lab, the microcontroller will be configured as the master and will use 7 bit addressing.

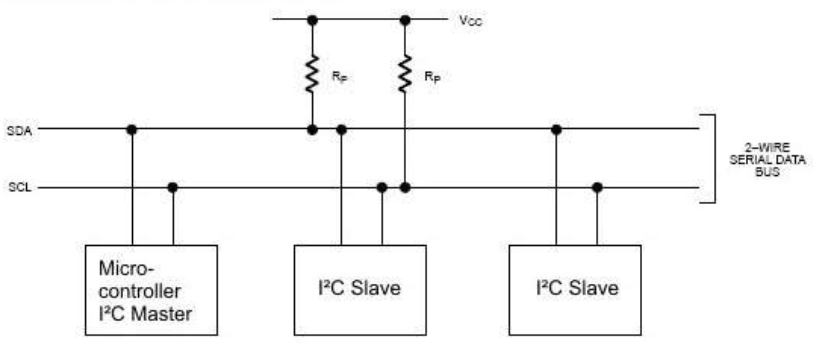


Figure 1: Typical 2-Wire Bus Configuration [1]

DS1624 Digital Thermometer and Memory

The DS1624 consists of a digital thermometer and 256 bytes of E^2 memory [3]. The thermometer measures temperatures from -55 degrees Celsius to +125 degrees Celsius in 0.03125 degrees Celsius increments, whereas the temperature values are read as a 13-bit value

(two byte transfer) [3]. *Figure* 2 illustrates the two byte transfers. **The MSB is read first and then the LSB**.

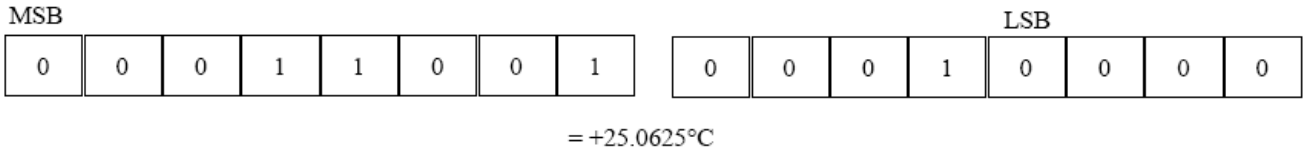


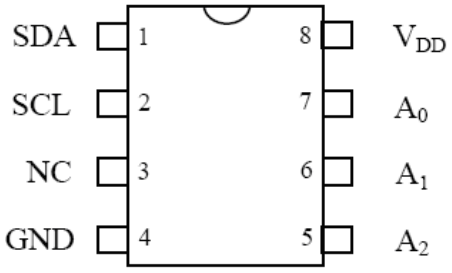
Figure 2: DS1624 Temperature Representation [3]

Figure 3 provides some temperature/data relationships.

TEMP	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125˚C	01111101 00000000	7D00h
+25.0625˚C	00011001 00010000	1910h
+½˚C	00000000 10000000	0080h
+0˚C	00000000 00000000	0070h
-½˚C	11111111 10000000	FF80h
-25.0625˚C	11100110 11110000	E6F0h
-55˚C	11001001 00000000	C900h

Figure 3: Temperature/Data Relationships [3]

The data is read/written using a 2-wire serial interface (open drain I/O lines) using the SDA and SCL lines. *Figure 4* provides the pin assignments and descriptions for the DS1624.



SDA - 2-Wire Serial Data Input/Output
SCL - 2-Wire Serial Clock
GND - Ground
A0 - Chip Address Input
A1 - Chip Address Input
A2 - Chip Address Input
V_{DD} - Digital Power Supply (+3V - +5V)
NC - No Connection

Figure 4: DS1624 8-Pin DIP (300 MIL) and Pin Description [3]

Figure 5 shows the master's (microcontroller's) bus activity to write to and read from the DS1624.

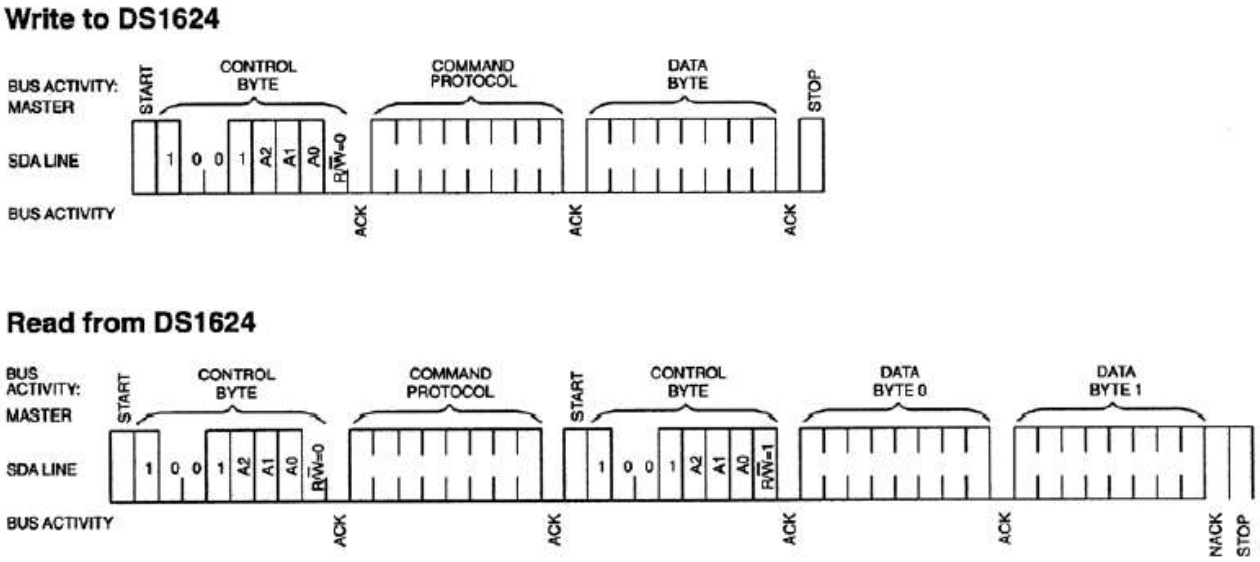


Figure 5: 2-Wire Communication with the DS1624 [3]

The complete datasheet for the DS1624 is located on the class website and at http://datasheets.maxim-ic.com/en/ds/DS1624.pdf. If unclear about the operation of the DS1624, please refer to this document.

Experiment

Read Temperature Sensor and Display on HyperTerminal using SCI

Goal: Utilize the I^2C protocol to communicate with the DS1624. HyperTerminal will be used to control when the temperature sensor should be read. Pressing a key from HyperTerminal should initiate a transfer from the DS1624 and display the results back on HyperTerminal. For example, if I press the "T" key on the computer's keypad, a temperature value in degrees Celsius should be displayed on HyperTerminal. The temperature only has to be displayed with a temperature resolution of 1 degree Celsius.

1.) Initialize the SCI Internal Communication device to be able to communicate with the HyperTerminal program.

2.) Initialize the CPU External Communication Device – SW_I2C using the Bean Inspector.

From the DS1624 datasheet, the 4 MSB bits of the initialization address are 1001.

3.) Generate code using Processor Expert.

4.) Using the embedded beans methods, accomplish the goal stated above. Be sure to look through the methods, understand what they do, and utilize your best options. To accomplish this goal, the *Memory Function Example* and *Command Set* provided in the DS1624 datasheet can be implemented. These portions of the datasheet are inserted in Appendix A of this handout for mere convenience.

Note: Writing to the configuration register requires approximately 10 ms at room temperature. Thus after issuing a write command, no further reads or writes should be requested for at least 10 ms [3].

Questions

1. What would be the 7-bit address needed to initialize the slave address if A0 = +5V, A1 = +5V, and A2 = 0 V? What is the purpose of the 8^{th} bit (R/W)?
2. How would the DS1624 output a temperature value of -33 degrees?
3. When you issued the command sequence to set the DS1624 for continuous conversion, you accessed the configuration/status register. What byte value would you send to the DS1624's configuration to make the device perform only one temperature conversion?
4. Provide some examples of analog devices and digital devices.
5. In your own words, describe ZigBee. What are some potential applications for ZigBee?

References

1. Using the I^2C Protocol
   http://avrhelp.mcselec.com/bascom-avr.html?Using_the_I2C_protocol
2. Motorola Datasheet. Motorola.
   http://www.csee.wvu.edu/classes/cpe313/motoroladatasheet.pdf
3. DS1624 Digital Thermometer and Memory
   http://datasheets.maxim-ic.com/en/ds/DS1624.pdf

Appendix F.5: Square Wave Frequency Calculation

Objective

The objective of this lab is to introduce the capture Processor Expert embedded bean. You will create a program to calculate & display period and frequency of arbitrary square wave.

Background

Capture Embedded Bean

The Capture bean is simply a capture function of a timer. The counter counts in free run mode. Using this bean, one can capture the input signal (on the input pin) on a selected edge (either the rising or falling edge). The capture bean captures values from a 16-bit, programmable counter.

Experiment

Calculate the period and frequency of an inputted square wave

Using the function generator on the NI-ELVIS, generate a 50 Hz square wave and wire it to an appropriate pin on the MCU Port. The output from the function generator is wired to a pin on the top left on the MCU Project Board (SYNC OUT). You can control the amplitude and frequency of the wave by using the dials on the front of the NI-ELVIS. Create a program using the Capture embedded bean to calculate the period and frequency of the square wave. Display this value using HyperTerminal either periodically or by pressing a key on the keyboard. Change the frequency of the square wave on the NI-ELVIS and display a variety of periods and their respective frequencies. You could confirm your displayed values by viewing the square wave on the oscilloscope.

Questions

1. How do you reset the counter register using the capture embedded bean? Please give the correct syntax.
2. How many channels are on the timer module (TIM)?
3. What is a real-time clock?
4. What is a DC motor? Be sure to list the main components.
5. Why is it necessary to have an H-bridge to drive the motor?

Appendix F.6: DC Motor Control using PWM

Objective

The objective of this lab is to introduce the concept of pulse width modulation (PWM), a function included on many microcontrollers to simplify the task of waveform generation. In this lab, students will use Pulse Width Modulation (PWM) to control the speed and rotational direction of a DC motor. Along with using the PWM characteristics of CodeWarrior, the students will also get hardware experience in using the L293B chip, a four channel driver that is commonly used in controlling DC motors.

Background

Pulse Width Modulation (PWM)

PWM of a signal involves modulating a pulses duty cycle whereas duty cycle **D = t/T**.

t = pulse duration

T = pulse period

PWM is used to transmit information over a communications channel or control amount of power sent to a load. *Figure 1* illustrates a signal with a duty cycle D and pulse duration T.

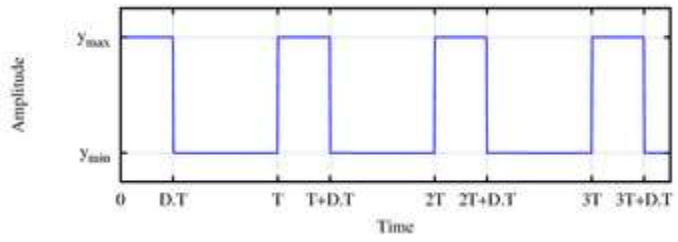


Figure 1: PWM Signal

DC Motor

DC Motors are used extensively in control systems as positional devices because their speeds and their torques can be precisely controlled over a wide range. The DC motor has a permanent magnetic field and its armature is a coil. When voltage and a subsequent current flow are applied to the armature, the motor begins to spin. The voltage level applied across the armature determines the speed of rotation.

One means to control a DC motor is to vary the pulse width of a digital signal input to the motor. By varying the pulse width, the average voltage delivered to the motor changes and so does the speed of the motor. The HCS12 PWM system can be used to control the DC motor. The HCS12 interfaces with a DC motor through a driver, as explained in the next section. The pin that controls the direction can be an ordinary I/O pin, but the pin that controls the speed must be a PWM pin.

L293B Push-Pull Four Channel Driver

Although some DC motors can operate at 5 V or less, the HCS12 cannot supply the necessary current to drive a motor directly. The minimum current required by any practical DC motor is much higher than any microcontroller can supply. Depending on the size and rating of the motor, a suitable driver must be selected to take control signals from the HCS12 and deliver the necessary voltage and current to the motor. One of these driver chips is the 16-pin four-channel L293B. This chip has four separate channels that can drive a total of one amp each. *Figure 2* provides a pin-out of the L293B. The Chip Enable 1 controls channels one and two, and Chip Enable 2 controls channels three and four.

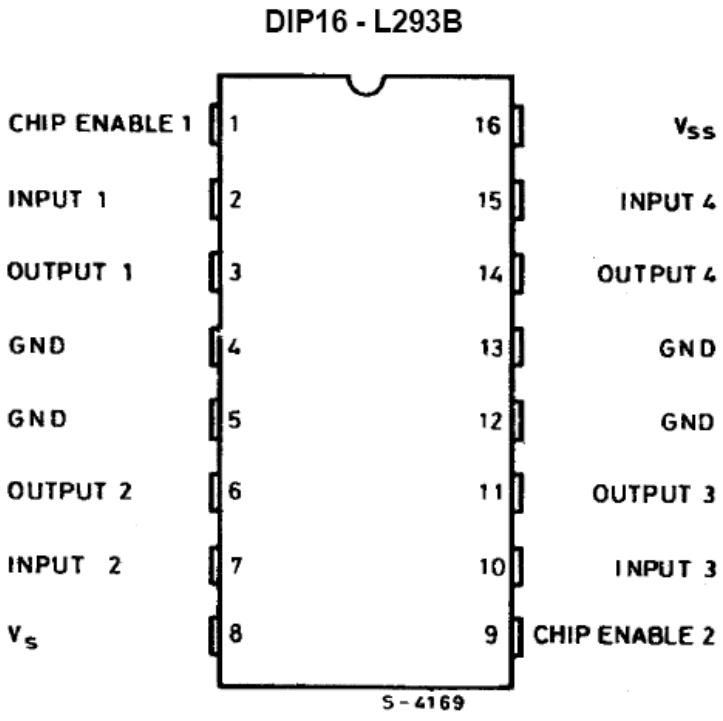


Figure 2: Pin-out of the L293B

Figure 3 provides a block diagram from the L293B datasheet showing the different pulses needed to create single direction and bi-directional motor controls. For the single directional portion of the lab assignment, please use channel #4. As mentioned above, pin #9 is the chip enable for the right side of the chip. This pin can be controlled by any basic I/O pin on the HSC12 microcontroller. Pin #15 accepts the PWM signal from the microcontroller to control the speed of the motor, and pin #14 is the output signal from the motor driver to the DC motor. Also notice that pins #4, #5, #12, and #13 must be grounded. You should use a separate power source to drive the motor than what is used to power the chip – this means that Vs (VCC) (pin #8) should be connected to a different supply than Vss (pin #16). Finally, be sure to pay attention to the waveform shown in the top right of *Figure 3* for it is important to generate the waveform as shown to correctly drive the motor (low → high).

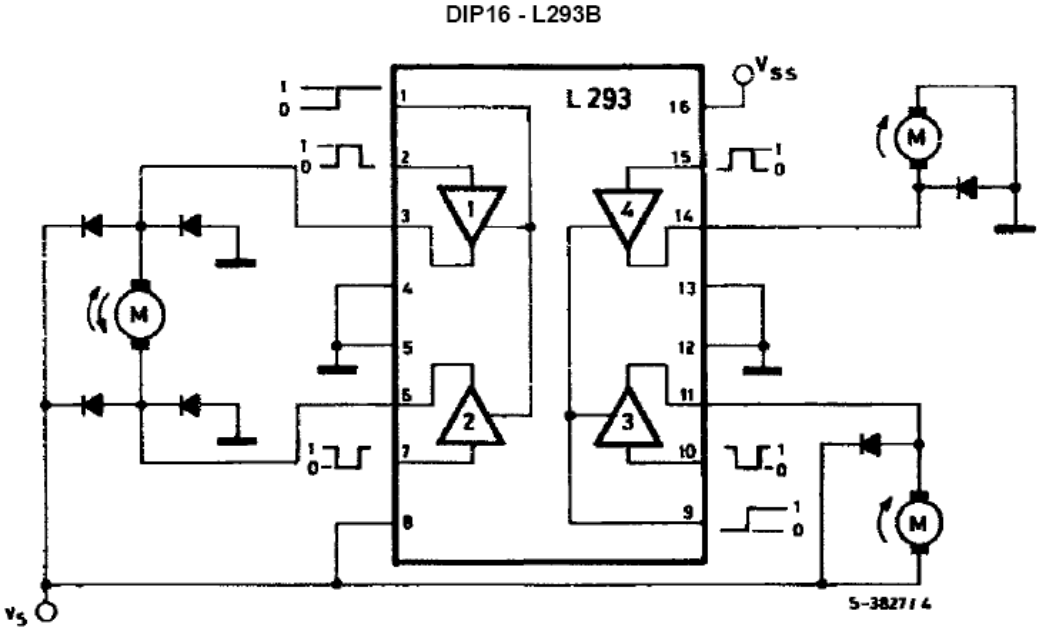
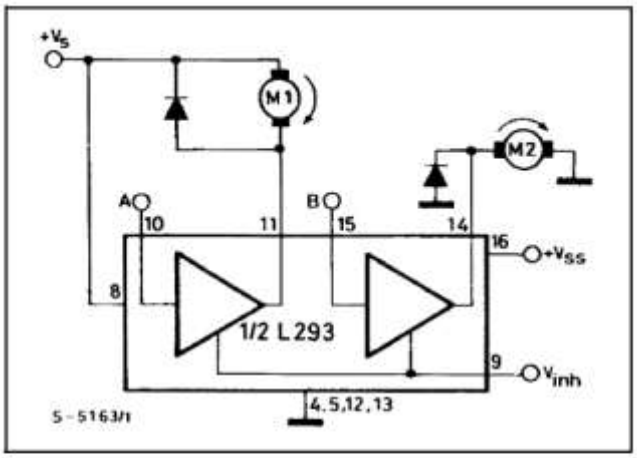


Figure 3: PWM Pulses to control

NOTE: In *Figures 3,* there are diodes used in each circuit. The diodes are EXTREMELY important, because we must control the direction of the current so we do not destroy the driver chip or microcontroller. Remember the line on the diode represents the negative side of a diode.

Figure 4 provides an additional diagram from the L293B motor driver datasheet that may help in this experiment.



V_{inh}	A	M1	B	M2
H	H	Fast Motor Stop	H	Run
H	L	Run	L	Fast Motor Stop
L	X	Free Running	X	Free Running
		Motor Stop		Motor Stop

L = Low H = High X = Don't Care

Figure 4: DC Motor Controls (w/ connection to ground and to the supply voltage)

Bypass Capacitor

A bypass capacitor (decoupling capacitor) is a capacitor used to decouple one part of an electric circuit from another. One common kind of decoupling is of a powered circuit from signals in the power supply as shown in *Figure 5.* The charged capacitor helps fill in any 'dips' in the voltage VCC (Vs, in our case) by releasing its charge when the voltage drops. The precise value of a bypass capacitor is not very important. For this lab, we can use a few microfarad capacitor or higher at the VCC and ground of the L293B chip.

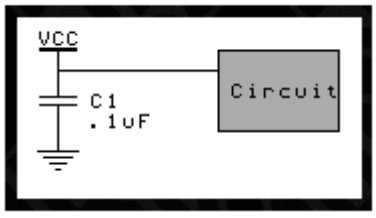


Figure 5: Bypass Capacitor

Experiment

Part 1: Control a LED using PWM

Goal: Generate a PWM signal with a period of 500 milliseconds and a duty cycle of 50%. Wire the output of the PWM signal to an LED.

Part 2: Controlling the speed of a single direction Motor control

Goal: Use HyperTerminal to allow the user to control the speed of the motor and turn the motor 'ON' and 'OFF'. When the motor is 'ON', the user should be able to increment the speed by pressing an 'I'. When the motor is 'ON', the user should be able to decrement the speed by pressing 'D'. The user should also be able to turn the motor 'ON' by pressing an 'N' and turn the motor 'OFF' by pressing an 'F'.

Setup the appropriate circuit as described above in *Figure 3* and be sure to use a separate power source to drive the motor, make sure you have a common ground between the supplies, and utilize a bypass capacitor to keep power surges from resetting the power supply. For this portion of the experiment, you may use a period of 100 microseconds.

VERY IMPORTANT NOTICE: DO NOT grab the motors shaft while it is running.

Questions

1. Why is it so important that we used diodes in all of our circuits?
2. What is the purpose of the capacitors connected to the L293B chip?
3. If we wanted to use motors that draw 2 amps each, what would we have to do to accomplish this task?
4. If operating in bi-directional mode, why would you have to stop the motor and wait before switching directions?
5. What is the maximum voltage that can be supplied to the L293B driver?
6. Provide some example applications for a servo motor.

Appendix F.7: Servo Motor Control using PWM

Objective

The objective of this lab is to introduce the servo motor and its operation. The students will learn how to use Pulse Width Modulation to control 2 servo motors using the switches as inputs.

Background

Servo Motor

A servo motor is a small device that has an output shaft. The servo's shaft can be positioned by sending the servo a coded signal. As long as the coded signal exists on the input line, the servo will maintain the angular position of the shaft. As the inputted signal changes, the angular position of the shaft changes. One of the main uses of servo motors is in robotics.

A servo motor consists of some control circuits and a potentiometer that is connected to the output shaft. The potentiometer allows the control circuitry to monitor the current angle of the motor. If the shaft is in the correct position, the motor shuts off. If the circuit finds that the angle is not correct, it turns the motor to the correct direction until the angle is correct. The output shaft is usually capable of traveling around 180 degrees. The amount of power applied to the motor is proportional to the distance it needs to travel. For example, if the motor needs to turn a large distance, the motor will run at full speed, and if it needs to turn only a small amount, the motor will run at a slower speed.

A servo motor consists of three wires – ground, supply, and a control wire. To communicate the angle at which the servo should run, you use the control wire. The angle of the motor is determined by the duration of the pulse that is applied to the control wire. This is called Pulse Coded Modulation. The servo expects to see a pulse every 20 milliseconds. The length of the pulse will determine how far the motor turns. For example, a 1.5 millisecond pulse will make the motor turn 0 degrees. If the motor is shorter than 1.5 ms, then the motor will turn the shaft closer to 0 degrees or counterclockwise. If the pulse is longer than 1.5 ms, the shaft turns closer to 180 degrees or clockwise. Figure 1 provides an illustration of the servo motor angular positions based on the pulse duration.

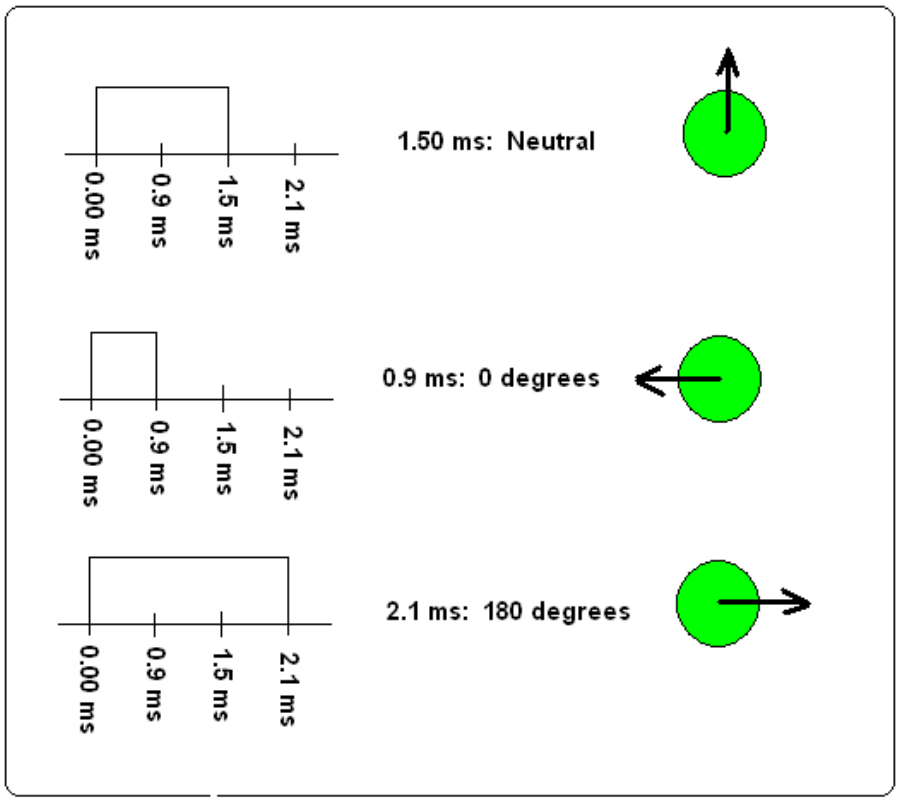


Figure 1: Pulse duration and Output Shaft Positions

Figure 1 is representative of the Hitec servos (HS-475) that are used in the lab experiment. Hitec servo motors require 3-5 V peak-to-peak square wave pulse. Their pulse duration is from 0.9 ms to 2.1 ms with 1.5 ms as center. The pulse refreshes at 50 Hz meaning that the servo motor expects to see a pulse every 20 milliseconds. The initial polarity of the pulse is 'High. All Hitec Servos can be operated within a 4.8-6V range. On all Hitec servos the Black wire is 'ground', the Red wire (center) is 'power', and the third wire is 'signal'). All Hitec servos turn in the clockwise direction. For more information on the Hitec servo motor, take a look at the Hitec servo datasheet (HS-475).

Experiment

Use the three switches on SW1 on the MCU Project Board to control the bottom Hitec servo motor. If SW1-1 is 'ON', the motor should be at 0 degrees. If SW1-2 is 'ON', the motor should be at 90 degrees (neutral). If SW1-3 is 'ON', the motor should be at 180 degrees. If any

combination of the switches is selected, the servo motor should not move from its previous position until only a single switch is turned 'ON'.

The top motor should be controlled in the same manner as described above except it should use the switches SW2 on the MCU Project Board.

Questions

1. What is a phototransistor? List some practical applications of a phototransistor.
   In order read a phototransistor, what processor expert bean would you have to use?
2. Design a circuit to operate a phototransistor in "Active" mode of operation. Be sure to indicate the appropriate resistor value and show how you found this value.
3. What is a lamp dimmer? What processor expert bean(s) could you use to create a lamp dimmer?
4. When implementing a bidirectional DC motor circuit, it is recommended to have a delay when switching the direction of the motor – forwards to backwards and vice versa. Why is this delay necessary?
5. Work with your lab partner to decide which final project you are going to do. Indicate either 'project1' or 'project2' as the answer to this question. Projects are posted on the class website under the 'Projects' tab.

References

1. What is a Servo? Seattle Robotics Society. http://www.seattlerobotics.org/guide/servos.html.

Appendix F.8: XBee Wireless Module Interfacing

Objective

The purpose of this lab is to provide students with an understanding of how to create custom applications with the XBee modules. The students must successfully demonstrate the communications between two XBee modules using the provided template project, Wireless UART Demonstration. Students should accept analog input values from the MCU project board potentiometer, convert these values using the Analog-to-Digital Converter on one of the XBee modules, transmit the converted values to another XBee module, and upon a keyboard press, display the converted values using HyperTerminal in the format "xxx" where "xxx" can range from 000-255.

Background

Potentiometer

A potentiometer is a variable resistor that can be used to acquire a manually adjustable output voltage. The potentiometer on the MCU Project Board -2 will be used to provide an analog input signal to the XBee ADC. *Figure 1* provides the circuit schematic of a potentiometer.

Figure 1: Potentiometer

Analog to Digital Converter (ADC)

An analog-to-digital converter changes a continuous signal to discrete digital numbers. The digital output can then be represented in any number base representation such as binary, hexadecimal, decimal, etc. The **resolution** of an ADC is the number of bits in the output conversion. The MCU's ADC can be configured using processor expert to perform either 8-bit or 10-bit conversions. The possible values that can be used to represent the input voltage for an ADC are $2^N - 1$, so for an 8-bit ADC, the possible values are 2^8-1 = 255. For example, if you have an analog input voltage with a range from 0-5 V and an 8-bit ADC, you will have an ADC voltage resolution or step size = $(5-0)/2^8$ = 0.0195 V. In conclusion, your analog signal can be

represented by 256 discrete values with 0.0195 V steps. It is important to note that if you increase the resolution, then you increase the accuracy of the ADC.

XBee Modules

The XBee modules were developed to provide a low-cost, low-power solution to support wireless sensor networks. These modules provide a line-of-sight range up to 300 feet, a RF data rate of 250 kilobits per second, operate from 2.8 – 3.4 V, and support the following network topologies: point-to-point, point-to-multipoint, and peer-to-peer. *Table 1* provides a detailed specification of the XBee module [8].

Table 1 XBee OEM RF Module Specifications [8]

Performance	
Indoor/Urban Range	up to 100 feet
Outdoor RF line-of-sight range	up to 300 feet
Transmit Power Output (software selectable)	1 mW (0 dBm)
RF Data Rate	250,000 bps
Serial Interface Data Rate (software selectable)	1200-115200 bps
Receiver Sensitivity	-92 dBm (1% packet error rate)
Power Requirements	
Supply Voltage	2.8-3.4V
Transmit Current (typical)	45 mA (@3.3 V)
Idle/Receive Current (typical)	50 mA (@3.3 V)
Power-down Current	< 10 μA
General	
Operating Frequency	ISM 2.4 GHz
Dimensions	0.960" x 1.087"
Operating Temperature	-40 to 85 °C
Networking and Security	
Supported Network Topologies	Point-to-Point, Point-to-Multipoint, Peer-to-Peer
Number of Channels (software selectable)	16 Direct Sequence Channels
Addressing Options	PAN ID, Channel and Addresses

The main components of the XBee modules include a MC9S08GT60 microcontroller and MC13193 RF chip. The MC9S08T60 is a member of the HCS08 Freescale microcontroller family. The HCS08 family utilizes a 40 MHz HCS08 central processor unit, features two serial

communications interface (SCI) modules and a serial peripheral interface module, and includes 8-channel, 10-bit analog-to-digital converters (ADCs) [3]. The MCS08GB60 datasheet provides a full list of features and can be found on the Freescale website at www.freescale.com. The MC13193 chip is a short range, low-power transceiver developed by Freescale Semiconductor which when used with an appropriate microcontroller, provides an efficient solution to short-range networks [4]. The datasheet for this chip can also be found on the Freescale website [3]. The XBee modules combine these two components and provide a 20-pin output. *Table 2* provides a list of the XBee pin assignments taken from the XBee datasheet, and *Figure 2* provides an illustration of the XBee breakout board.

Table 2 XBee Module Pin Assignments [4]

Pin	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / **CONFIG'**	Input	UART Data In
4	DO8*	Output	Digital Output 8
5	**RESET'**	Input	Module Reset (reset pulse must be at least 200 ns)
6	PWM0/RSSI	Output	PWM Output 0 / RX Signal Strength Indicator
7	PWM1	Output	PWM Output 1
8	[reserved]	-	Do not connect
9	**DTR'** / SLEEP_RQ / DI8	Input	Pin Sleep Control Line or Digital Input 8
10	GND	-	Ground
11	AD4 / DIO4	Either	Analog Input 4 or Digital I/O 4
12	**CTS'** / DIO7	Either	Clear-to-Send Flow Control or Digital I/O 7
13	ON / **SLEEP'**	Output	Module Status Indicator
14	VREF	Input	Voltage Reference for A/D Inputs
15	Associate / AD5 / DIO5	Either	Voltage Reference for A/D Inputs
16	**RTS'** / AD6 / DIO6	Either	RTS Flow Control, Analog Input 6 or Digital I/O 6
17	AD3 / DIO3	Either	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Either	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Either	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0	Either	Analog Input 0 or Digital I/O 0

The modules can operate with minimum connections of VCC, GND, DOUT, and DIN, and firmware upgrades can be made with minimum connections of VCC, GND, DIN, DOUT, RTS, and DTR.

Figure 2 XBee Breakout Board

X-CTU Software

The X-CTU Software was developed by MaxStream to provide an interface for configuring and testing the XBee modules. The software is easy to use and install and works well with Freescale CodeWarrior. After installing the software on your machine, the X-CTU software will execute when you compile a CodeWarrior project. *Figure 3* provides a screenshot of the X-CTU software with the serial communications properties setup.

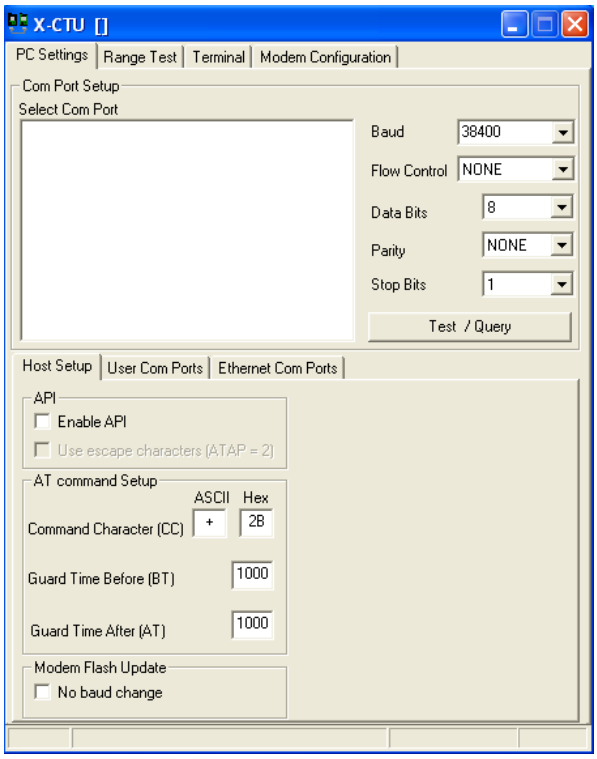


Figure 3 X-CTU Software Screenshot

Experiment

Goal: Utilize one of the XBee Analog to Digital Converters to read an analog inputted value from the MCU Project Board potentiometer on a XBee module (1), transmit the values to another XBee module (2), and upon a keyboard press, display the values using HyperTerminal onto the computer screen.

1. For this experiment, we will want create a project using Freescale CodeWarrior for the HCS08 since the XBee houses an 8-bit microcontroller
2. Use the Analog-to-Digital Converter Module section in [5] to figure out how to properly initialize the control registers for an 8-bit resolution, continuous conversion
3. Open up the provided CodeWarrior project "Wireless2.mcp"
4. Insert the register initialization code into the "Wireless2.c" file
    a. Be sure to include this code after the MCUInit() and RadioInit() functions in order to prevent the overwriting of your initialization of the ATD registers
5. Upon a keyboard press – read, transmit, and display the converted values
6. Properly wire and connect the XBee modules to the MCU Project Board -2. The RS-232 port on the bottom right of the MCU Project Board -2 will be connected to the computer, and pins TXD and RXD will be connected to the XBee module. Refer at *Table 2* and *Figure 2* for wiring details.
7. Select 'Project' >> 'Make' (X-CTU software should automatically prompt if no errors)
8. On the "PC Settings" tab, select the appropriate COM port. The COM port settings are 38400 baud, flow control = NONE, 8 data bits, parity bits = NONE, and 1 stop bit.
9. On the "Modem Configuration" tab, select the 'XB24' modem, update firmware checkbox, and the 'Wireless UART Demonstration' function set. Then click the 'Write' button.
10. When the Info box appears the XBee module should be reset. The module can be reset either using the button on the programming board or using the XBee pin 5 (RESET). The X-CTU software should resume and program the XBee module.
11. Repeat steps 6-10 for the second XBee module.
12. Open 'Programs' >> 'Accessories' >> 'Communications' >> 'HyperTerminal' to test the programs
13. Be sure that the HyperTerminal settings match the settings provided in *Step 8*

Appendix F.9: Bidirectional DC Motor Control Project

Objective

Create a user interface using the keypad and LCD to control a bidirectional motor and lamp dimmer. Display the speed and direction of the motor the LCD and provide the user with options to change the direction and speed. The user should be able to control the motor, the lamp, or both simultaneously.

Additional Information

- Lamp Dimmer Portion
  - Refer to pages 379-392 in your textbook
  - Be sure to design the appropriate circuit to drive the LED (Assume: VLED = 1.8V)
- Bidirectional Motor Control
  - Refer to pages 393-398 in your textbook
  - Utilize the L293B datasheet
  - When switching between forward and reverse mode make sure to stop the motor and wait.
  - Utilize the information from your past lab experiments and the references located on the class website at http://www.csee.wvu.edu/classes/cpe313/labs.html under *Links.*

Appendix F.10: Servo Motor Control using a Phototransistor Project

Goal

Utilize two servo motors, parabolic mirror, and phototransistor to locate the position of a flashlight. The servo motors should be controllable by the keypad. The LCD should be used to display the status of the search and display three states "Waiting", "Searching", and "Found".

Additional Information

- Refer to the OP599 transistor datasheet (located on the class website) for information on the phototransistor
- Be sure to design the interface to the phototransistor correctly – refer to http://www.er-online.co.uk/minisumo/EdgeDetection.php for more information
- Utilize the information from your past lab experiments and the references located on the class website at http://www.csee.wvu.edu/classes/cpe313/labs.html under *Links.*