The University of British Columbia
Department of Electrical and Computer Engineering

EECE 474
Bomberbots Project
Final Report

Submitted To
Dr. G. Dunford
By
Group 5

Alex Lau
Peter Chan
Stephen Chu
Sunny Chan
Terrence Tam

March 24, 2002

ABSTRACT

A mechanical robot game has been designed and implemented specially for a disabled boy with the objective for him to interact with friends. The idea was inspired from the classic video game, “BomberMan”, originally created by Hudson Soft in 1989. The objective of the game is to “blow up” the opponent with electronic bombs. The game setting is a square grid with two players starting at opposite corners. The robots will be walking around in the battlefield randomly. The players can tell the robot to ‘lay’ bombs by pressing the button on the user panel. Only one bomb can be laid on the grid at a time and the explosion occurs when the robot steps on the activated bomb. The movement of the robots and the position of the bombs are managed by the HC11 microprocessor inside the battlefield control tower. Commands are sent from the tower through RF connections to the robots. Each robot has a BASIC Stamp 2 microcontroller as its brain. Line detection sensors are used to prevent the robots from hitting the walls. The game ends when one of the robots ran out of life units.

TABLE OF CONTENTS

ABSTRACT …… ii
LIST OF FIGURES …… iv
LIST OF TABLES …… v
1.0 INTRODUCTION …… 1
2.0 DESIGN SPECIFICATIONS …… 3
2.1 Game Objective …… 3
2.2 Robot Design …… 4
2.3 Battle Field Design …… 5
3.0 ROBOT CONSTRUCTION …… 7
3.1 BASIC Stamp 2 Micro-controller …… 7
3.2 Drive System …… 10
3.3 Line Sensor Circuitry …… 12
3.3.1 IR Emitter/Detector Array …… 12
3.3.2 Threshold Comparator …… 15
3.4 RF Communication …… 17
3.5 Power Solution and Batteries …… 19
4.0 BATTLE ARENA …… 22
4.1 HC-11 Microprocessor …… 22
4.2 Bomb Circuitry …… 22
4.3 RF Transmission …… 26
5.0 SOFTWARE …… 30
5.1 Robot Algorithm …… 30
5.2 Battle Arena Algorithm …… 32
5.2.1 Receiving Data from User (Pushbutton control) …… 32
5.2.2 Controlling the LEDs …… 33
5.2.3 Sending data to the RF module …… 35
5.2.4 Software Algorithm Design …… 36
6.0 BUDGET …… 38
7.0 SUMMARY …… 39
8.0 RECOMMANDATIONS …… 40
APPENDIX A: PBASIC SOURCE CODE …… 41
APPENDIX B: FLOW CHART FOR PBASIC SOFTWARE …… 45
APPENDIX C: C SOURCE CODE …… 46
APPENDIX D: FLOW CHART FOR C SOFTWARE …… 70
APPENDIX E: ROBOT CIRCUIT DIAGRAM …… 71
APPENDIX F: ROBOT PICTURES …… 72

LIST OF FIGURES

Figure 1. Battlefield Board ………………………………………………………… 6

Figure 2. BASIC Stamp 2 (24-pin DIP package) Schematic …………………. 8

Figure 3. BASIC Stamp 2 and its Components ………………………………. 8

Figure 4. BASIC Stamp 2 Programming Connections ………………………. 9

Figure 5. Servo Circuitry ……………………………………………………. 10

Figure 6. Futaba S-148 Servo – Gear Diagram ……………………………….. 11

Figure 7. Dimensions and Schematic Diagram for QBR1114 ……………….. 13

Figure 8. A Simplified Circuit Diagram Consisting One QRB1114 Sensor …. 14

Figure 9. Threshold Comparator Circuit Diagram ……………………………. 16

Figure 10. RF Receiver RXM -418-RM Circuit ………………………………… 18

Figure 11. RF data stream ……………………………………………………….. 19

Figure 12. Player Controller Panel ……………………………………………… 24

Figure 13. Bombing System Circuit …………………………………………….. 25

Figure 14. Block Diagram of Radio Frequency Communication System …….. 27

Figure 15. Transmitter Circuit ………………………………………………….. 28

Figure 16. Block Diagram of the Transmission Protocol ……………………..… 29

LIST OF TABLES

Table 1. BASIC Stamp 2 Pin Description ……………………………………….. 9

Table 2. Summary of Voltage Requirement for Power Source 1………………. 20

Table 3. Summary of Voltage Requirement for Power Source 2……………….. 20

Table 4. Bit inputs for the decoder at each location on the game board ………. 34

1.0 INTRODUCTION

The Bomberbots game is an innovative and original project designed specially for a disabled boy with the objective for him to interact with friends. The report details the design, construction, and testing of the Bomberbots game. In order for the project to be successful, the group spent an intensive amount of time in research and development and decided that the game play was the most important aspect because it dictates whether the game can be implemented mechanically given our budget goals and time constraints. This was our main concern throughout the project and it is the reason why we chose this modified BomberMan-type game.

An interactive game between two players can be very fun and exhilarating especially when it is physically implemented with moving parts instead of watching it on a monitor. We wanted to build a project that served a purpose in aiding a child to interact with other children through playing games. The game interface should be simple for both players to control yet provide maximum enjoyment through its interaction. The player has the ability to activate the bombs but they will only be activated for a certain amount of time before it is deactivated. The strategy for the player will be to predict the future movement of their robot to avoid any activated bombs and yet set the ones that may "blow up" the opponent.

Since the player controller panels must be attached to the micro-controller unit, the group decided to use RF to control the robots wirelessly, as this will enhance the visual

presentation of the game. The micro-controller unit was mounted underneath battle board and acted as the central processing unit of the game. The robots, player controller panels, and the bomb lights were all connected to this unit. Finally, a great deal of software engineering was needed to properly design the structure of the game flow.

This report describes the design of the robots, battle arena, and software. Each section discusses its components and method of operation. The report will conclude with an assessment of the budget and any future recommendations.

2.0 DESIGN SPECIFICATIONS

The following sections outline the design specification for the game objective, the robot design, and the battlefield design.

2.1 Game Objective

The objective of Bomberbots is to implement a physical game between two players that are trying to "blow up" the other robot by activating bombs on the battle arena. Each player can activate a bomb by pressing the button on the user panel. The HC11 software algorithm generates the movement of the robot randomly so there is no need for the player to control the movement of the robot.

The robots start at opposite corners of the battle arena. Each robot faces a preset direction and each robot has 3 lives. Once the game has begun, the microcontroller will direct each robot in a certain path at the same speed. As the robots reach each intersection, the players can activate the bombs by pressing the button on the controller panel. These live bombs stay on for approximately 5 seconds and then it deactivates which will represent an explosion. Only one bomb can be activated on the battle arena at a time.

If a robot enters an intersection with an activated bomb, it will detonate and the robot will stop for one turn and its life decreases by one. A tone will be played while the robot is stunned. This bomb will reset and can be activated again when a player sets it. A player

can detonate a bomb that has been activated by the same player. The game continues until either player has no lives and when this happens, both robots are stopped forever.

2.2 Robot Design

The role of the two robots is to give the players a physical entity to look at so that the players can see the position of the robots and plan their strategies during the game. The tasks of the robots are to move around in the battlefield without hitting the walls and show some indication when they are stunned by a bomb.

The BASIC Stamp 2 micro-controller is chosen to be the brain of the robots because it is easy to use and has a generous amount of I/O pins to control various components.

The movement mechanism of the robots is implemented using the differential drive system with two modified servos and a third supporting wheel. This minimizes the project cost and still enables the robots to move around without difficulty.

In order for the robots to avoid hitting the walls of the battlefield, a number of methods such as using IR object detection sensors, color sensors, and bumper sensors were being considered. Finally, line detection was chosen because of its reliability and exceptional performance.

Since the robots will be moving around randomly in the battlefield, there is no way to predict where the future positions of the robots might be. Therefore, in order to avoid collisions of the robots, the future positions of the robots are predetermined by the HC11 microprocessor underneath the battle arena. The HC11 will make sure the robots will not collide with each other. After the "next move" is determined, commands will be sent to the robots through RF connections.

If a robot steps on a bomb, it will indicate that it has been stunned by playing a tone through a electronic buzzer implemented on the robot.

Each robot is being powered by two 9V batteries. One of the batteries is to power the BASIC Stamp 2 micro-controller and the IR sensors in the line following module. The other battery is mainly to provide power to the servos. This way the robot would have enough power to move around and would not affect the BASIC Stamp 2 performance.

2.3 Battle Field Design

The game environment will be on a 95cm x 95cm square board divided into 16 squares, which are separated by nine 5cm x 5cm pillars and a wall around the perimeter of the board. The height of the battlefield is 8cm. See Figure 1 for the dimensions of the battlefield board. The floor base and the walls are made from 3/8-inch plywood and the pillars are made from blocks of pinewood. Due to the specifications of the phototransistors on the robots, we have decided it would be optimal to paint the inside of

the battle arena with black paint and use white electrical tape on the surface to serve as tracks for the robots to follow.

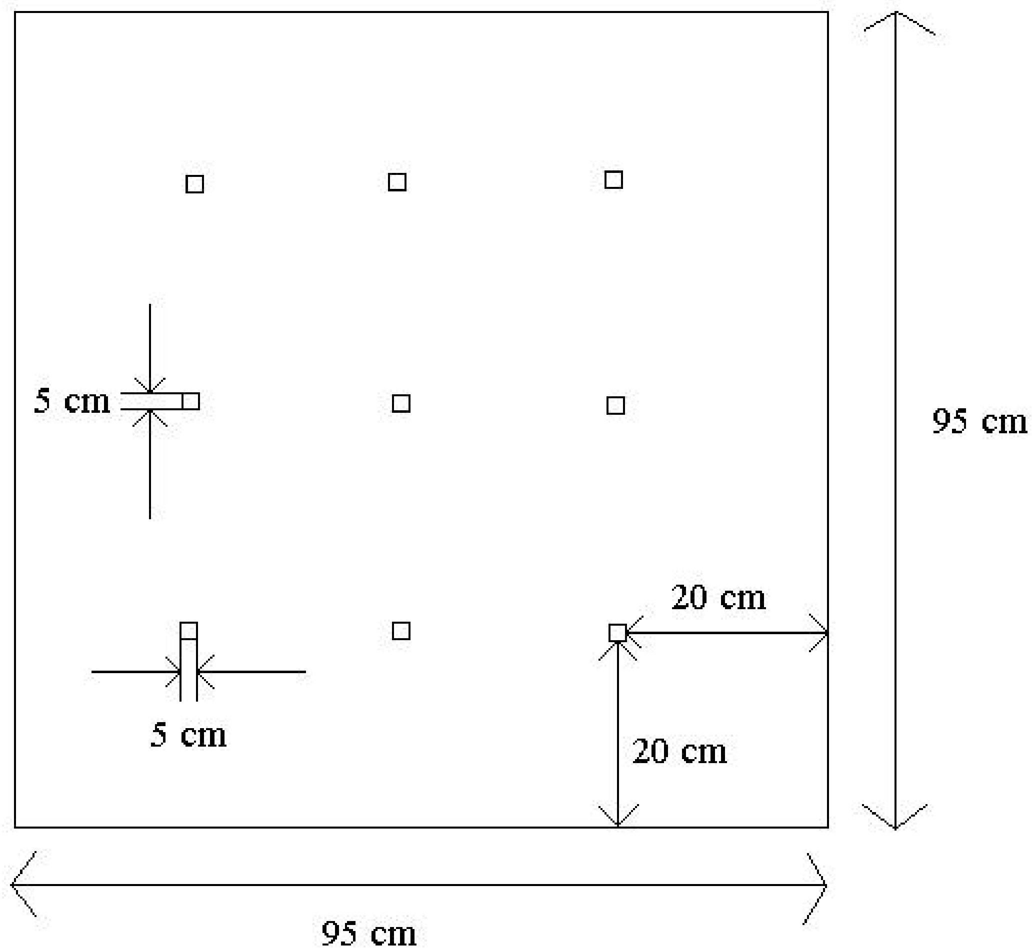


Figure 1. Battlefield Board

3.0 ROBOT CONSTRUCTION

Each robot has structure consists of three layers: the main circuitry layer on the top, the servo layer in the middle, and the line sensor circuitry layer at the bottom. The main circuitry layer contains a microcontroller BASIC Stamp 2, a DB-9 serial port connector, a Radio Frequency (RF) receiver, a power switch, power inputs from batteries, and power outputs for the other two layers. The servo layer contains two modified servos each attached with a wheel. The line sensor circuitry layer contains an IR emitter/detector array and a threshold comparator circuit. In addition to the three layers, a table tennis ball is used as the back wheel and two 9V batteries are attached at the back of each robot.

3.1 BASIC Stamp 2 Microcontroller

The BASIC Stamp 2 microcontroller was chosen to be the brain of the Bomberbots. It is a very popular robotics microcontroller module built by Parallax Inc. BASIC Stamp 2 is chosen over other programmable microprocessors and microcontrollers because of its simple, but powerful, language structure. It is very fast in downloading and it is easy for programmer to debug codes. BASIC Stamp 2 is capable of running approximately four thousand instructions per second and is programmed with a simplified and customized form of the BASIC programming language called PBASIC (Parallax Basic). It has an EEPROM of 2K Bytes in size, capable of holding approximately 500 to 600 instructions. The EEPROM size is sufficient for our robots because PBASIC is very compact and our source code for the robots is not very complicated.

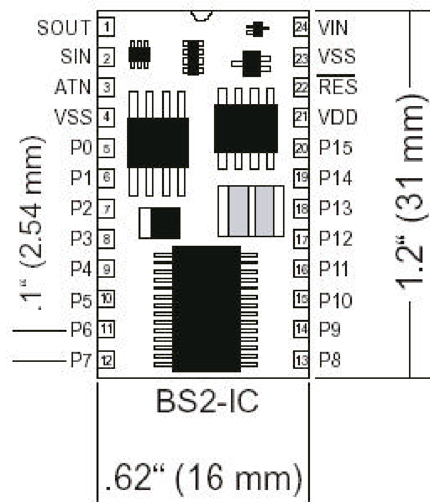


Figure 2. BASIC Stamp 2 (24-pin DIP package) Schematic

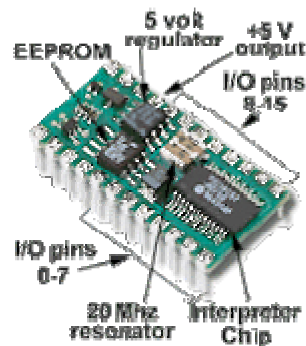


Figure 3. BASIC Stamp 2 and its Components

BASIC Stamp 2 has a BASIC Interpreter chip, a PIC16C57 microchip, internal memory (RAM and EEPROM), a 5-volt regulator, and 16 general-purpose I/O pins. The 16 programmable I/O pins are suite for digital input and output with TTL/CMOS level (0 to 5 volt) signals. It can be used to directly interface to TTL-level devices such as buttons,

LEDs, speakers, potentiometers and shift registers. This property helps to simplify our design in a great deal.

Pin	Name	Description
1	SOUT	Serial Out: connects to PC serial port RX pin (DB9 pin 2 / DB25 pin 3) for programming.
2	SIN	Serial In: connects to PC serial port TX pin (DB9 pin 3 / DB25 pin 2) for programming.
3	ATN	Attention: connects to PC serial port DTR pin (DB9 pin 4 / DB25 pin 20) for programming.
4	VSS	System ground: (same as pin 23) connects to PC serial port GND pin (DB9 pin 5 / DB25 pin 7) for programming.
5-20	P0-P15	General-purpose I/O pins: each can sink 25 mA and source 20 mA. However, the total of all pins should not exceed 50 mA (sink) and 40 mA (source) if using the internal 5-volt regulator. The total per 8-pin groups (P0 – P7 or P8 – 15) should not exceed 50 mA (sink) and 40 mA (source) if using an external 5-volt regulator.
21	VDD	5-volt DC input/output: if an unregulated voltage is applied to the VIN pin, then this pin will output 5 volts. If no voltage is applied to the VIN pin, then a regulated voltage between 4.5V and 5.5V should be applied to this pin.
22	RES	Reset input/output: goes low when power supply is less than approximately 4.2 volts, causing the BASIC Stamp to reset. Can be driven low to force a reset. This pin is internally pulled high and may be left disconnected if not needed. Do not drive high.
23	VSS	System ground: (same as pin 4) connects to power supply's ground (GND) terminal.
24	VIN	Unregulated power in: accepts 5.5 - 15 VDC (6-40 VDC on BS2-IC rev. e), which is then internally regulated to 5 volts. May be left unconnected if 5 volts is applied to the VDD (+5V) pin.

Table 1. BASIC Stamp 2 Pin Description

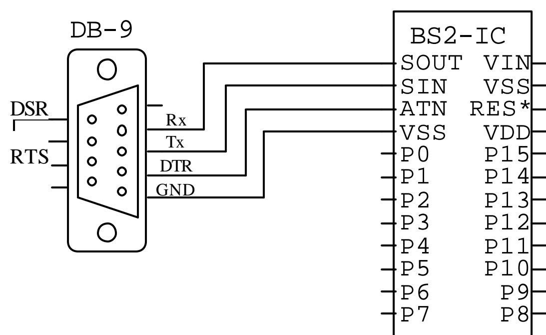


Figure 4. BASIC Stamp 2 Programming Connections

3.2 Drive System

When deciding the suitable drive system components for our Bomberbots, we choose to use differential drive with two servos instead of using DC or stepper motors. Although DC and stepper motors are cheaper in cost, they are not as easy to operate and control as servos, especially in the case of using a digital micro-controller like BASIC Stamp 2. DC and stepper motors require a very complicated control circuitry with encoders. Servos do not require extra circuitry when they are used to be controlled by digital micro-controllers. They simply respond to a pulse width modulation signal (PWM) sent by a digital micro-controller.

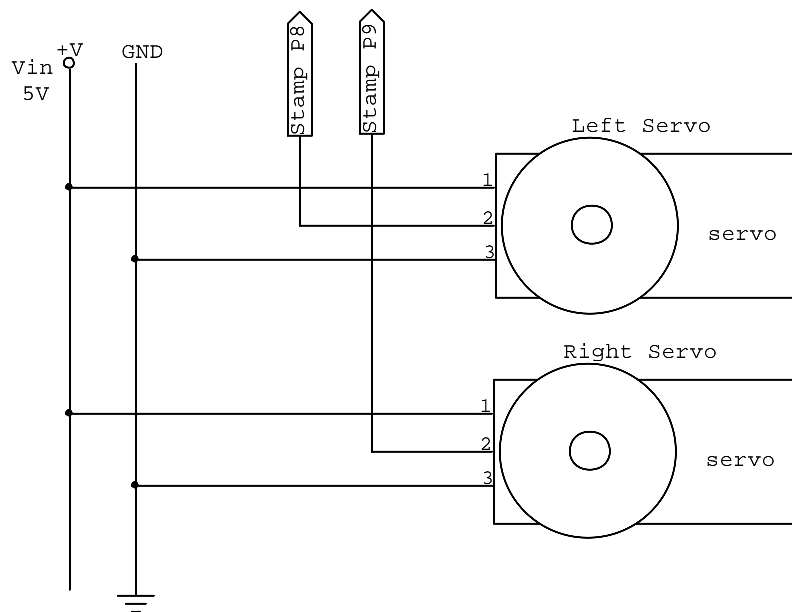


Figure 5. Servo Circuitry

Most servos are designed to move about 90° to 180° total. In order to use the servos as the driving system of our Bomberbots, we need to modify them so that they can rotate continuously. Each of our Bomberbots uses two modified servos. We choose to use the Futaba-S148 servos because they provide low cost, easy-to-modify gear motors. All we need to do is to remove the stop tab and the drive plate from the bottom of the final gear. Each servo is attached with a wheel. The two servos act as the front wheel while a table tennis ball attached at the back of each Bomberbots act as a tail wheel.

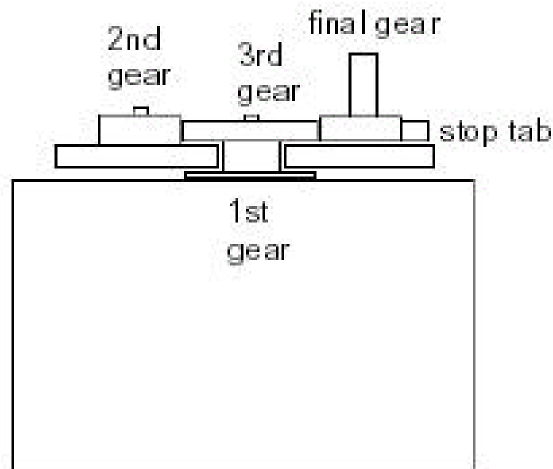


Figure 6. Futaba S-148 Servo – Gear Diagram

In order to let the Bomberbots turn corners in shorter spaces, make sharper cornering turns, and cause less friction on the tail wheel, we placed the servos near the center of the chassis. Since the batteries are at the back of the chassis, a high weight distribution is distributed at the back. The table tennis ball wheel provides a very smooth surface for minimizing friction as much as possible at the tail.

3.3 Line Sensor Circuitry

One of the features of our Bomberbots is line following. They are able to follow lines and determine whether they are at intersection or not. Our line following module is composed of two distinct sections: an IR emitter/detector array and a threshold comparator circuit.

3.3.1 IR Emitter/Detector Array

Each of our robots has an IR emitter/detector array that consists of five reflective object sensors. This type of sensors is designed to discriminate a white versus a black surface. The sensors we are using are optical interrupter switches (part number QRB1114), which consist of an infrared emitting diode and an NPN silicon phototransistor. The phototransistor responds to radiation from the emitting photodiode only when a reflective object passes within its field of view. The sensing distance for QRB1114 is approximately 3.81mm. This sensor is good for finding white lines on black floor or black lines on white floor. In our project, we programmed our Bomberbots to identify white lines on black floor.

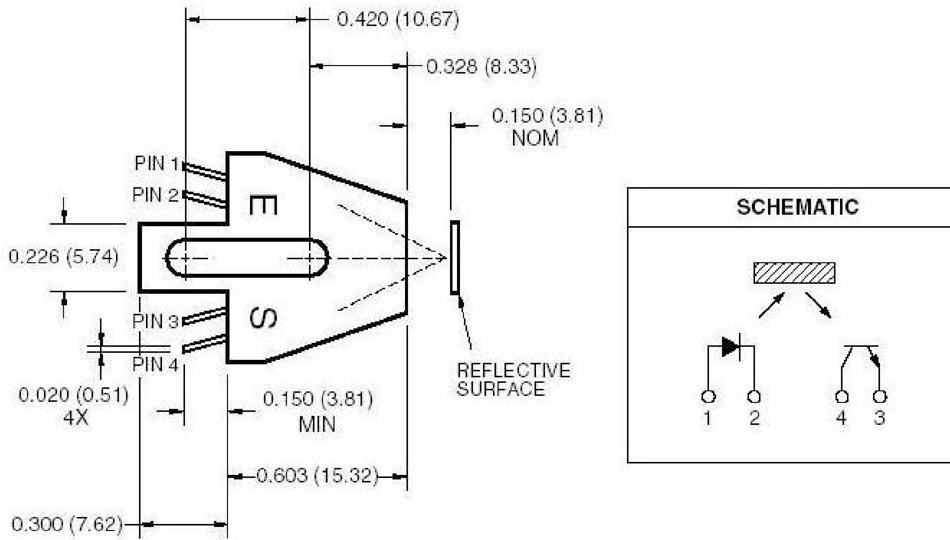


Figure 7. Dimensions and Schematic Diagram for QBR1114.

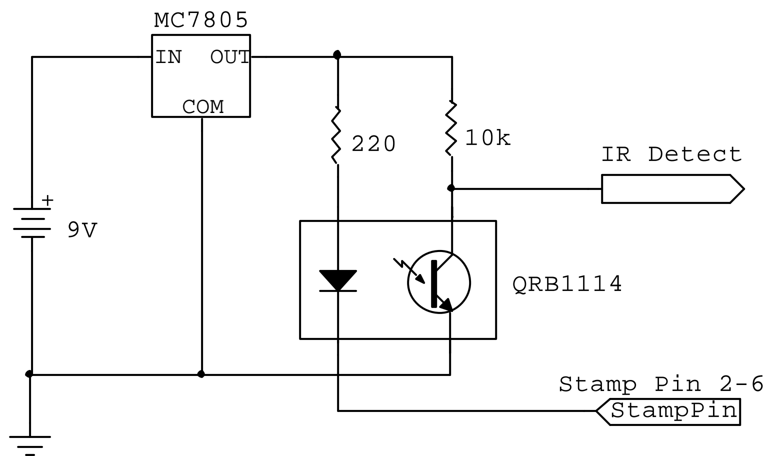


Figure 8. A Simplified Circuit Diagram Consisting One QRB1114 Sensor

The above circuit diagram shows how to drive and read the sensor. The light emitting photodiode uses a +5 volt power source in series with a 220 O resistor. It can be activated by making the associated BASIC Stamp output pin low. When it is active, an IR light beam will be emitted and the reflected bean will strike of the phototransistor end, affecting the current flow through it. More reflected IR causes greater current to flow through the transistor.

The light sensing phototransistor uses a +5 volt power source in series with a 10 kO resistor and the IR Detect output is taken at the collector pin of the phototransistor. This circuit provides a voltage output proportional to the amount of light received by the sensing phototransistor. It forms a voltage divider with the output dependant on current flow through the transistor. As the current flow through the transistor increases, the voltage across the 10 kO resistor also increases, causing the IR Detect output decreases. Therefore, the greater the reflected IR, the lower the output voltage. When there is little or no reflection, the current flow through the transistor is reduced making it look like a very large resistance in the circuit, causing the output voltage to increases. Therefore, the smaller the reflected IR, the higher the output voltage.

Decreasing the resistance (smaller than 220 O) on the photodiode can increase the intensity of the IR beam, which results in a slightly larger range of detection. However, this increase in intensity causes the sensor to be sensitive enough to detect the little reflection from black surface, causing the sensor unable to distinguish black and white surfaces. The resistance on the phototransistor is somewhat arbitrary, as long as it is high

enough to give a good voltage swing and low enough that ambient light does not tend to trip the sensor too much. Any resistor from 10 kO to 20 kO will work well.

For each of our Bomberbots, the five IR sensors are under direct control of the BASIC Stamp 2. It is programmed that only one sensor is turned on at a time. The output of the 5 IR sensors is connected to a threshold comparator, which ensures the detection of a reflective and non-reflective surface would have an output of either 0V or 5V (logic low or high). Allowing multiple IR sensors to light will cause inaccurate and unpredictable results. Refer to the **Section 5.2** of this report for more detail on how the BASIC Stamp 2 controls the five IR sensors.

3.3.2 Threshold Comparator

The second portion of the line following module is the threshold comparator. The purpose of this circuit is to compare the output from the output from the IR detector with the level setting on the threshold potentiometer. The comparator is acting as an A/D (analog/Digital) converter.

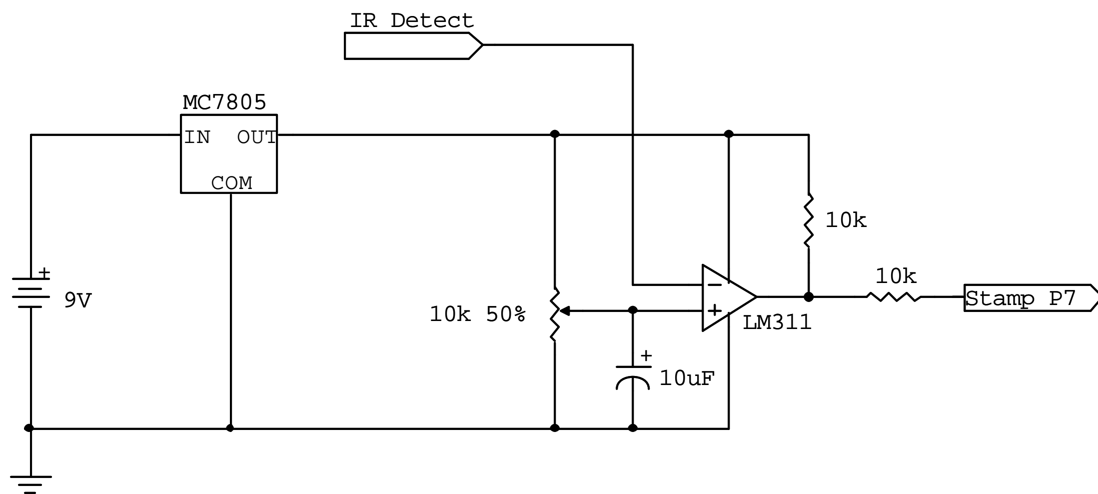


Figure 9. Threshold Comparator Circuit Diagram

The above circuit diagram shows the configuration of the threshold comparator. The IR Detect signal is connected to the minus terminal of the single voltage comparator LM311 while the threshold pot is connected to the plus terminal. LM311 works in a very simple manner. Its output would be either high or low, depending on which of the two input signal has the higher voltage. If the minus input is higher than the plus input, the comparator will output low (0V). If the input is higher than the minus input, then the comparator will output high (5V). The purpose of the threshold potentiometer is to allow us to adjust the input voltage into the plus terminal. For our Bomberbots, we set this input voltage into the plus terminal to be around 2.5V.

Choosing the suitable value of the potentiometer is very important to this circuit. If the value is picked to be very small (for instance 1 kO), the comparator may still work well by itself. However when it is connected to the output of the IR sensors, the comparator would output high at a lower voltage, unable to output high at 5V. Any value larger than

5 kO would allow the comparator to be stable when connected to the IR sensors. For our Bomberbots, we chose a 10 kO potentiometer. A 10 kO resistor is connected in series with the comparator output. This output resistor is optional, however it would drain down the current flow into BASIC Stamp 2, preventing it to become malfunction.

When an IR sensor is turned on and detects a highly reflective surface, the output voltage of the sensor goes low. Therefore, if it falls below the threshold pot setting, the comparator will output low. When the sensor detects a non-reflective surface, the output voltage of the senor will go high. If this voltage is higher than the threshold pot setting, the comparator will output high. The IR sensor is very reliable in the sense that it outputs 0.2V when it detects a highly reflective surface and 4.0V when it detects a non-reflective surface. The output of the IR sensors can actually be connected directly to BASIC Stamp I/O pins. The BASIC Stamp 2 is able to recognize 0.2V as a low and 4.0V as a high. Although the comparator is seems to be optional, the existence of the comparator ensures the input to the BASIC Stamps to be digitalized.

In summary, reflective surfaces cause the Line Following module to output a high and non-reflective surfaces case it to output a low.

3.4 RF Communication

The Linx RM Series Transmitter and Receiver 418 MHz modules are chosen for the RF link between the battlefield control tower and the Bomberbots because of their ease of

use. They have a range of over 500 feet and can support data rates up to 10Kbps. In addition, they also have a wide supply range (5.9 – 9 VDC for the transmitter and 3.9 – 9 VDC for the receiver) and low power consumption (~6mA for the transmitter and ~14mA for the receiver) and they also have a wide operating temperature range (-10 °C to 50 °C). These properties make the RF modules suitable for the project.

The Linx RM Series 418 MHz Receiver has two output pins, one for analog and one for digital. The digital pin is connected to one of the BASIC Stamp 2 pin to read the data received as shown in **Figure 10**.

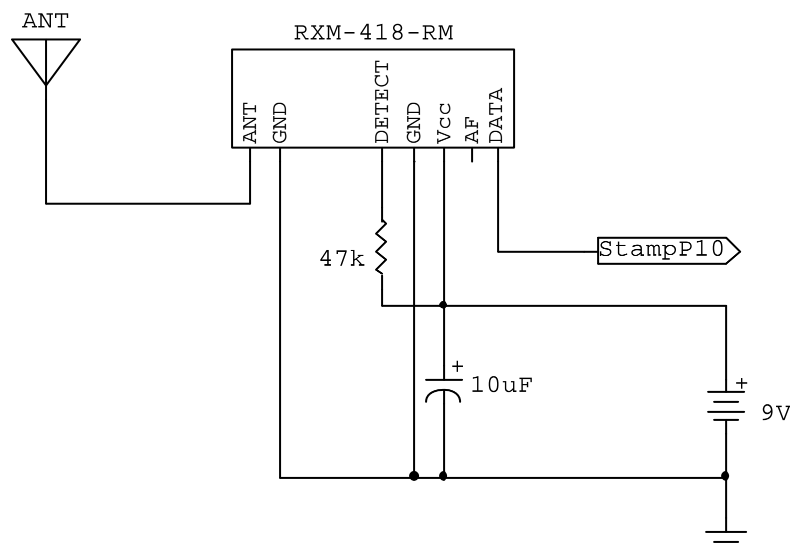


Figure 10. RF Receiver RXM-418-RM Circuit

In case of receiving noise or junk data from other sources, the RF data from the HC11 is sent following the scheme shown in **Figure 11**. At the beginning of each data stream, a

byte of junk data will be sent. This is to ensure the connection between the RF modules is established. Then a synchronization byte is sent following the actual command byte.

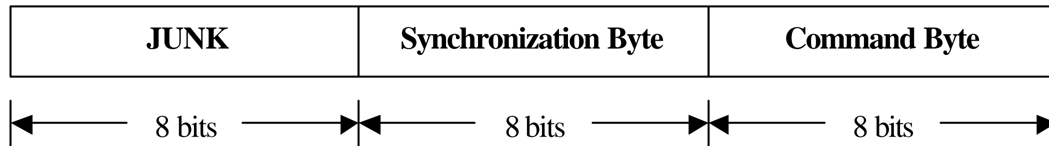


Figure 11. RF Data stream

At the beginning on each turn, the robot will stand by and wait for RF commands. It will check the data received from the receiver and look for the synchronization byte to make sure the data is from the battlefield control tower. Then the next eight bits will be stored into memory. The BASIC Stamp 2 will do this for three times and then compare the values. If they are all the same then it is confirmed to be a valid command and BASIC Stamp 2 will try to execute it. After finish executing the command, the robot will go back to the stand by state and wait for the next command.

3.5 Power Solution and Batteries

Power plays a very important role in the performance of our Bomberbots. Therefore special care in powering the Bomberbots is needed. For each Bomberbots, we have 2 power sources: one for powering BASIC Stamp 2 and the Line Following Module, one for powering the two servos and the RF receiver. Each set of the loads is powered using

a 9V DC battery. The purpose of dividing all loads into 2 separate power system is that we do not want too much load on a single power source, making too much power draw and causing the power supply unstable.

Load	Voltage Required
BASIC Stamp 2 module	6V – 12V (If applying to internal voltage regulator)
Line Following module	5V

Table 2. Summary of Voltage Requirement for Power Source 1

Load	Voltage Required
Futaba S-148 Servo (2)	4.8V – 6V
RF Receiver RMX-418-RM	3.9V – 9V

Table 3. Summary of Voltage Requirement for Power Source 2

For power source 1, we need it to power the mirco-controller and the Line Following module. BASIC Stamp 2 has a built-in 5-volt regulator, which converts an input of 6 to 15 volts down to 5 volts. Since it is recommended to limit the voltage supply to 12 volts on BASIC Stamp 2, we choose to power it using a 9V battery. The Line Following module requires 5V to power it. Therefore we added a 5-volt regulator MC7805ACT to convert the voltage from the 9V battery down to 5V.

For power source 2, we need it to power the 2 servos and the RF receiver. The 2 servos require a lot of current draw and the voltage level plays a very important role on their

performance. The speed of the servos is greatly affected by the voltage level. If the voltage input drops below the require range, the speed becomes very unstable. Since our Bomberbots drive system relies greatly on the speed and the performance of the servos, we need to regulate the voltage applied to the servos. We use a 5-volt regulator MC7805ACT to regulate the voltage input to the servos to ensure speed of the servos would stay constant. In order to allow MC7805ACT and the RF receiver RMX-418-RM to operate, we choose to use a 9V battery. The voltage to the RF receiver is not regulated because a drop in voltage supply would only affect its range of operation. If we regulate the voltage down, we will minimize it operating range.

4.0 BATTLE ARENA

The Battle Arena consists of a HC-11 microprocessor, the bombing system circuitry and the radio frequency communication system circuitry.

4.1 HC-11 Microprocessor

The reason we chose to use HC-11 microprocessor in our project is because it is a popular product used for controlling devices, which makes the learning and developing process easier during the development. The HC11 includes powerful functions such as, Parallel Input/Output, Serial Communication Interface (SCI), Serial Peripheral Interface (SPI), Timing System and Analog-to-Digital Conversion. These functions are quite flexible in that almost anything can be implemented and so it more than fulfills our needs for this project. It also has many I/O pins available for general or specific use. Also, the Image Craft Compiler 11 is a popular development tool used for programming the HC11, which is available at UBC. Finally, the price of the HC11 is lower than other types of microprocessors.

4.2 Bombing System

The Bombing System is a system that would take input data from the two players and activate a bomb according to the player that pressed the button first and the robot that was on top of the bomb. Of course the bomb must be off before it is activated or else the

robot would explode. Whichever player pressed the button first, his or her bomb would be activated for two rounds. After two rounds, the bomb would explode and the whole cycle of activating a bomb starts over again.

The Bombing System consists of two buttons, one for each player, fourteen bombs and two safe zones inside a 4 x 4 grid-like maze. The two safe zones are at (0,0) and (3,3) while the fourteen bombs are at everywhere else. This system consists of an active low four-to-sixteen decoder, fourteen not gates, fourteen 150 Ω resistors, fourteen LEDs which represents the fourteen bombs, two 1000 Ω resistors, two pushbuttons inside two controller panels and a HC11. The Bombing System is powered up by two 9 VDC batteries connected in parallel. A voltage regulator is used to lower the 9 VDC to approximately 5 VDC.

Two controller panels were built for each pushbutton for each player. The controller panel is simply an aluminum box with dimensions of 14cm x 14cm. This hand held control unit has a push button mounted on the topside and connected to the micro-controller unit via two wires. One wire is for the input to the button and the other wire is the output from the button. See figure 12 for the player controller panel.

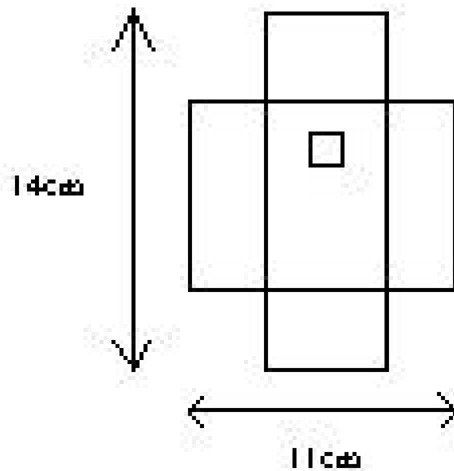


Figure 12. Player Controller Panel

The two pushbuttons are connected to a 5 VDC source and to pins PA0 and PA1 on the HC11. The pins PA0 and PA1 are also connected to a 1000 Ω resistor, which is connected to ground. This had to be done to prevent false inputs to PA0 and PA1 when the pushbuttons are not pressed otherwise an LED would light up inappropriately. We found this out experimentally when we observed an LED lighting up even though we didn't press any buttons. We fixed this phenomenon by grounding the inputs, PA0 and PA1, with a resistor; therefore, whenever a pushbutton is not pressed, the input PA0 and PA1 would always be 0 VDC. However, when a pushbutton is pressed, the input to either PA0 or PA1 would be 5 VDC because the current would flow directly to the pin instead of the resistor since there is less resistance on that path. The pins PA3 to PA6 on the HC11 are dedicated to controlling the 14 LEDs on the 4 x 4 grid-like maze. The output of the pins PA3 to PA6 are from 0000 to 1111 but only 0001 to 1110 could turn on an LED; these four bits would be the inputs to the four-to-sixteen decoder. Since the

four-to-sixteen decoder is an active low device, fourteen inverters are needed to rectify this problem so that only one LED would be on at a time and not all thirteen LEDs on at once otherwise too much current would be drawn. A 150 Ω resistor is connected between each LED and an inverter to prevent the LED from drawing too much current and burning out prematurely. Each LED that was turned on would only last for around 5 seconds before it is turned off which would symbolize a bomb exploding. See figure 13 for the circuitry of the Bombing System.

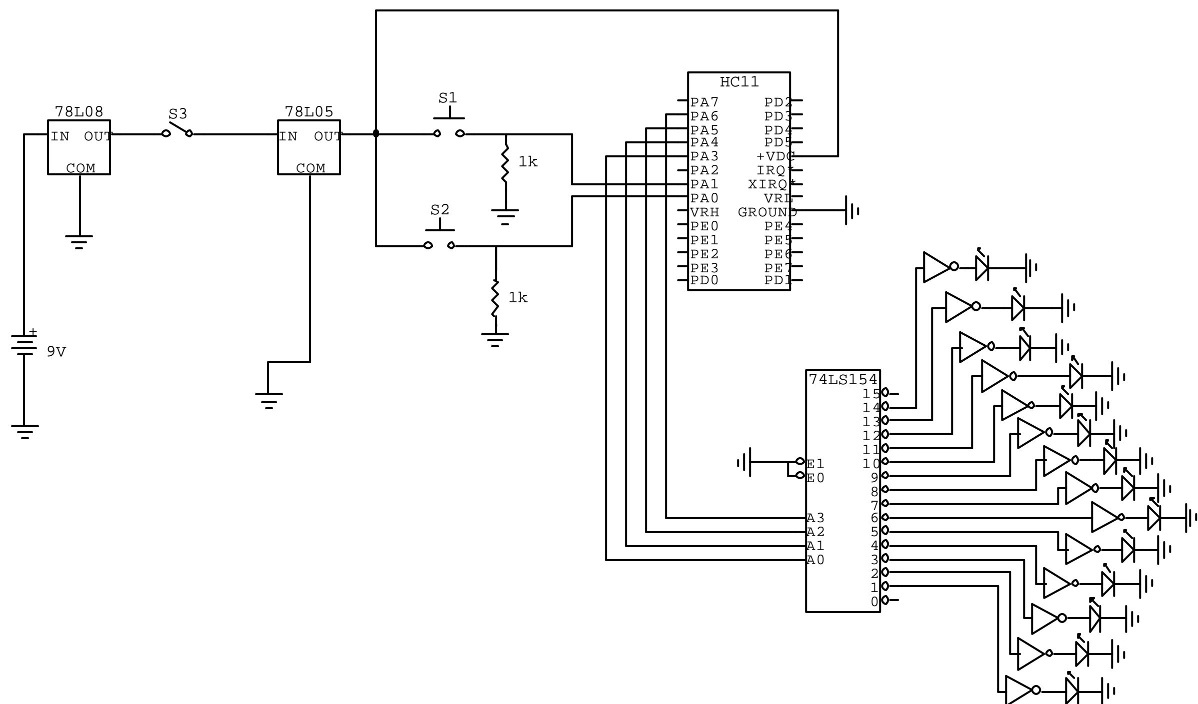


Figure 13. Bombing System Circuit

4.3 Radio Frequency Communication System

The Radio Frequency Communication System was implemented for Bomberbots for its convenience in communicating between the HC11 micro-controller and the Basic Stamp chip since no physical link was needed. Had a serial cable been used, the robots would have maneuvered with much difficulty throughout the maze and possibly tripping over the serial cables. Therefore, RF was chosen for its less cumbersome properties and its ease of use but noise, interference and other distortional effects had to be considered and possibly eliminated which will be discussed further in this section.

In our project, we chose the Linx RM Series Transmitter and Receiver 418 MHz Modules because these modules used a SAW – stabilized FM/FSK modulation scheme. They also have a range in excess of 500 feet, which can support data rates up to 10Kbps. In addition, they also have a wide supply range (5.9 – 9 VDC for the transmitter and 3.9 – 9 VDC for the receiver) and low power consumption (~6mA for the transmitter and ~14mA for the receiver) and they also have a wide operating temperature range (-10 °C to 50 °C). These benefits were essential to our success. See Figure 14 for the block diagram of the Radio Frequency Communication System.

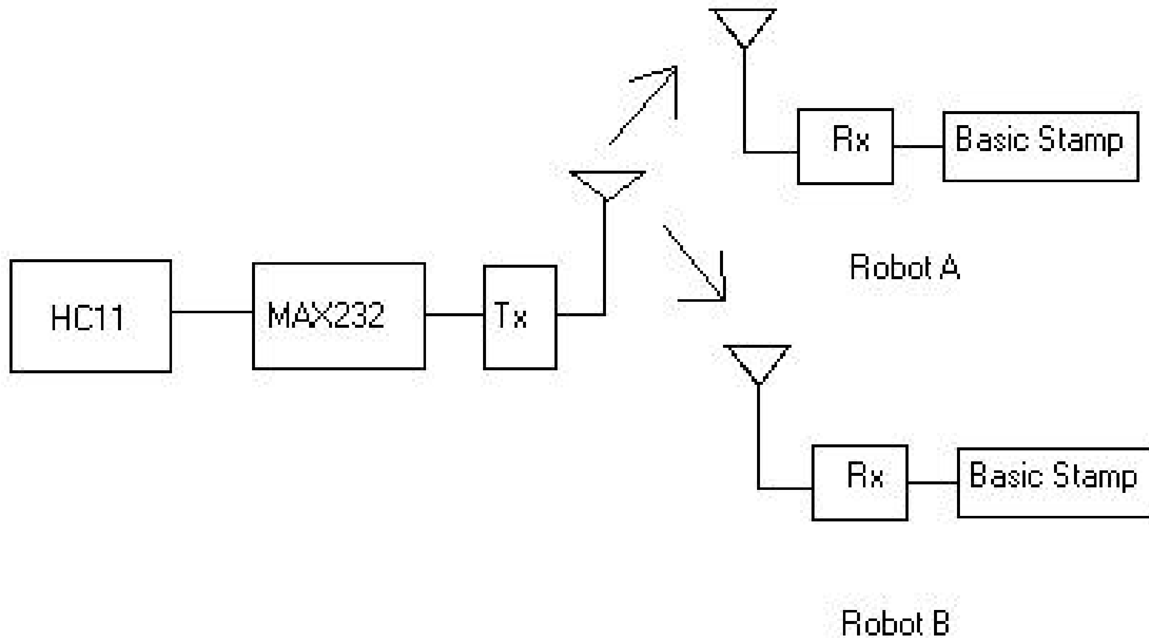


Figure 14. Block Diagram of Radio Frequency Communication System

The transmitter was interfaced to the HC11 micro-controller but a MAX232 was required in between the HC11 micro-controller and the transmitter because the output of the HC11 micro-controller is a RS232 voltage (approx. 15V), which is much greater than what the transmitter can take as data (approx. 0-9V). Therefore, a MAX232 was needed to lower the RS232 voltage level to a TTL/CMOS voltage level (approx. 5V), which is a more appropriate level that the transmitter can accept.

To increase the output power from the antenna, the supply voltage for the transmitter was adjusted to approximately 8 VDC by using a 8 volt regulator to lower the battery voltage

form 9.55 VDC to approximately 8 VDC. This was done to try to overcome the noise in the surrounding environment and hence increase the signal to noise ratio to ensure the receiver would receive our signals easily. See Figure 15 for the transmitter circuitry.

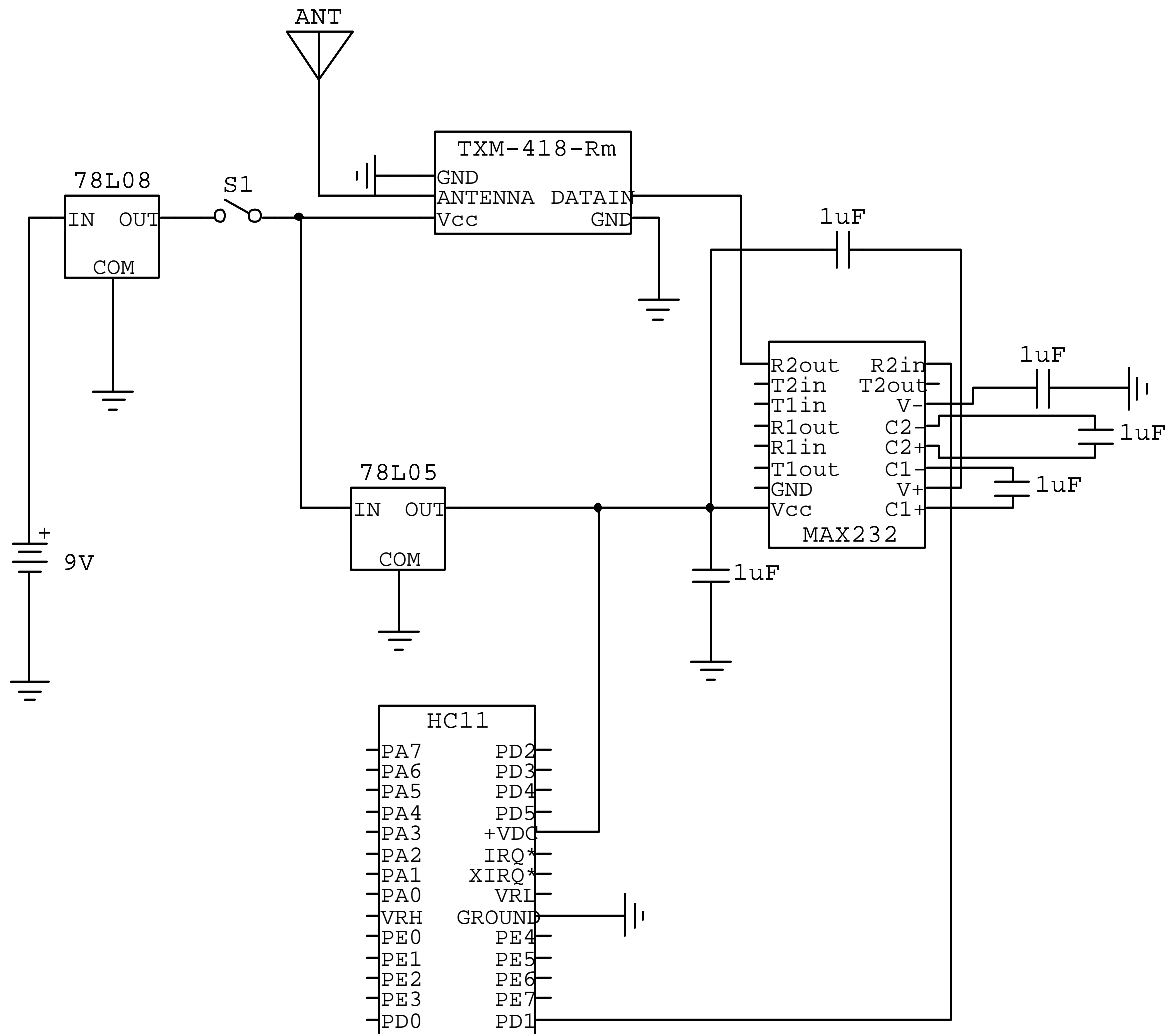


Figure 15. Transmitter Circuit

A transmission protocol was developed to minimize interference. The transmission protocol consisted of sending one junk byte, one synchronization byte and one message byte and this would be replicated ten times. The junk byte was sent to wake up the receiver while the synchronization byte was sent to distinguish which robot we are communicating with. The message byte contains the command for the particular robot. For each command we sent, 30 bytes are sent to a robot. The only time the robot was stimulated is when it recognizes a particular synchronization byte intended for itself. See figure 16 for a block diagram of the transmission protocol.

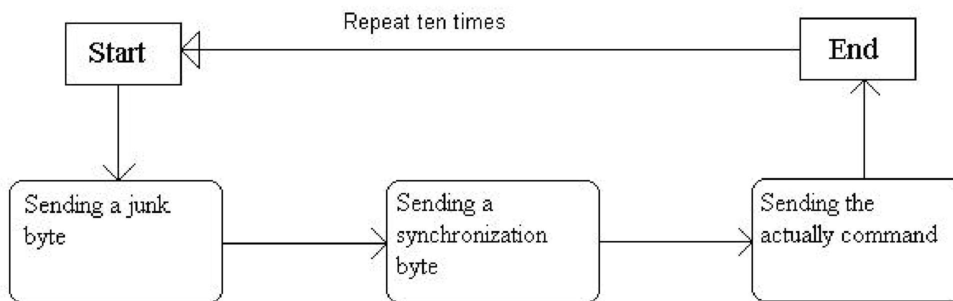


Figure 16. Block Diagram of the Transmission Protocol

5.0 SOFTWARE

There are two main parts to the software for the Bomberbots project: the PBASIC code for the BASIC Stamp 2 and the C code for the HC11. The following sections discuss the general structure for each part of the software. For the actual source code, please refer to **Appendix A** and **Appendix C**.

5.1 Robot Algorithm

The robot uses the BASIC stamp 2 micro-controller as its brain and the software is written in PBASIC. The main tasks for the robot is to receive RF commands from the HC11, move around in the battlefield according to the command, and "explode" if it steps on a bomb. The flow chart for the PBASIC software is shown in **Appendix B**.

At the beginning on each turn, the robot will stand by and wait for RF commands. Once the robot receives some data, it will store it into a variable. The robot will do this for three times and then it will compare the three variables. If the three variables are all the same, then the data received will be considered as a valid command. This is to make sure whatever the robot is receiving from the RF receiver is not noise or junk data from other sources.

Once the command is confirmed to be valid, then the robot will try to execute the command. The robot will go forward, turn left, turn right, or "explode". If the command

is not one of these choices, it will simply go back to the stand by state and wait for the next RF command.

When the robot is instructed to go forward or make a turn, the software will send a pulse signal to drive the servo. At the same time, it will turn on the line sensors to check the white line, making sure that the robot is not going to bump into the walls.

The line following circuit consists of five IR photo sensors. Each of them is connected to one BASIC Stamp pin. Since BASIC Stamp 2 does not support interrupts, a fast polling routine is used to accomplish the same effect. Since the number and size of the tasks involved is small, this technique is fast enough to accomplish the same effect as interrupts.

IF the data received is an “explode” command, the robot will play a song using the buzzer to indicate that it has been hit by a bomb. The software will enter a subroutine and get the song notes and the duration from two lookup tables. Then it will send the corresponding frequency to the buzzer pin.

After the robot executed a command, it will go back to the stand by state and wait for the next RF command. This will repeat until the robot has run out of life.

5.2 Battle Arena Algorithm

In order for the hardware to interact with each other in a harmonious manner, software is required to control all aspects of the game design including lighting up LEDs, interpreting user input and sending commands to the two robots via radio frequency. The Micro-Core 11 contains 32kbytes EEPROM and 32kbytes RAM, which provides a sufficient amount of space for our software program. Also, a high level language called C is used to write the software program and Image Craft Compiler 11 (ICC11) is used to compile and translate the C program code into an executable language called machine language. An executable file is created by ICC11 and the file is directly downloaded to the HC11 via a serial cable. The software is a major part of our project. It is responsible for receiving, interpreting and responding to user input, lighting up LEDs at the desired location and at the appropriate time, sending commands to robots to either move forward, left, right or stop via a transmitter and control the flow of the game.

5.2.1 Receiving Data from User (Pushbutton control)

The signals from the pushbuttons are received using the PORTA input pins. We have dedicated PA0 and PA1 to be the two input pins for player1 and player2. PA0 and PA1 are either connected to ground, which represents logic 0 or connected to 5 VDC, which represents logic 1. When a pushbutton is not pressed, the input pin always receives a logic of 0 but when a button is pressed the input pin receives a logic of 1. Our software algorithm is quite straightforward because we only need to check the least two significant bits of the PORTA register to see which pin has a high input in order to find out which

button was pressed first. Once we find out which button was pressed, the bombing algorithm will be called to light up the desired LED. We chose to use PA0 and PA1 to be our input pins in order to use up PORTA and the leave the other ports for various uses.

5.2.2 Controlling the LEDs

The LEDs are controlled using the pulse width modulation (PWM). This implementation controls how long the LEDs should stay on for a period of time. Our design requires a total of four PWM lines in order to control 14 LEDs. The four PWM lines are output to a 4-to-16 decoder, then each output of the decoder is connected to an inverter and then an LED, since the decoder is active low.

To generate a specific pulse length to fit our purpose, we used the output compare function (OC) to program the LEDs to turn off after a certain time. Each OC has a 16 bit compare register and an output pin associated with it. To operate the OC function, a 16 bit value was assigned to a 16 bit register and whenever the free running counter reaches that value, an interrupt was generated. Then an interrupt handler will implement the appropriate action during the interrupt.

The free running counter in the HC11 chip increments from $0000 to $FFFF every time the program is started. In a 2MHz clock system, the counter takes 32.77ms to count from $0000 to $FFFF. As a result, we discovered the maximum pulse width (32.77ms) was too short for the needs. Therefore, to extend the pulse width, we declared a new counter

in our code and we had to set the OC register to have the value $FFFF. Consequently, an interrupt was generated every 32.77ms. In our interrupt handler, we then increased the new counter we declared by one until our new counter reached 153, which is around five second (32.77ms*153). Afterwards, we then set the OC output pin to low to turn off the LEDs.

The OC registers we used are OC2 to OC5 corresponding to pins PA6 – PA3 respectively. At time 0, the pulse is set to high to turn on the LEDs and when our new counter has reached 153, the pulse is set back to low to turn off the LEDs. Each pin represents a bit for the 4-to-16 decoder, so in order to light up an LED at position (1,1), PA4 and PA6 should be low and PA3 and PA5 should be high. See Table 4 for the bit inputs to the decoder at each location on the game board.

	0	1	2	3
0	**Safe Zone**	0001	0010	0011
1	0100	0101	0110	0111
2	1000	1001	1010	1011
3	1100	1101	1110	**Safe Zone**

Table 4. Bit Inputs to the Decoder at Each Location on the Game Board

Finally, the Vectors.c file needed to be changed so that the interrupt points to the correct interrupt handler. Also, the PACTL register needs to be set so that the four PA6–PA3 pins become output pins.

5.2.3 Sending data to the RF module

The commands to the robots are sent through RF using the Serial Communication Interface (SCI) function of the HC11. The SCI can send a byte of information at a time. A start bit, logic 0, is transmitted or received to indicate the start of each message; an end bit, logic 1, is transmitted or received to indicate the end of each message. The SCI communication consists of a TxD pin for transmitting data, a RxD pin for receiving data and a ground. In a complex implementation of SCI, an interrupt is used to prevent loosing data when receiving and to do some simple error checking. In our case, we are only sending data; therefore, we are not concerned if data is lost or corrupted because we are sending it a repeated number of times.

A busy wait while loop is used to wait for the transmitting register to be available to transmit data. Each command is sent ten times to ensure the robot would receive the command at least three times.

Before the data can be sent, a register called the SCI Status Register (SCSR) needs to be checked. This register is a flag register that stores the status of the transmitting register. The most significant bit of SCSR is called the Transmit Data Register Empty Flag (TDRE). This bit needs to be 1, which means the data register is empty, in order to be able to transmits data. After the status of that bit becomes 1, we store the pre-defined command to the SCI Data Register (SCDR) to be ready to be sent out. Then a shift

register in the HC11 outputs the data to the TxD pin from the least significant bit first to the most significant bit at a baud rate of 2400 to the transmitter.

In our original design, we tried to send data at a baud rate of 9600. However, this transmitting rate was too fast for the Basic Stamp to sample and thus it caused some of the data to be lost during our testing process. After we slowed the transmitting rate down to 2400, all the data was received accurately on the robot side.

5.2.4 Software Algorithm Design

In our game, the HC11 software algorithm is mainly used to control the movement of the robots. At the beginning of the game, the two robots are placed at the starting position, which are positions (0,0) and (3,3) on the game board. After the game is started, each robot walks in a pre-defined path. The current position of each robot is known and will be changed after a command has been sent out to each robot. In our algorithm, in order to send a command, a junk byte is sent first to wake up the receiver and then a synchronization byte is sent to distinguish which robot is being commanded. Finally, the actual command is then sent out. This whole process is repeated ten times to ensure the robot receives our message instead of noise or other messages from other groups. The robots follow a distinct path that we have created through software and eventually they will end up at the starting position and face the same direction as they did at the beginning. If one of the robots is still not dead at that time, then the robots are going to go through the same path again until one of the robots explodes and dies. While the

robots are at the intersection of the maze, the players can press a button to lay a bomb at the robot's current position, which is represented by lighting up an LED. Afterwards, the robot moves to the next intersection according to what command the HC11 sends. A busy wait while loop is used in between each move to make sure the robot is waiting at the intersection for the next command to be sent. We implemented a two dimensional array to keep track of the position of the bomb and the position of each robot. The algorithm checks and compares the position of the bomb and the position of the robot before each move. If the position of the bomb is one, which means the bomb is activated and the position of the robot matches the position of the active bomb then a stop command is sent to the robot to symbolize the robot is exploding. The robot that exploded stays at the intersection for a period of time; meanwhile, the variable that keeps track of the life of the robot is decreased by one. If the life of the robot reaches zero, the program will stop and the other player wins. The flow of our algorithm is to wait for the robot to reach the intersection, check for the bomb position, check for the players' button and finally send a command to the robot for the next move. This sequence is repeated until one of the players dies.

6.0 Budget

Quantity	Part	Source	Price	Amount
1	MicroCore-11	EE Department	$100	$100.00
4	Futaba FP-S148 Servo	Vancouver Robotics	$29.95	$119.80
2	Basic Stamp II	Digikey	$77.42	$154.84
2	RF Receiver 418MHz	Digikey	$57.59	$115.18
2	RF Transmitter 418MHz	Digikey	$35.26	$70.52
10	QRB1114 (photo diodes)	Digikey	$2.00	$20.00
N/A	Miscellaneous components	EE Department	N/A	$35.00
			Total	$615.34

7.0 SUMMARY

The mechanical robot game Bomberbots was inspired by the classic video game called "BomberMan" created by Hudson Soft and is designed specially for a disabled boy with the objective for him to interact with friends. The game consists of two controller panels, two robots, and a battle arena board with the console unit inside the control tower. The HC11 microprocessor keeps track of where the robots are on the arena board and accepts inputs from the player controller panel. It also outputs instructions to the two robots via a transmitter and sets bomb through LED activation. Each Bomberbot is controlled by one BASIC Stamp 2 microprocessor that receives coded data from its RF receiver module. The coded digital signal consists of a junk byte, synchronization byte, and a message byte. The robots use IR sensors attached to its bottom side to follow white lines on the grid, which prevent it from bumping into the pillars and the border walls in the battle arena. The drive system of the robot consists of two servos and a third supporting wheel. The battle arena consists of a 4 x 4 square grid with 14 LED bombs placed in the middle of each intersections on the board except for the "safe-zone". The circuitry to the 14 LEDs is just a simple four-to-sixteen decoder connected to the HC11 microprocessor. As for the player controller panel, it is a hand-held aluminum box with a single button to activate bombs that is wired to the console. The HC11 processes this data and outputs the necessary information through wireless transmission and activating LED bombs. The software algorithm makes proper adjustments to the movement of each robot so that they will never cross paths and collide.

8.0 RECOMMENDATIONS

Throughout the course of the project, our group has encountered many modification concepts that would improve the design of our robots, battle arena, and game play. Some of these changes were made to ensure a properly working game but any non-vital modifications were not implemented due to cost restrictions, time constraints, and insufficient equipment. The following improvements were discovered through Internet research, recommendations from robotics experts, textual reading, and discussions with previous EECE 474 students:

- A larger board with more grids and more room for each square of intersection will increase the level of complexity in the game
- Each robot may be allowed to activate more than one bomb at a time
- Build a joystick on the player controller panel to allow them to control the movement of each robot
- Use both a transmitter and a receiver on both robots and the main control unit to maintain a constant two-way communication to ensure proper operation.
- Replace IR emitters with ultra bright LEDs to ease the debugging process and for the "good look"
- Implement "ultra bombs" which, when explode, effects more than one grid space

APPENDIX A: PBASIC SOURCE CODE

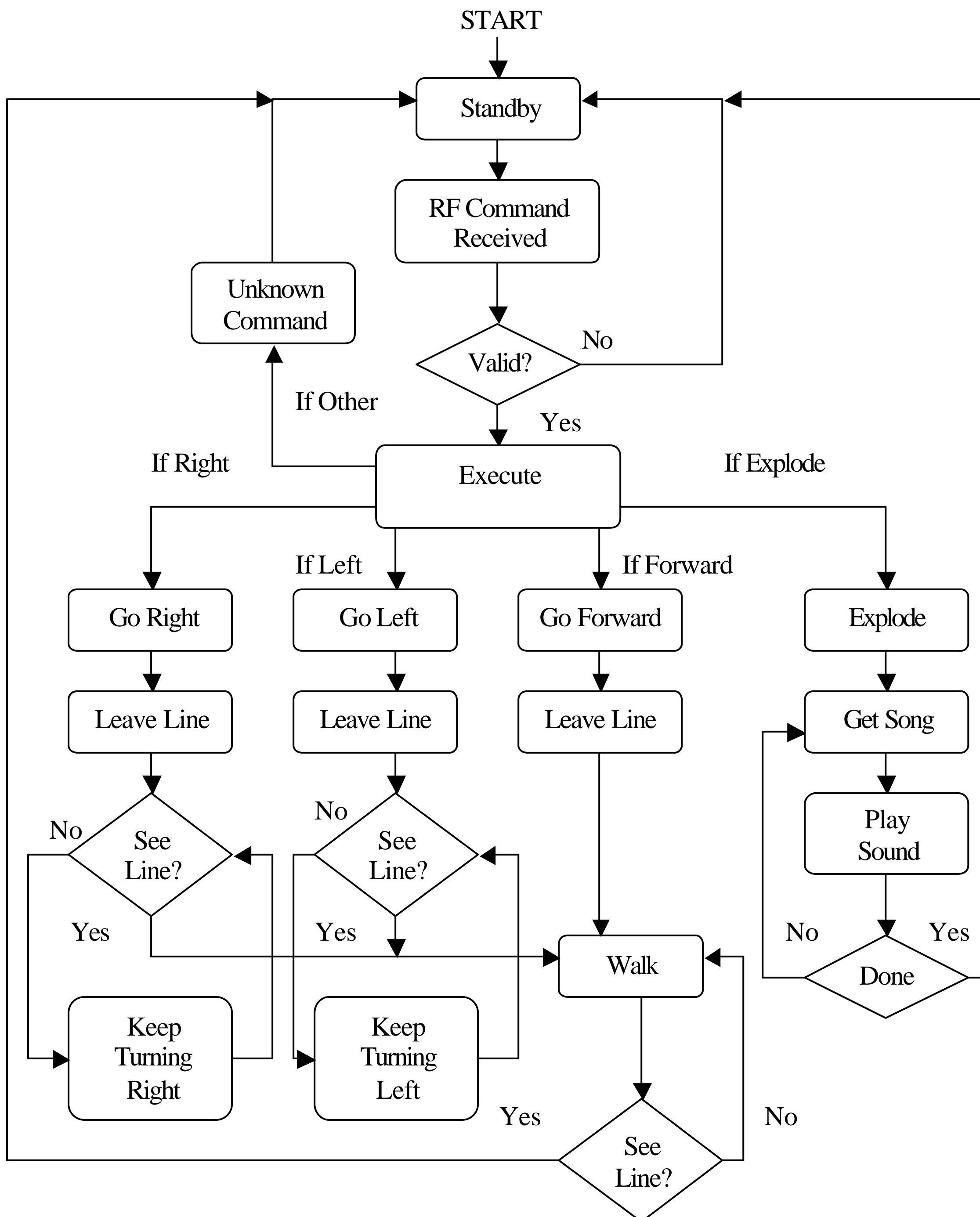
```
'{$STAMP BS2}
'
' -----[ Title ]-----------------------------------------------------
'
' File:      BOMBERBOT.BS2
' Purpose:   Software algorithm for the bomberbot project, code include
'            line following sensors, RF, and bomb algorithm
' Author:    Peter Chan
'            Sunny Chan
'
' -----[ Program Description ]---------------------------------------
'
' This program allows the robot to receive and execute RF commands
' accordingly.  The RF commands directs the robot to go either left,
' right, or forward and whether it will be stepping on a bomb or not.
'
' -----[ Revision History ]------------------------------------------
'
' 19 March 2002 - Start Date
'                 Added Line Following Algorithm
' 20 March 2002 - Added RF Algorithm
' 23 March 2002 - Added Bomb Algorithm
'
' -----[ I/O Definitions ]-------------------------------------------
'
LServo      CON   8                     ' servo motor connections
RServo      CON   9

Receiver    CON   10                    ' RF receiver

Buzzer      CON   15                    ' song buzzer
'
' -----[ Constants ]-------------------------------------------------
'
LEDon       CON   0                     ' LF LEDs are active low
LEDoff      CON   1

WLine       CON   0                     ' white line on black field
BLine       CON   1                     ' black line on white field
LFmode      CON   WLine                 ' set pgm for white line

MStop       CON   750                   ' motor stop
SpeedHigh   CON   60                    ' high speed
SpeedLow    CON   30                    ' low speed

Synch       CON   %00001010             ' synchronization byte (0A in
HEX)
Baud        CON   396                   ' 2400 baud
'
' -----[ Variables ]-------------------------------------------------
'
temp        VAR   Byte                  ' for loop counter
ledPos      VAR   Nib                   ' LED position in lfBits
lfBits      VAR   Byte                  ' line follower input bits
```

```
redat       VAR   Byte                    ' RF data received
data1       VAR   Byte                    ' RF data storage 1
data2       VAR   Byte                    ' RF data storage 2
data3       VAR   Byte                    ' RF data storage 3
index       VAR   Byte                    ' lookup table index for song
note        VAR   Word                    ' song note
duration    VAR   Word                    ' song note duration
'
' -----[ Initialization ]--------------------------------------------
'
OutL = %01111100                          ' all LF LEDs off (pins 2 to 6
high)
DirL = %01111100                          ' make LF LED pins outputs

index = 0                                 ' start of song
'
' -----[ Main Code ]-------------------------------------------------
'
PAUSE 2000                                ' program starts running after
2 sec

Main:
  SERIN Receiver,Baud,[WAIT(Synch),redat]
  data1 = redat                                ' storing to data1
  SERIN Receiver,Baud,[WAIT(Synch),redat]
  data2 = redat                                ' storing to data2
  SERIN Receiver,Baud,[WAIT(Synch),redat]
  data3 = redat                                ' storing to data3

  IF data1 = data2 AND data1 = data3 THEN Execute     ' check RF data
  GOTO Main

Execute:
  IF data1 = %00000001 THEN Go_Forward         ' execute RF command
  IF data1 = %00000010 THEN Go_Right
  IF data1 = %00000011 THEN Go_Left
  IF data1 = %00001111 THEN Explode
  GOTO Main

Go_Forward:                                    ' robot is going
straight
  FOR temp = 1 TO 100
      PULSOUT RServo,MStop - SpeedHigh
      PULSOUT LServo,MStop + SpeedHigh
  NEXT
  loop_forward:
  GOSUB Read_Line_Follower
  IF (lfBits <> %11111) THEN Move_Forward
  IF (lfBits = %11111) THEN Walk

Go_Right:                                      ' robot is going right
  FOR temp = 1 TO 100
      PULSOUT LServo,MStop + SpeedHigh
  NEXT
  loop_right:
  GOSUB Read_Line_Follower
  IF (lfBits <> %11111) THEN Turn_Right
```

```
  IF (lfBits = %11111) THEN Brake

Go_Left:                                               ' robot is going left
  FOR temp = 1 TO 100
      PULSOUT RServo,MStop - SpeedHigh
  NEXT
  loop_left:
  GOSUB Read_Line_Follower
  IF (lfBits <> %11111) THEN Turn_Left
  IF (lfBits = %11111) THEN Brake

Explode:
  GOTO Play_Song                                       ' check if robot is hit
  Done_Play_Song:
  index = 0                                            ' reset song index
  GOTO Main

Walk:                                                  ' walk forward
  FOR temp = 1 TO 100
      PULSOUT RServo,MStop - SpeedHigh
      PULSOUT LServo,MStop + SpeedHigh
  NEXT
  loop_walk:
  GOSUB Read_Line_Follower
  IF (lfBits <> %11111) THEN Keep_Walking
  IF (lfBits = %11111) THEN Done

Turn_Left:                                             ' keep turning left
  PULSOUT RServo,MStop - SpeedLow
  GOTO loop_left

Turn_Right:                                            ' keep turning right
  PULSOUT LServo,MStop + SpeedLow
  GOTO loop_right

Move_Forward:                                          ' keep walking
  PULSOUT LServo,MStop + SpeedLow
  PULSOUT RServo,MStop - SpeedLow
  GOTO loop_forward

Keep_Walking:                                          ' keep walking
  PULSOUT LServo,MStop + SpeedLow
  PULSOUT RServo,MStop - SpeedLow
  GOTO loop_walk


Brake:                                                 ' brake
  PULSOUT LServo,MStop
  PULSOUT RServo,MStop
  Pause 500
  GOTO Walk

Done:
  Pause 1500
  FOR temp = 1 TO 5                                    ' back up a little
    PULSOUT LServo,MStop - SpeedHigh
    PULSOUT RServo,MStop + SpeedHigh
```

```
  NEXT
  pause 2000
  GOTO Main

Play_Song:
  GOSUB Get_Song
  IF note = 0 THEN Done_Play_Song               ' finish playing song
  FREQOUT Buzzer,duration,note                  ' send signal to buzzer
  index = index + 1                             ' index to next note
  GOTO Play_Song
'
' -----[ Subroutines ]-----------------------------------------------
'
Read_Line_Follower:
  lfBits = 0                                    ' clear last reading
  FOR ledPos = 2 TO 6
      OutL.LowBit(ledPos) = LEDon               ' turn the LED on
      PAUSE 1                                   ' allow sensor to read
      lfBits.LowBit(ledPos) = In7 ^ LFmode      ' record sensor reading
      OutL = OutL | %01111100                   ' turn LEDs off
  NEXT
  lfBits = lfBits >> 2                          ' shift bits to zero
index
  RETURN

Get_Song:
  LOOKUP index,[550,450,350,250,150, 50,0],note       ' song note
  LOOKUP index,[500,500,500,500,500,500],duration     ' note duration
  RETURN
'
' -----[ End of Program ]--------------------------------------------
```

APPENDIX B: FLOW CHART FORPBASIC SOFTWARE



APPENDIX C: C SOURCE CODE

```
/***********************************************
      FILES NEEDED TO BE INCLUDED
***********************************************/

         #include <stdio.h>
         #include <hc11.h>

/***********************************************
      FUNCTION DECLARATION
***********************************************/

void    setup (void);                    //The function setup all the initial values
int bombbutton1 ();                //The function check the button press by the user
int bombbutton2 ();          //The function check the position of the bomb and robot
void timercontrol1andmask1 (int x, int y);          //The function that setup the timer
                                                     //control and mask register
void timerflag1 (int x, int y);          //The function that setup the timer flag register
void checkbomb ();                                  //The check bomb function
void transmitwait ();                    //The function wait for the recieve signal
void update1 ();
void update2 ();
void sendingdata1 ();                    //The function that send the command
void sendingdata2 ();
void delay();                                          //A delay function
void delay2();                                         //Another delay function

/***********************************************
      VARIABLES
***********************************************/


         volatile int bombbit1;
         volatile int bombbit2;
         volatile char data;          //The data recieved
         volatile int flag1x, flag2x, flag1y, flag2y;


         int robot1state;
         int robot2state;

         int robot1pos[4][4];          //The array store the position of the robot 1
         int robot1x[5];
         int robot1y[5];
```

```
volatile int bot1x;
volatile int bot1y;

int robot2pos[4][4];            //The array store the position of the robot 2
int robot2x[5];
int robot2y[5];
volatile int bot2x;
volatile int bot2y;

int bombsloc[4][4];             //The array keep track of the position of the bomb

volatile int life1;                             //The life variable for robot 1
volatile int life2;                             //The life variable for robot 2

volatile int z;          //The counter in the checking robot function, delay function

volatile int i, j, a, b;                  //The counter in the setup function

volatile int c;                           //The counter in the sending data

volatile int d;                           //The counter in the delay2 function

/************************************************
        DEFINE  CONSTANTS
************************************************/

volatile int pulse_width=0x0BB8;    /*1.50ms*/
volatile int count=0;
volatile int check1=0;
volatile int check2=0;

#ifndef BOMBERGUY_C
#define BOMBERGUY_C

#define forward         0xC3        /* 1100 0011    forward command for robot*/
#define right           0xD2        /* 1101 0010    right command for robot 1*/
#define left            0xE1        /* 1110 0001    left command for robot 1*/
#define stop            0xF0        /* 1111 0000    stop command for robot 1*/
#define sync1   0xAA  /* 1010 1010    the bit use to communicate with robot 1*/
#define sync2   0xBB  /* 1011 1011    the bit use to communicate with robot 2*/
#define junk            0x00                /* 0000 0000    the junk bit*/

#endif

/************************************************
    INTERRUPT SERVICE ROUTINES
```

```
***********************************************/

        #pragma interrupt_handler fiveHandler


/***************************************************************************
     SETUP FUNCTION
***************************************************************************/


void setup(void)
        {
                setbaud(BAUD9600);

                PACTL |= 0x88;                  /* Set Port A, bits 3 & 7 as outputs. */

                SCCR1 |= 0x00;
                SCCR2 |= 0x0C;          /* Set up the register for sci serial communication */



                for ( i = 0; i < 4; i++)
                        {
                                for ( j = 0; j < 4; j++)
                                        {
                                                robot1pos[i][j] = 0;
                                                robot2pos[i][j] = 0;
                                        }
                        }



                robot1state = 0;
                robot2state = 0;
                bombbit1 = 0;
                bombbit2 = 0;
                life1 = 300;
                life2 = 300;

                robot1pos[0][0] = 1;
                robot2pos[3][3] = 1;
                bot1x = 0;
                bot1y = 0;
                bot2x = 3;
                bot2y = 3;
```

```
/**********Initializing the path of the robot but parts are skip for simplicity******/

                robot1x[0] = 0;
                robot1y[0] = 0;
                robot1x[1] = 1;
                robot1y[1] = 0;
                robot1x[2] = 1;
                robot1y[2] = 1;
                robot2x[0] = 3;
                robot2y[0] = 3;
                robot2x[1] = 3;
                robot2y[1] = 2;
                robot2x[2] = 2;
                robot2y[2] = 2;

/**********Initializing the path of the robot but parts are skip for simplicity******/

                for ( a = 0; a < 4; a++)
                        {
                                for ( b = 0; b < 4; b++)
                                        {
                                                bombsloc[a][b] = 0;
                                        }
                        }
        }


/**********************************************************************
    CHECK IF A ROBOT IS ON A BOMB FUNCTION
**********************************************************************/

void checkbomb()
        {

                if (robot1pos[bot1x][bot1y] == bombsloc[bot1x][bot1y])
                {
                        //This case is enter only when player step on bomb

                        if ( life1 > 0 )
                        {
                                life1--;
                                bombbit1 = 1;

                        while(c<10)
                                {
```

```
                // sending a stop command

                        transmitwait();
                        SCDR = junk;
                        transmitwait();
                        SCDR = sync1;
                        transmitwait();
                        SCDR = stop;
                        delay();
                        c = c+1;
                }

                c = 0;
        }
        else
        {
                exit(0);
        }
}


if (robot2pos[bot2x][bot2y] == bombsloc[bot2x][bot2y])
{
        if ( life2 > 0 )
        {
                life2--;
                bombbit2 = 1;

        while(c<10)
                {
                        transmitwait();
                        SCDR = junk;
                        transmitwait();
                        SCDR = sync2;
                        transmitwait();
                        SCDR = stop;
                        delay();
                        c = c+1;
                }

                c = 0;

        }
        else
        {
                exit(0);
```

```
                }
        }


    }

/****************************************************************************
    CHECK IF THE BUTTON IS PRESSED FUNCTION
****************************************************************************/



void checkbutton()
    {


        if (robot2pos[bot2x][bot2y] == 1)
        {
            if (!bombbutton2 ())
            {
                count = 0;
                check2 = 1;
                flag2x = bot2x;
                flag2y = bot2y;

                timercontrol1andmask1 (bot2x,bot2y);

                asm("cli");

                pulse_width=0xFFFF;
            }
        }

        if (robot1pos[bot1x][bot1y] == 1)
        {
            if (!bombbutton1 ())
            {
                count = 0;
                check1 = 1;
                flag1x = bot1x;
                flag1y = bot1y;

                timercontrol1andmask1 (bot1x,bot1y);

                asm("cli");
```

```
                        pulse_width=0xFFFF;
                    }
            }

        }


/*  If an output compare register matches the free-running counter and the
    corresponding bits of the TCTL1 and TMSK1 are set,  this interrupt subroutine
    will be called            */

void fiveHandler(void)
{

   /*     The first thing we have to do is to enable some of the bits in TFLG1

          bit          7       6        5        4        3        2      1      0
          TFLG1   0C1F OC2F OC3F OC4F I4/O5F IC1F IC2F IC3F                  */


  /*  set the corresponding bit of the TFLG1 to represent the output pin that we want to
generate */

                if (check1 == 1)
                    {
                        timerflag1 (flag1x,flag1y);
                    }

                if (check2 == 1)
                    {
                        timerflag1 (flag2x,flag2y);
                    }


    TOC2 = pulse_width;
    TOC3 = pulse_width;
    TOC4 = pulse_width;
    TOC5 = pulse_width;



   /* By doing this, when later the free-running counter reaches PULSE_WIDTH, it will
call this interrupt service
    subroutine */
```

```
    count = count + 1;      /* count up to 153 to represent 5 sec */

    /* With this command, it will set the OLX (where X is the corresponding number for
your preferred output pin).
    This will cause the output pin changes from high to low after five seconds. */

    if (count == 153)
    {
          TCTL1 = (TCTL1 & 0xAA);

      bombsloc[flag1x][flag1y] = 0;
/* This reset the location of the bomb to false*/

      bombsloc[flag2x][flag2y] = 0;
/* This reset the location of the bomb to false*/

            bombbit1 = 0;
            bombbit2 = 0;
          }

}


int bombbutton1 ()
          {
                              if ((PORTA & 0x01) == 0)
                                        return (1);
                              else
                                        return (0);
          }

int bombbutton2 ()
          {
                    if ((PORTA & 0x02) == 0)
                              return (1);
                    else
                              return (0);
          }

// This function setups the appropriate register for timing conrtrol

void timercontrol1andmask1 (int x, int y)
          {
                    if ((x == 0) && (y == 0))
```

```
{
        TCTL1 = TCTL1 | 0x00 ;
        TMSK1 = TMSK1 | 0x00 ;
}

else if ((x == 0) && (y == 1))
{
        TCTL1 = TCTL1 | 0x30 ;
        TMSK1 = TMSK1 | 0x20 ;
}

else if ((x == 0) && (y == 2))
{
        TCTL1 = TCTL1 | 0xC0 ;
        TMSK1 = TMSK1 | 0x40 ;
}

else if ((x == 0) && (y == 3))
{
        TCTL1 = TCTL1 | 0xF0 ;
        TMSK1 = TMSK1 | 0x60 ;
}

else if ((x == 1) && (y == 0))
{
        TCTL1 = TCTL1 | 0x03 ;
        TMSK1 = TMSK1 | 0x08 ;
}

else if ((x == 1) && (y == 1))
{
        TCTL1 = TCTL1 | 0x33 ;
        TMSK1 = TMSK1 | 0x28 ;
}

else if ((x == 1) && (y == 2))
{
        TCTL1 = TCTL1 | 0xC3 ;
        TMSK1 = TMSK1 | 0x48 ;
}

else if ((x == 1) && (y == 3))
{
        TCTL1 = TCTL1 | 0xF3 ;
        TMSK1 = TMSK1 | 0x68 ;
}
```

```
else if ((x == 2) && (y == 0))
{
        TCTL1 = TCTL1 | 0x0C ;
        TMSK1 = TMSK1 | 0x10 ;
}

else if ((x == 2) && (y == 1))
{
        TCTL1 = TCTL1 | 0x3C ;
        TMSK1 = TMSK1 | 0x30 ;
}

else if ((x == 2) && (y == 2))
{
        TCTL1 = TCTL1 | 0xCC ;
        TMSK1 = TMSK1 | 0x50 ;
}

else if ((x == 2) && (y == 3))
{
        TCTL1 = TCTL1 | 0xFC ;
        TMSK1 = TMSK1 | 0x70 ;
}

else if ((x == 3) && (y == 0))
{
        TCTL1 = TCTL1 | 0x0F ;
        TMSK1 = TMSK1 | 0x18 ;
}

else if ((x == 3) && (y == 1))
{
        TCTL1 = TCTL1 | 0x3F ;
        TMSK1 = TMSK1 | 0x38 ;
}

else if ((x == 3) && (y == 2))
{
        TCTL1 = TCTL1 | 0xCF ;
        TMSK1 = TMSK1 | 0x58 ;
}

else if ((x == 3) && (y == 3))
{
        TCTL1 = TCTL1 | 0xFF ;
```

```
                TMSK1 = TMSK1 | 0x78 ;
            }
        }


void timerflag1 (int x, int y)
        {
            if ((x == 0) && (y == 0))
            {
                TFLG1 = 0x00 ;
            }

            else if ((x == 0) && (y == 1))
            {
                TFLG1 = 0x20 ;
            }

            else if ((x == 0) && (y == 2))
            {
                TFLG1 = 0x40 ;
            }

            else if ((x == 0) && (y == 3))
            {
                TFLG1 = 0x60 ;
            }

            else if ((x == 1) && (y == 0))
            {
                TFLG1 = 0x08 ;
            }

            else if ((x == 1) && (y == 1))
            {
                TFLG1 = 0x28 ;
            }

            else if ((x == 1) && (y == 2))
            {
                TFLG1 = 0x48 ;
            }

            else if ((x == 1) && (y == 3))
            {
                TFLG1 = 0x68 ;
```

```
	}

	else if ((x == 2) && (y == 0))
	{
		TFLG1 = 0x10 ;
	}

	else if ((x == 2) && (y == 1))
	{
		TFLG1 = 0x30 ;
	}

	else if ((x == 2 && y == 2))
	{
		TFLG1 = 0x50 ;
	}

	else if ((x == 2) && (y == 3))
	{
		TFLG1 = 0x70 ;
	}

	else if ((x == 3) && (y == 0))
	{
		TFLG1 = 0x18 ;
	}

	else if ((x == 3) && (y == 1))
	{
		TFLG1 = 0x38 ;
	}

	else if ((x == 3) && (y == 2))
	{
		TFLG1 = 0x58 ;
	}

	else if ((x == 3) && (y == 3))
	{
		TFLG1 = 0x78 ;
	}
}
```

```
/*************************************************************************
      COMMAND FUNCTION
*************************************************************************/



void sendingdata1 ()
{
        /* output command to rotbot1 */

        switch (robot1state)
        {
                case 0:
                        {
                                while(c<10)
                                {
                                        transmitwait();
                                        SCDR = junk;
                                        transmitwait();
                                        SCDR = sync1;
                                        transmitwait();
                                        SCDR = forward;
                                        delay();
                                        c = c+1;
                                }

                                c = 0;

                        while(c<10)
                        {
                                        transmitwait();
                                        SCDR = junk;
                                transmitwait();
                                        SCDR = sync1;
                                transmitwait();
                                        SCDR = right;
                                delay();
                                        c = c+1;
                                }

                                c = 0;

                                                /*send forward to R1
                                                  send turn right to R1*/
```

```
				update1();
				/* update the robot position */

			/*each case statment stand for each move of the robots*/
			}

		case 1:
			{
				while(c<10)
				{
					transmitwait();
					SCDR = junk;
				transmitwait();
					SCDR = sync1;
				transmitwait();
					SCDR = forward;
				delay();
				c = c+1;
			}

			c = 0;

			while(c<10)
			{
					transmitwait();
					SCDR = junk;
				transmitwait();
					SCDR = sync1;
				transmitwait();
					SCDR = right;
				delay();
					c = c+1;
				}

				/*send forward to R1
				  send turn right to R1*/

				update1();					/* update the
robot position */

				/*each case statment stand for each move of the robots*/
			}

		case 2:
			{
				while(c<10)
```

```
		{
			transmitwait();
			SCDR = junk;
		transmitwait();
			SCDR = sync1;
		transmitwait();
			SCDR = forward;
		delay();
		c = c+1;
	}

	c = 0;

	while(c<10)
	{
			transmitwait();
			SCDR = junk;
		transmitwait();
			SCDR = sync1;
		transmitwait();
			SCDR = right;
		delay();
			c = c+1;
		}

		c = 0;


	/*send forward to R1
	  send turn right to R1*/

		update1();			/* update the robot position */

	/*each case statment stand for each move of the robots*/
	}

case 3:
	{

		while(c<10)
		{
			transmitwait();
			SCDR = junk;
		transmitwait();
			SCDR = sync1;
		transmitwait();
```

```
					SCDR = forward;
				delay();
				c = c+1;
			}

			c = 0;

			while(c<10)
			{
					transmitwait();
					SCDR = junk;
				transmitwait();
					SCDR = sync1;
				transmitwait();
					SCDR = right;
				delay();
					c = c+1;
				}

				c = 0;


			/*send forward to R1
			  send turn right to R1*/

				update1();			/* update the robot position */

				/*each case statment stand for each move of the robots*/
			}

	}


}


void sendingdata2 ()
{
	/* output command to rotbot2 */

	switch (robot2state)
	{
		case 0:
			{

				while(c<10)
```

```
            {
                transmitwait();
                SCDR = junk;
            transmitwait();
                SCDR = sync2;
            transmitwait();
                SCDR = forward;
            delay();
            c = c+1;
        }

        c = 0;

        while(c<10)
        {
                transmitwait();
                SCDR = junk;
            transmitwait();
                SCDR = sync2;
            transmitwait();
                SCDR = left;
            delay();
                c = c+1;
            }

            c = 0;

                            /*send forward to R2
                              send turn left to R2*/

            update2();                  /* update the robot position */

            /*each case statment stand for each move of the robots*/
        }

case 1:
        {

        while(c<10)
            {
                transmitwait();
                SCDR = junk;
            transmitwait();
                SCDR = sync2;
            transmitwait();
                SCDR = forward;
```

```
            delay();
            c = c+1;
        }

        c = 0;

        while(c<10)
        {
                transmitwait();
                SCDR = junk;
            transmitwait();
                SCDR = sync2;
            transmitwait();
                SCDR = left;
            delay();
                c = c+1;
            }

            c = 0;

            /*send forward to R2
              send turn left to R2*/

            update2();              /* update the robot position */

            /*each case statment stand for each move of the robots*/
        }

case 2:
        {

        while(c<10)
            {
                transmitwait();
                SCDR = junk;
            transmitwait();
                SCDR = sync2;
            transmitwait();
                SCDR = forward;
            delay();
            c = c+1;
        }

        c = 0;

        while(c<10)
```

```
        {
                transmitwait();
                SCDR = junk;
            transmitwait();
                SCDR = sync2;
            transmitwait();
                SCDR = left;
            delay();
                c = c+1;
            }

            c = 0;

        /*send forward to R2
         send turn left to R2*/

            update2();          /* update the robot position */

        /*each case statment stand for each move of the robots*/
        }

case 3:
        {


        while(c<10)
            {
                transmitwait();
                SCDR = junk;
            transmitwait();
                SCDR = sync2;
            transmitwait();
                SCDR = forward;
            delay();
            c = c+1;
        }

        c = 0;

        while(c<10)
        {
                transmitwait();
                SCDR = junk;
            transmitwait();
                SCDR = sync2;
            transmitwait();
```

```
                SCDR = left;
            delay();
                c = c+1;
            }

            c = 0;

        /*send forward to R1
         send turn right to R1*/

            update2();              /* update the robot position */

            /*each case statment stand for each move of the robots*/
        }

    }


}
/************In the sendingdata1 and sendingdata2 function only first few moves are
shown for simplicity*****************************************/

void update1 ()
{
    //Update the position of robot1

    robot1pos[bot1x][bot1y] = 0;
    bot1x = robot1x[robot1state];
    bot1y = robot1y[robot1state];
    robot1pos[bot1x][bot1y] = 1;

    robot1state = robot1state + 1;

}


void update2 ()
{
    //Update the position for robot2

    robot2pos[bot2x][bot2y] = 0;
    bot2x = robot2x[robot2state];
    bot2y = robot2y[robot2state];
    robot1pos[bot2x][bot2y] = 1;

    robot2state = robot2state + 1;
```

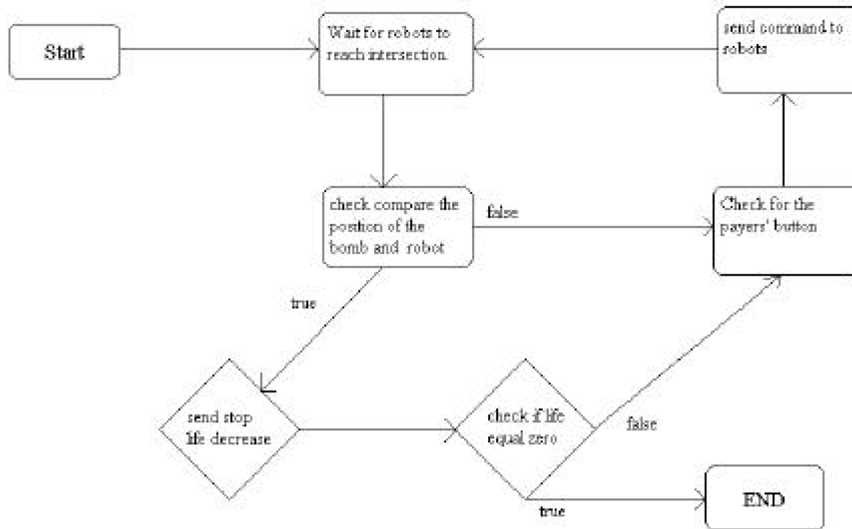
```
}


void transmitwait()
{
        /* A function to wait for the SCDR to be empty i.e ready to send */

        while(!(SCSR & 0x80))
                {
                }
}



/*****************************************************************************
      DELAY FUNCTION
*****************************************************************************/

void delay()
{
         /* A function to delay after sending signal */

        z = 0;
        while( z < 2000 )
                {
                        z = z+1;
                }
}

void delay2()
{
         /* A function to delay after sending signal */

        d = 0;
        while( d < 200000 )
                {
                        d = d+1;
                }
}



/***********************************************************
      MAIN FUNCTION
***********************************************************/
```

```
void main (void)
	{
		setup ();
		/*puts ("\nBomber Man");*/

		while (1)
			{

				delay2();

				checkbomb();

				checkbutton();

				if (!(bombbit1 == 1))
				{
	//if the robot1 is not bomb send the next command
					sendingdata1();
				}

				if (!(bombbit2 == 1))
				{
	//if the robot2 is not bomb send the next command
					sendingdata2();
				}


			}
	}

/* If you need to set up interrupt vectors (e.g. single chip mode system
 * or system without monitor, then you can simply include vectors.c. The
 * "right" way to do things is to set up a project with mulitple files,
 * but to just try out the compiler, this works just as well.
 * HC16's vector is in the crt16.o file
 */
#if defined(_HC11) || defined(_HC12)
#include "vectors.c"
#endif
/* note that since vectors.c uses pragma to change the text section name
 * there should not be stuff after this unless you change the name
 * back
 */
```

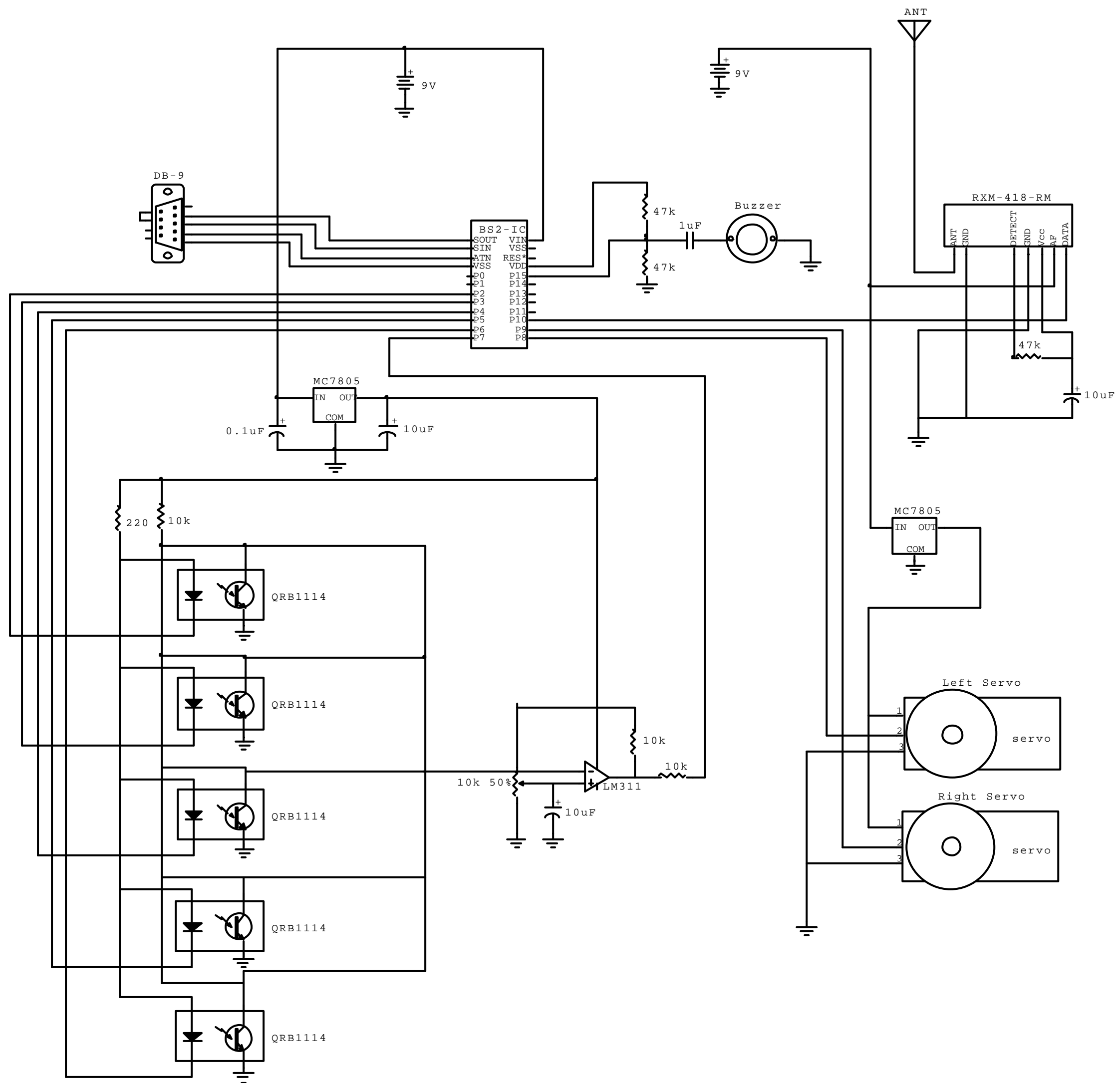
```
/* As is, all interrupts except reset jumps to 0xffff, which is most
 * likely not going to useful. To replace an entry, declare your
function,
 * and then change the corresponding entry in the table. For example,
 * if you have a SCI handler (which must be defined with
 * #pragma interrupt_handler ...) then in this file:
 * add
 *        extern void SCIHandler();
 * before th table.
 * In the SCI entry, change:
 *        DUMMY_ENTRY,
 * to
 *  SCIHandler,
 */
extern void _start(void);            /* entry point in crt??.s */
extern void fiveHandler(void);  /* Generates PWM */

#define DUMMY_ENTRY       (void (*)(void))0xFFFF

#ifdef _HC12
#pragma abs_address:0xffd0
#else    /* HC11 */
#pragma abs_address:0xffd6
#endif

/* change the above address if your vector starts elsewhere
 */
void (*interrupt_vectors[])(void) =
        {
        /* to cast a constant, say 0xb600, use
          (void (*)())0xb600
         */
#ifdef _HC12
        /* 812A4 vectors starts at 0xff80, but most entries are not used
          if you use Key Wakeup H, change the start address to 0xffCE and
          add one entry to the beginning */
        DUMMY_ENTRY,     /* BDLC */                    /* Key Wakeup J */
        DUMMY_ENTRY,     /* ATD */                     /* ATD */
```

```
        DUMMY_ENTRY,    /* RESERVED */                  /* SCI 1 */
#endif
        DUMMY_ENTRY,    /* SCI */
        DUMMY_ENTRY,    /* SPI */
        DUMMY_ENTRY,    /* PAIE */
        DUMMY_ENTRY,    /* PAO */
        DUMMY_ENTRY,    /* TOF */
        fiveHandler,    /* TOC5 */      /* HC12 TC7 */
        fiveHandler,    /* TOC4 */      /* TC6 */
        fiveHandler,    /* TOC3 */      /* TC5 */
        fiveHandler,    /* TOC2 */      /* TC4 */
        DUMMY_ENTRY,    /* TOC1 */      /* TC3 */
        DUMMY_ENTRY,    /* TIC3 */      /* TC2 */
        DUMMY_ENTRY,    /* TIC2 */      /* TC1 */
        DUMMY_ENTRY,    /* TIC1 */      /* TC0 */
        DUMMY_ENTRY,    /* RTI */
        DUMMY_ENTRY,    /* IRQ */
        DUMMY_ENTRY,    /* XIRQ */
        DUMMY_ENTRY,    /* SWI */
        DUMMY_ENTRY,    /* ILLOP */
        DUMMY_ENTRY,    /* COP */
        DUMMY_ENTRY,    /* CLM */
        _start  /* RESET */
        };
#pragma end_abs_address
```

APPENDIX D: FLOW CHART FORC SOFTWARE



APPENDIX E: ROBOT CIRCUIT DIAGRAM



APPENDIX F: ROBOT PICTURES