THE UNIVERSITY OF BRITISH COLUMBIA

Project 2 - Coin Picking Robot

Group B8

Student #	**Student Name**	**% Point**	**Signature**
59822890	Will Chaba	105	WC
46046041	Bryan Nsoh	90	BN
12521589	Isabelle André	110	IA
95040986	Debby Lin	100	DL
75286914	Eric Wu	90	EW
50266873	Daniel Nadeem	105	DN

University of British Columbia
Electrical and Computer Engineering
ELEC 291 L2B Winter 2020
Instructor: Dr. Jesus Calvino-Fraga

Date of submission: April 8, 2020

Table of Contents:

1.0 Introduction

This report outlines the design process, specifications, and challenges with the hardware and software assembly of a coin picking robot. In this project, we have designed a coin picking robot that is programmed to be both automatic and remotely controllable to cover an area enclosed by a specified perimeter of 0.5 square meter to pick up every sort of coin in the canadian currency which are scattered at random before the coin picking process begins. As according to the project specifications[1], we used two different microcontrollers, one for the main coin picking function and another for the remote control. The main coin picking system uses the EFM8 microcontroller, which is powered using four AA batteries to operate the robot's wheels. The main board also uses a 9V battery to control the servo arm movements and electromagnet for picking up coins. To detect coins, the robot uses a metal detector consisting of an inductor driven by a constant current to create a magnetic field, inducing an emf voltage that causes detectable increases in frequency of the current through the inductor as the robot moves over a coin. Once the coin is detected, the robot stops, sends a signal to its two servos controlling the arm, and picks up the coin using an electromagnet. When picking up a coin, the electromagnet is turned on and off using an N-mosfet switch controlled by the EFM8 microcontroller. The coin picking robot's area of operation is restricted to within a perimeter wire enclosing a 0.5 square meter area. Due to the absence of lab equipment such as a function generator in the unforeseen circumstances due to the pandemic, an AC source delivered by a 9V battery powered LM555 acting as an oscillator to generate a square wave passing through the perimeter wire and inducing detectable flux changes near the wire. To detect the perimeter induced emf, two tank circuits consisting of a 0.1uF capacitor and 1mH inductor are attached to the bottom of the robot that leads into a simple peak detector and amplifier circuit[2]. We tuned the perimeter frequency to the resonant frequency of the tank circuit. The presence of two tank circuits covers

[1] Calvino-Fraga, Jesus, "Project 2 Requirements", University of British Columbia, Vancouver, 2020
[2] Calvino-Fraga, Jesus, "Instructions on how to assemble the coin picker using parts from the project 2 kit", University of British Columbia, Vancouver, 2020

all angles of approach to the perimeter wire, allowing the circuit to resonate to the perimeter wire current frequency if approached from a different angle. The coin picking robot features an automatic mode and a manual mode. In automatic mode, all of the preceding features are operated automatically until 20 coins are picked up. After 20 coins are picked up, the robot transfers control to the remote control. The remote control utilizes an STM32F051 microcontroller and a transmitter and receiver to send inputs controlling the robot's movements including all mobility and pick-up motions.

2.0 Investigation

2.1 Idea Generation

Ideas and solutions for individual components of the project were first investigated, evaluated, and optimized appropriately following a problem solving process before being implemented. For instance, we were required to use two different types of microcontrollers for the robot and the remote controller. The microcontrollers offered included the EFM8LB1, STM32F052, the PIC32MX130, the MSP430F2553, and the SAMD20E16. We observed each individual microcontroller's properties to analyse their advantages and disadvantages in this project. There was an absolute consensus on using the EFM8 microcontroller as we were previously familiar with its uses, having done many of our labs with it. Regarding the second microcontroller that would be used for the remote control system, each group member shared their inputs on each option. Microcontrollers were ranked, then eliminated one by one as we compared their functionalities to the project guidelines. Thus, the STM32F051 Microcontroller was chosen to be used in the remote control system due to the abundance of additional resources available on the internet and ease of readability of its datasheet.

As bonus features consist of a large portion of this project's grading, we began brainstorming for additional features to implement in early stages of development. Each team member voiced some of their ideas one by one. The most popular concepts were moving animations on a liquid crystal display (LCD) screen as the coin picking robot picked up coins, and a buzzer speaker playing the mario theme song and sound effects as the robot moved around to pick up coins. The two options were compared and voted upon by the team members. The soundtrack idea was chosen to be implemented first as it would leave more open space to plan the wire management, resulting in more convenience to fit all the electrical components on a single bread board.

2.2 Investigation Design

To decide on a final design, the ideas generated needed to be put into test. We designed a comprehensive investigation of using rough sketches of each design outlining their requirements and benefits and weighted their characteristics for analysis. During the investigation process, we utilized rough codes given to us by Jesus Calvino Fraga and datasheets to understand which microcontroller would best fit our requirement. The EFM8LB1 was especially beneficial for having makefiles ready for usage through our resources. For that reason, we gave EFM8LB1 microcontroller bonus points in our investigation of design. In addition, the EFM8 family datasheet was familiar to all team members as we have done several labs already on this microcontroller. Therefore the EFM8LB1 gains another point in our investigation. As for our remote control system, we also considered the ease of access of each datasheet and compatibility of our coding habits. Although we have no preference for which architecture the microcontroller uses, the ease of use of the datasheet was very crucial to us. We looked over and ranked each datasheet in order and decided that the STM32F051 was the best in terms of readability according to what we are used to, giving it a score in our investigation. Furthermore, a member of our team found a number of websites with an abundance of resources we can utilize for this microcontroller. This gave us the encouragement to try to work with this microcontroller. On another note, we begin working on our special feature after the hardware is completed, while the original special feature designs were nominated right from the beginning of the project, we did not foresee that we are going to run out of space on the breadboard for the LCD screen setup. We debated whether we could move a few things around for the screen, as well as investigated the hardware set up for the magnetic buzzer CEM-1203, which could be used as a speaker if we drive it at different frequencies. As a result, we found that the speaker was far more spatially conservative, as well as easy to implement due to the availability of the music makefile for EFM8 microcontroller. Thus, we decided that we will use the CEM-1203 magnetic buzzer as a speaker for our special feature.

2.3 Data Collection

Due to the nature of our design being purely qualitative, we did not have any quantitative analysis to collect data for. The only recorded data were our preference points towards a certain microcontroller or special feature design. For the following categories we ask all of the team members to give a maximum of 4 points to each microcontroller: familiarity with the datasheet, ease of use of the datasheet, difficulty of hardware set up, and how fitting we think the microcontroller is for the main robot or remote. The scores are then tallied up and compared to other microcontrollers, with the EFM8LB1 and STM32F051 coming on top. On another note, we have also conducted a group vote for which special feature we prefer between the LCD display and the piezo speaker tune. Everyone in our group was only allowed to vote for one of the two designs, we also had to individually consider each aspect: difficulty in construction of the hardware, the coding, and how much resources are available. We have planned to have a debate if there was a tie in our votes for each option, but there was an overwhelming preference for the speaker within our team so there was no need to have that debate.

2.4 Data Synthesis

As previously stated, there was no need to collect any quantitative data for deciding which design we are ultimately going to implement into our coin picking robot project. We synthesized the data using our personal judgement collectively as a team of six people each voicing their input and researching the qualities of each microcontrollers in terms of how familiar with the datasheet we can train ourselves to be in the shortest time and how easily accessible the datasheets are along with the difficulty of hardware set up and how appropriate the microcontroller is for this specific project requirements. After, as was previously discussed, we followed up with a team voting session and tallied up the score for comparison and analysis later on. We have done a similar process with our consideration of which special feature to implement.

2.5 Analysis of Results

The result of the investigation step helped shape the final project design. We used our data to compare the EFM8LB1, STM32F052, PIC32MX130, MSP430F2553, and the SAMD20E16 microcontroller and came to the conclusion that it was slightly more preferable to use the EFM8 and STM32F052 for our project. Our team had consensus with how familiar we are with the EFM8 and how easily adapted to the STM32F052 we have become after investigating its datasheet. Furthermore, as our vote suggested and our understanding of each design have made apparent to us that we will save a considerable amount of space using the speaker special feature as opposed to the LCD screen display special feature. Thus, these three parameters were decided unanimously by the team and we began the process of coding and designing the main program to meet our required specifications as well as begin to brainstorm which kind of function and or what type of music we want to have on our speaker special feature.

3.0 Design

3.1 Use of Process

Different aspects of the project were considered individually before their integration into the final design. Ideas and solutions for individual components of the project were first investigated, evaluated, and optimized appropriately following an engineering problem solving process before being implemented.For instance, features to include in the coin picking robot were first listed and brainstormed, while ensuring that we would still have a convenient placement of pins in use on the EFM8 board as well as enough space on a single breadboard to fit all the required electrical components for the prioritized design goals. We looked into the capabilities and restraints of the previously chosen EFM8 and STM32F051 microcontrollers as it pertained to this project. Working with makefiles with the STM32F051 proved to be challenging at first, as the team members were not familiar in using them. Unlike in the EFM8 microcontroller case, the STM32F051 flash loader was not included in the CrossIDE interface used to build and flash programs into the microcontrollers. The flash loader therefore needed to be accessed through the STM32F051 Makefiles. The Makefile documentation and STM32F051 datasheet were consulted, and required a GNU ARM Embedded Toolchain to be downloaded as well as multiple folders to be linked to the path environment variable. The team members split in two groups: one group that would solely work with the EFM8 on the main robot prototype and another that would work on the remote control system with the STM32F051. Therefore, only the second group would be required to use the STM32F051's Makefiles and flash loader, allowing for more efficient time management and splitting of tasks. Both sub teams prototyped, and planned the development and final assembly of the C code through sharing pseudo code. The final testing portion involved performing trial runs of the coin picking process with different canadian coins, as well as ensuring that the manual controls functioned as well as in automatic mode, moving in eight different directions to pick up all canadian coins.

3.2 Need and Constraint Identification

A battery operated coin picking robot was designed and assembled by our team to be controlled automatically and remotely. Two different microcontrollers were required for this project. The first microcontroller chosen was an EFM8, which was previously used. However, the team needed to familiarize themselves with the STM32F051 microcontroller from ARM as Makefiles were required to flash programs into the new microcontroller. In automatic mode, the robot must be able to pick up 20 current canadian coins including 0.05$, 0.10$, 0.25$, 1.00$ and 2.00$ coins. A perimeter wire is needed to enclose the area in which the robot operates to prevent it from going out of bounds. The robot cannot go beyond the perimeter wire, and will change its angle of approach to change direction and stay within the area. After 20 coins are picked up, the robot changes to manual mode, in which the robot can be remote controlled by a team member using the second microcontroller of choice, a transmitter, and a receiver to move the robot and pick up coins. The coin picking process must be done using a metal detector and an electromagnet, turning on and off accordingly. Furthermore, due to the current situation in social distancing due to the COVID-19 pandemic, the project as well as all the requirements above were completed in social isolation, with no access to lab equipment, help from other teams, and help from the professor, which was limited to assistance through email. The team was required to communicate through messaging and video calls, and each member needed to work on their own tasks and components individually. This was a tremendous challenge as each member was isolated from each other, and needed to deliver their part of the project to avoid delaying the rest of the project's completion.

3.3 Problem Specification

The two major problems we encountered were the electromagnetic and the perimeter detector. The electromagnet can not be powered directly from the EFM8 board since the board only provides 3.3V. We tried using MOSFETs to make a switch to control whether we want the power from the battery to go through or not, however it was not working as expected. For the perimeter detector, since we didn't have access to the function generator in the lab, we made a LM555 timer circuit to generate signals. When we checked the signal generated from the circuit, it was very inconsistent and we were not sure what kind of wave we're looking for. The inductors on the car were also not getting a frequency or amplitude change. These two hardware problems made us unable to design the proper code to achieve what we wanted. Furthermore, there seemed to have four faulty connections on the left side of the op-amp used to amplify the signal received by the tank circuit resonance. Several days of debugging lead to this conclusion, as the pins from the chip would connect then disconnect. This problem gradually evolved into the entire left side of the breadboard losing its connections. In addition, on the day of the due date for the project, one servo motor began showing a weaker signal, discovered by its weaker movements and lower range of motion. A few hours later, the second motor began showing the second symptoms. However the problem was later solved by replacing the two servo motors.

3.4 Solution Generation

We considered using an operational amplifier to bring the voltage up from 3.3V to what we needed to power the electromagnet. We also considered using photo coupling via the LTV847 chip in addition to mosfets to get the 6V needed to power the electromagnet. We obtained a compact, portable oscilloscope with which we were able to observe the signal created by the LM555 timer in the perimeter detector. Both our perimeter and coin picking inductors were registering no frequency or amplitude changes. As it was difficult to get assistance due to the current pandemic situation, team members asked for assistance from the professor through email. As the professor explained the functioning of the tank circuit and peak detector's op-amp, significant changes were made to the peak detector. First, as the tank circuit consisted of a 0.1 uF capacitor and 1 mH inductor, the resonance frequency of the circuit was calculated to be approximately 16 kHz. To generate a 16 kHz square wave, two 3 kOhm resistors were added to the LM 555 circuit as shown in Figure 3.4.1

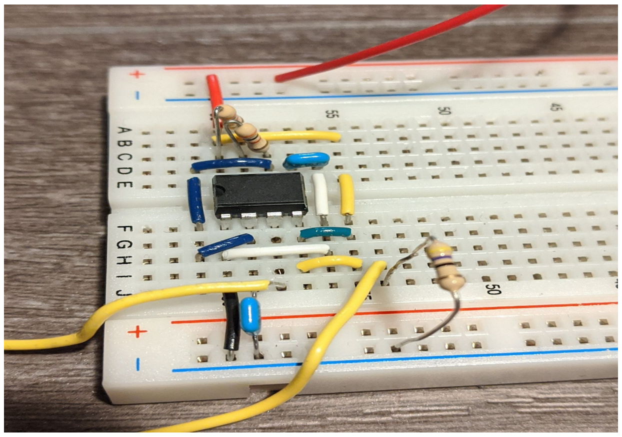


Figure 3.4.1 Oscillator Circuit 16 kHz Square Wave

To detect the wave in the perimeter wire an LM358 op-amp was used with a gain of 101 to amplify the signal, as shown in the oscilloscope output in Appendix A, using a 10 kOhms resistor as the feedback resistor (Rf) and 100 Ohms as the load resistor (Rl). The complete circuit simulation of the peak detector and tank circuit, kindly provided by Professor Calvino can be observed in Appendix C. When measuring the peak voltage after the diode, the oscilloscope showed a significant rise in voltage when passing over the perimeter wire, as shown by the oscilloscope output in Appendix B. When debugging these components, we redid our circuit's wiring for both these components to make sure the error wasn't caused by hardware. We then proceeded to debug the software.

3.5 Solution Evaluation

Using an operational amplifier to get the right voltage worked. However, it took up too much space on the circuit and we weren't able to fit enough components on the same breadboard. We decided to use the photo coupling as the LTV847 chip was already being used to power the wheels and the only extra hardware we needed to install was a mosfet and some wires. This configuration also allowed us to conveniently switch the electromagnet on or off using the microprocessor. The errors we got with the perimeter detector were caused by a faulty breadboard and it began working once we moved the same hardware and wiring to another empty portion of the same breadboard. However, as time passed, all rails on the left side of the breadboard gradually lost connection to the wiring, which was discovered after making multiple continuity tests. Therefore, a complete change of breadboard was required. This was a lengthy and anxious process that took several hours, and was completed after drawing a complete map of the circuit board to prevent any potential errors and misplacement.

3.6 Safety and Professionalism

To avoid any safety issues, proper lab safety practice was used throughout the project timeline, specifically when soldering and using the oven. Hair was tied neatly, closed shoes were worn, and lab benches were cleaned and wiped neatly after each use. When soldering in the lab, the team ensured to wear safety goggles and made others aware when soldering. To prevent fire hazards, soldering iron and other lab equipment were turned off promptly after each use. Due to the current circumstances regarding COVID-19, the team no longer had access to lab equipment and had to improvise quickly at home while in social distancing. To limit social interactions, the team split tasks and worked individually or in pairs on different components. The majority of the work was done through screen sharing, video calling, and messaging. This approach proved to be most practical, yet was still very challenging as the team had limited lab equipment and could only help each other remotely in the instance of a bug.

3.7 Detailed Design

Hardware Block Diagram: Transmitter

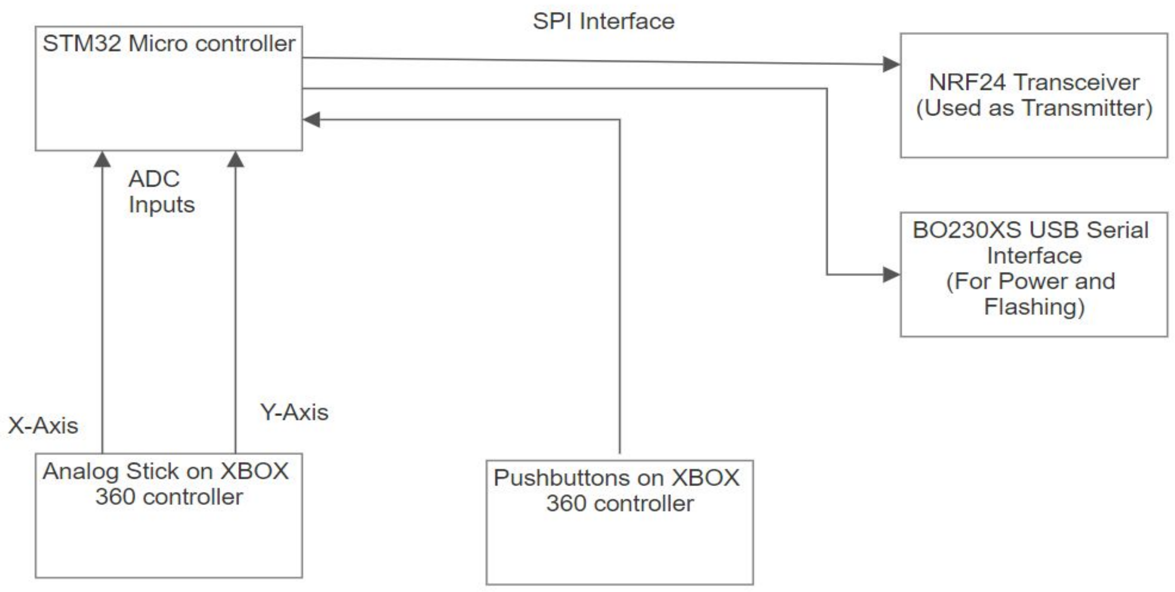

SPI Interface
STM32 Micro controller
NRF24 Transceiver (Used as Transmitter)
ADC Inputs
BO230XS USB Serial Interface (For Power and Flashing)
X-Axis
Y-Axis
Analog Stick on XBOX 360 controller
Pushbuttons on XBOX 360 controller

Figure 3.7.1 Transmitter Hardware Outline

Circuit Explained: Transmitter

The hardware for the transmitter is fairly simple, it is controlled with a STM32 microcontroller, with a BO230XS USB serial interface to connect to a computer for flashing and for power. We also used a NRF24 Transceiver to communicate with the EFM8 onboard our robot. To collect user inputs, we connected a heavily modified Xbox 360 controller to the board (See photo 1 of Appendix E). After attempting to reuse the existing buttons on the controller, we eventually managed to mound the provided pushbuttons from our kit into the controller underneath the ABXY buttons on the controller, so when pressed they make contact and depress the push buttons underneath. Essentially, we were using the existing circuit board in the controller as a physical support to hold our components in place, not as a circuit in any way. For directional control, we decided to use an analog stick from the controller. We left the stick mounted in the controller, but soldered wires onto the pins on the back of it, using a wiring pinout for a Xbox 360/Playstation 2 analog stick adapter found on Adafruit's Website[3] (See photos 2 and 3 of appendix E). Using the onboard ADC on the STM32, we were able to get input from four facebuttons (could have done more though this was more than enough) and one analog stick, and send it over to the robot using the NRF24 transceiver.

[3] Adafruit,"Analog 2-axis Thumb Joystick with Select Button + Breakout Board", Photo, 2016.
https://www.adafruit.com/product/512#description
(See Appendix F)

Hardware Block Diagram: Robot

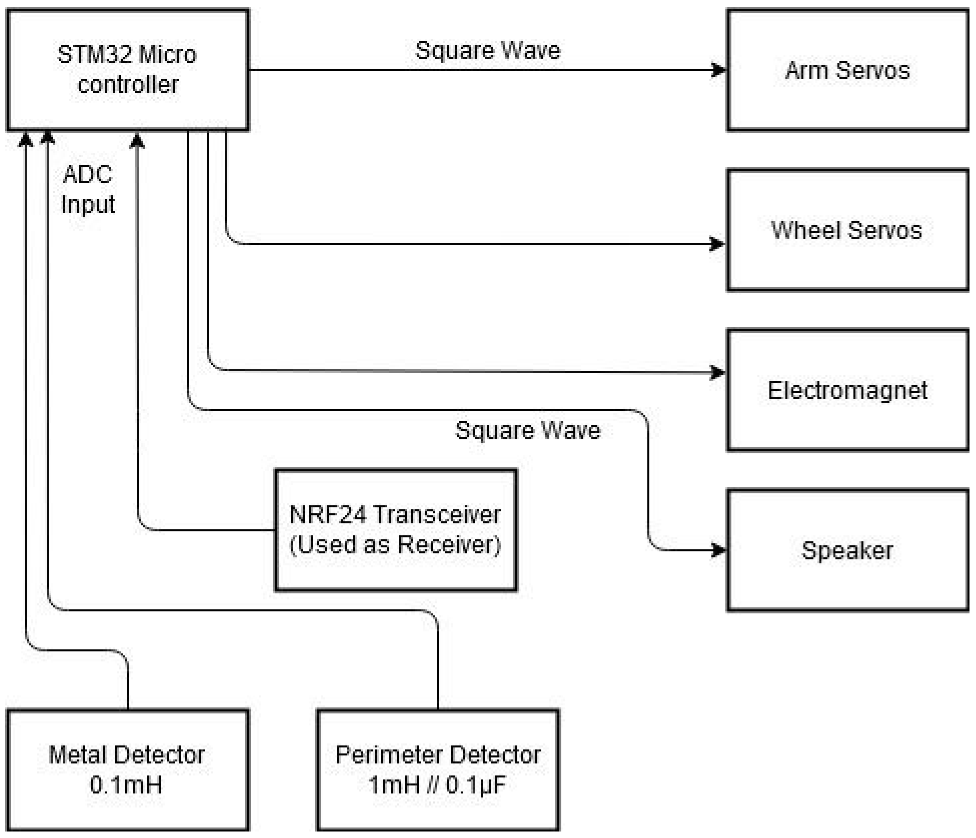


Figure 3.7.2 Main Functionality Hardware Design Outline

Circuit Explained: Robot

The robot consists of multiple components that are receiving and providing signals to the EFM8 microprocessor. The wheels that provide the robot with motion are controlled through the use of voltage differences, allowing forwards and backwards traversal, as well as rotation and even directional steering. Through the use of photo coupling from the LTV847 chip, we were able to provide the 6 volts necessary to power the wheels, while controlling them with the 5 volt microprocessor. The servos that allowed for control of the electromagnet arm were controlled in a different manner, as they were controlled through direct positioning through the use of a variable square wave. This process used NFETs to increase the voltage that we could supply to the arm servos which allowed for strong and fast arm motions. The electromagnet was controlled in a similar manner to that of the wheels; a direct voltage was applied to a photo couple to either allow the 6 volt difference across the magnet, or to block it. The NR24 receiver was wired to the EMF8 microcontroller and pin P3.0 was connected to ground to set the EMF8 up as a receiver. Once we set the EMF8 to manual mode, it would receive instructions from the controller via radio waves generated using another NRF24 chip and the STM32 microprocessor in transmitter mode. Each received radio signal specified an instruction or series of movements for the EMF8 to execute. The metal detector shown in Figure 3.7.3 was built through an inductor which was supplied with an alternating current signal.

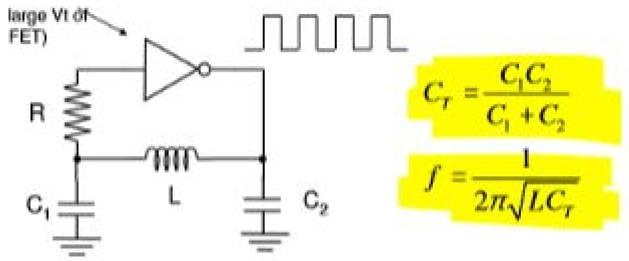

large Vt of FET)
R
L
C_1
C_2
$C_T = \frac{C_1 C_2}{C_1 + C_2}$
$f = \frac{1}{2\pi\sqrt{LC_T}}$

Figure 3.7.3 Basic Metal Detector: Colpitts Oscillator[4]

As the robot passed over a large enough metal object (i.e. a coin) the frequency of the current signal changed significantly. By detecting this change, we were able to determine when a coin was underneath the robot. The perimeter detector involved a similar operation to the metal detector, however it was opposite. The inductor on the robot was not supplied with a signal, the wire that served as the perimeter was and the inductor on the robot experienced an induced current due to passing over the perimeter wire. By reading this substantial reading, we could determine when the robot passed over the perimeter wire. As an additional feedback measure, a speaker was added to provide audible notification of passing collecting a coin, and various other situations.

[4] Calvino-Fraga, Jesus, "Project 2 Lecture Slides", University of British Columbia, Vancouver, 2020

Software Block Diagram: Transmitter

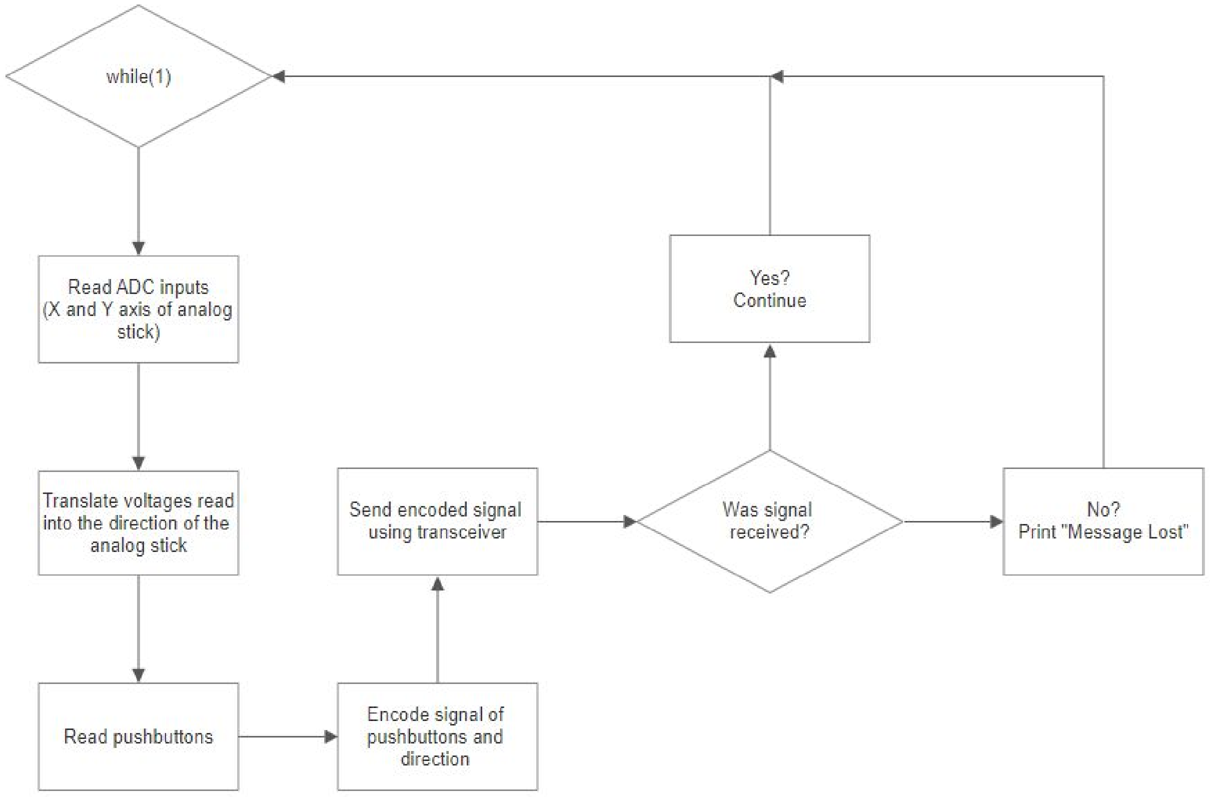

while(1)
Read ADC inputs (X and Y axis of analog stick)
Translate voltages read into the direction of the analog stick
Read pushbuttons
Encode signal of pushbuttons and direction
Send encoded signal using transceiver
Was signal received?
Yes? Continue
No? Print "Message Lost"

Figure 3.7.3 Transmitter Software Overlook Flowchart

Software Explained: Transmitter

The software for the transmitter checks for user input and sends it wirelessly to the EFM8 onboard our robot. It accomplishes this by first reading the ADC to find the analog stick's position. It translates the voltage it reads into a number from 1-9, in which the desired direction for the robot will be the number's position on a dialpad. Next we check all four of our pushbuttons, and encode the two inputs into a 5 digit string, with one digit for each button, and one digit for the directional input of the analog stick. This signal is then sent using the NRF24 wireless transceiver to the EFM8, which upon successful receival of the transmission, will return a verification that the message was sent correctly. If our controller/transmitter does not receive this confirmation, it will print an error message to the screen. Regardless of error status, the code will loop back to the top and send a new set of inputs to the robot. Due to the quick refresh rate of sending new inputs, one lost input is not a serious setback, and thus no major action is taken beyond informing the user that there may be a problem. This cycle of collecting inputs and sending them will continue indefinitely.

Software Block Diagram: Robot

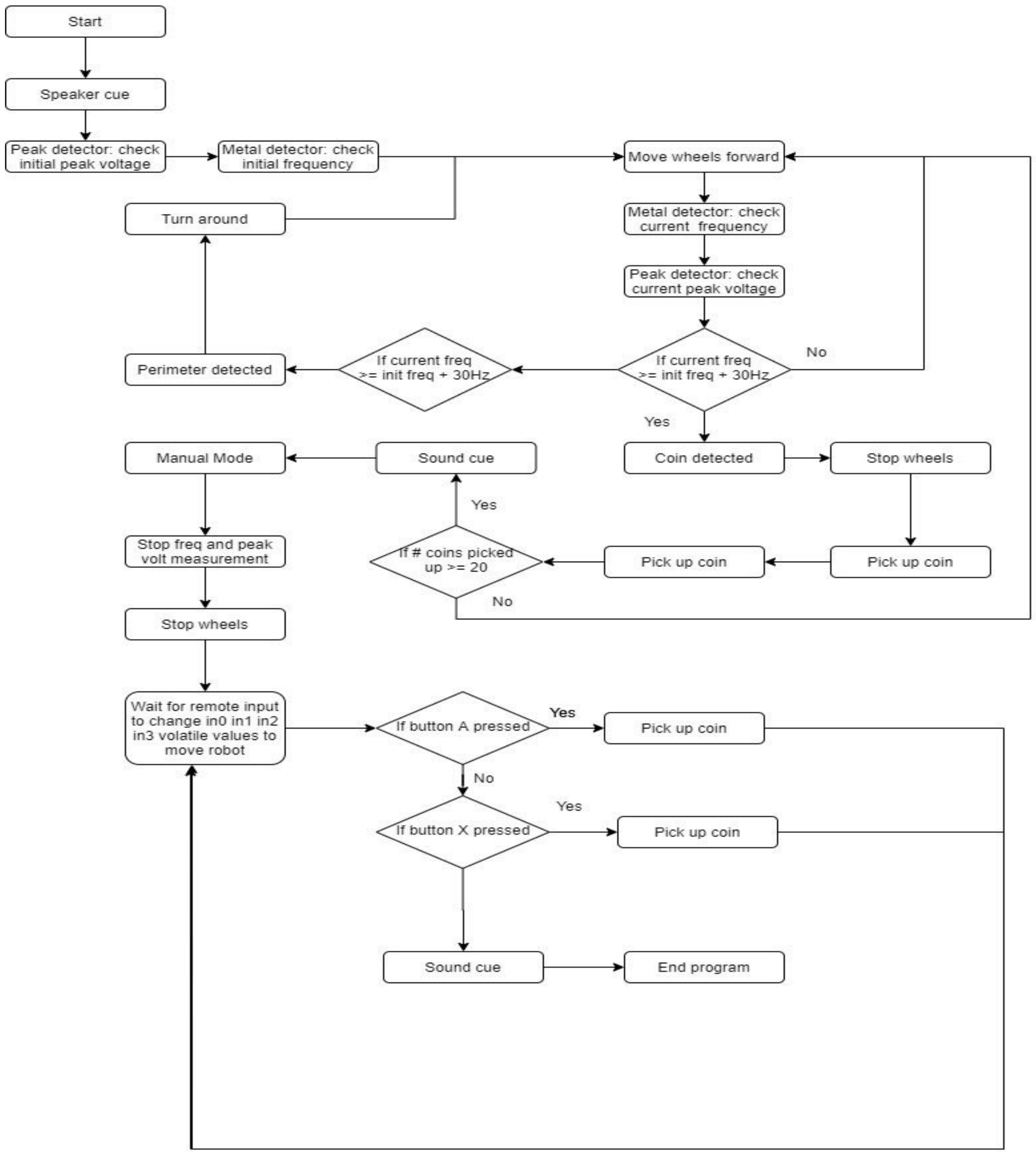

Start
Speaker cue
Peak detector: check initial peak voltage
Metal detector: check initial frequency
Move wheels forward
Turn around
Metal detector: check current frequency
Peak detector: check current peak voltage
Perimeter detected
If current freq >= init freq + 30Hz
If current freq >= init freq + 30Hz
No
Yes
Manual Mode
Sound cue
Coin detected
Stop wheels
Yes
Stop freq and peak volt measurement
If # coins picked up >= 20
Pick up coin
Pick up coin
No
Stop wheels
Wait for remote input to change in0 in1 in2 in3 volatile values to move robot
If button A pressed
Yes
Pick up coin
No
Yes
If button X pressed
Pick up coin
Sound cue
End program

Figure 3.7.4 Main Functionality Software Outlook Flowchart

Software Explained: Robot

After flashing the program to the board, a short tune plays to indicate the start of the program. The microcontroller then measures an initial frequency using the metal detector and an initial voltage from the peak detector as thresholds to compare the future measurements. This ensures a better and more accurate comparison of the measured frequency and voltage values as depending on the current environment, the initial frequency and voltage may be of different magnitudes than before. The servo arm is reset to its initial position centered at 1.2 (lower servo) and 1.2 (upper servo). The volatile variables (in0, in1, in2, in3) controlling the wheel motors are changed to 80,20,80,20, indicating that the robot is now moving forward until a coin or the perimeter is detected. The code then enters the main while loop, where the frequency of the metal detector and peak voltage are continuously measured to detect coins or the wire perimeter. If a coin is detected by comparing the initial frequency value with a raise of 20 Hz to the current measured frequency, the servo arm picks up the coin, controlling its two servo motors by inputting different pulse widths through a process named pulse width modulation (PWM). The servo arm controls and coin pick up routine are as described in Figure 3.7.5.

```
void arm_pick_up(void) {                  //picks up coins
       PWMMAG = 0;
       waitms(500);
       arm_flag = 1;
       pwm_reload1=0x10000L-(SYSCLK*2.3*1.0e-3)/12.0;       //down
       PWMMAG = 1;                                      //electromagnet on
       waitms(500);
       arm_flag = 0;
       pwm_reload0=0x10000L-(SYSCLK*2.4*1.0e-3)/12.0;       //sweep left
       waitms(500);
       arm_flag = 1;
       pwm_reload1=0x10000L-(SYSCLK*0.6*1.0e-3)/12.0;       //pick up
       waitms(500);
       arm_flag = 0;
       pwm_reload0=0x10000L-(SYSCLK*0.7*1.0e-3)/12.0;       //carry right
       waitms(500);
       arm_flag = 1;
       pwm_reload1=0x10000L-(SYSCLK*1.0*1.0e-3)/12.0;       //drop
       PWMMAG = 0;
//Electromagnet off
       //ParseMDL(cointune);
       waitms(500);
       arm_flag = 0;
       pwm_reload0=0x10000L-(SYSCLK*1.2*1.1e-3)/12.0;       //centered
       waitms(500);
}

void arm_reset(void) {            //resets and centers arm
       PWMMAG = 0;
       //Electromagnet off
       arm_flag = 1;
       pwm_reload1=0x10000L-(SYSCLK*1.2*1.0e-3)/12.0;        //up
       waitms(500);
       arm_flag = 0;
       pwm_reload0=0x10000L-(SYSCLK*1.2*1.0e-3)/12.0;        //centered
       waitms(500);
}
```

Figure 3.7.5 Main Control Source Code

As the coin is dropped into the bucket, another sound cue is enabled using a sound flag to indicate the use of timer 2 for sound rather than the robot's wheels. After the coin picking process, a coin count is incremented and the robot continues to move forward until the perimeter is reached or another coin is detected. When the perimeter is reached, the peak detector detects a significant rise in voltage due to the square wave emitted by the function generator, in our case, simulated by a timer 555 circuit. The measured peak voltage is compared to its initial value, and either turns right on the spot rapidly for 1.5 second if the rise is significant enough, or continues on its path as before. The code then checks for the number of coins acquired so far, comparing the incremented coin count variable to twenty. If the count exceeds, a sound cue is enabled, and the robot stops its measurements and wheels as the automatic mode switches to remote control mode. In remote control mode, the remote buttons and analog stick sends signals to the transmitter, which are received by the receiver, and in turn changes the volatile variables to control the wheels and trigger the arm movements. When button A is pressed, the servo arm begins the coin pick up routine before resetting to its initial position, and when button B is pressed, the robot plays a final longer tune from Mario, ending the program.

3.8 Solution Assessment

Servo Motor Controls Testing

The servo motor circuitry required extensive testing to develop a working mosfet switch that allowed for high voltage and accurate motion. The NFETs were wired as follows: 6 volts connected to source, microcontroller signal connected to gate and signal wire of motor, drain connected to power source of motor. Although we are supplying the square wave signal to both the power and signal wires of the motors, we believe that this provides a more precise positioning of the arm. After assembling the two servo motors onto the arm, the motor ranges and directions needed to be analysed before creating the arm pick up choreography. The directions and full ranges were found as follows in Table 2.8.1.

	Up	**Down**	**Left**	**Right**
Top Servo	0.6	2.4	X	X
Bottom Servo	X	X	2.4	0.6

Table 3.8.1 Arm Servo Motor Ranges and Directions

The neutral position for the servo arm to return to after each pick up was decided to be at 1.2 (top servo) and 1.2 (bottom servo). The arm pick up choreography was then created by testing different ranges for the appropriate arm position required to pick up a coin. The servo pulse width (positions) ranges and movement description are shown in Table 2.8.2.

Servo Motor	**Pulse Width**	**Movement Description**
Top	2.3	Upper arm down
Bottom	2.4	Sweep left for coins
Top	0.6	Coin pick up
Bottom	0.9	Carry right
Top	1.2	Drop bucket
Bottom	1.2	Center arm

Table 3.8.2 Servo Arm Pickup Process Ranges and Controls

Analog Stick Voltage Testing

Instead of using a wii nunchuck as was demonstrated in class, we decided to go the route of repurposing an Xbox 360 controller with a broken circuit board. This ended up not being too difficult, as the analog sticks used on the controller are quite common, and sites like Adafruit even have off the shelf circuit boards you can buy to use these sticks with an arduino. For our uses however, we wanted to keep it in the controller, so we soldered wires onto the pins of the analog stick but left it inside the old circuit board to hold it in place. Using the provided "PrintADC2Inputs" makefile for the STM32, we were able to test the voltage ranges of the X and Y axis of the analog stick. The two axes acted like potentiometers, and acted independently of each other. They both have a center voltage, and would reach a maximum if pushed all the way to one side, or zero if pushed in the opposite direction. Our testing, as shown in Table 3.8.3 found that with the analog stick connected to the 3.3V source rails for the STM32, the center point for the X axis (left-right) was 1.62V, with a left extreme of 2.75V, and a right extreme of 0V.

Direction:	Xaxis:	Yaxis:
Center	1.62	0.99
Right	0	0.99
Left	2.75	0.99
Up	1.62	1.95
Down	1.62	0
Up-Right	0	0.99
Up-Left	2.75	0.99
Down-Right	0	0
Down-Left	2.75	0

Table 3.8.3 Analog Stick Testing Data

Likewise the Y axis has a center point of 0.99V, with an upward extreme of 1.95V, and a downwards extreme of 0V. Using this information, one can easily translate the voltages into a direction in software, which gives us a directional input for our machine.

4.0 Lifelong Learning

This project has been very valuable as a learning opportunity for everyone on the team. Everyone benefited from different aspects throughout the process. For instance, we learnt a lot from each other. Many were new to Github and had to learn its interface in order to communicate our ideas back and forth and debug each other's code in an orderly manner. Those with a little more experience with Github were able to help us navigate our way through the system and optimize our communication efficiency using the tool. Likewise, we were able to apply material learned in prior courses. Most have had a background from the course CPEN 211 and CPSC 259 last semester, which was beneficial for understanding the ins and outs of microcontroller features and C language, and for quickly being able to find the root of compilation error messages. This course was also helpful in analysing and understanding the raw code provided by Dr. Calvino-Fraga in class[5] as well as integrating our own solutions into the provided code.

Moreover, some of us had to teach ourselves how to quickly navigate through large chunks of code in our source file or through a clutter of text in the data sheets to find what we are looking for. In fact, there were quite a few instances of self learning during our creative process. Those of us who are not as familiar with Makefiles and environment variables for the set up of different microcontrollers utilized online discussion boards to see real life examples of some of the instructions being used, we then inferred from those examples ways we could use the instruction to reconstruct the desired functionality.

Ultimately, these experiences throughout this project will have lasting effects on our work efficiency in the future. Be it another group project or not, we have learnt tools to use for better communication between our teammates, we have applied classroom knowledge directly to our project, and have learnt how to deliver efficient and quality work in a timely manner by utilizing internet resources along with our data organization skills.

[5] Calvino-Fraga, Jesus, "Project 2 Lecture Slides", University of British Columbia, Vancouver, 2020

5.0 Conclusion

This report has discussed the development of an automated coin picking robot with audio output from a magnetic buzzer, the robot is also manually controllable via a remote controller with a NRF24 wireless transceiver. The objectives of this lab were to assemble the necessary hardware and write the necessary software to drive a robot that automatically detects an external electromagnetic parameter and clear a path within the parameter to pick up any coin detected using a metal detector. This robot gets its mobility ability to pick up coins from servo motors and electromagnet that are controlled using an EFM8 microcontroller and external batteries.

Different aspects of the project were considered individually before their integration into the final design. Each feature and individual components of the project were first investigated, evaluated, and optimized appropriately following an engineering problem solving process before being implemented. Some of these features include our remote control and speaker audio output. We successfully implemented these features and made our robot more interesting with a whole new depth of possible applications.

This project encouraged utilization of assembly and C coding skills with microcontrollers that we have practiced with in prior labs with the practical as well as theoretical knowledge about operational amplifiers, MOSFETs, inductors, capacitors, speakers, diodes, pwm and transistors that were taught in other electrical engineering classes. This project went on for several weeks. Approximately 70 hours of work were done on this project, most of which were spent debugging hardware and overcoming our physical distancing dilemma with one of our teammates volunteering to drive his own vehicle to individual team members to deliver necessary parts for debugging the assigned feature each member was responsible for.

6.0 References

Adafruit,”Analog 2-axis Thumb Joystick with Select Button + Breakout Board”, Photo, 2016. https://www.adafruit.com/product/512#description

Calvino-Fraga, Jesus, “Project 2 Requirements”, University of British Columbia, Vancouver, 2020

Calvino-Fraga, Jesus, “Project 2 Lecture Slides”, University of British Columbia, Vancouver, 2020

Calvino-Fraga, Jesus, “Instructions on how to assemble the coin picker using parts from the project 2 kit”, University of British Columbia, Vancouver, 2020

Calvino-Fraga, Jesus, “Instructions on how to assemble the robot using parts from the project 2 kit”, University of British Columbia, Vancouver, 2020

7.0 Bibliography

On Semiconductor, “NTD3055L104,NTDV3055L104”, Mosfet 3055L104 datasheet, 2016.

On Semiconductor, “NTD2955, NVD2955”, Mosfet 2955 datasheet, 2016.

Texas Instruments, “LM555 Timer”, LM555 datasheet, 2015.

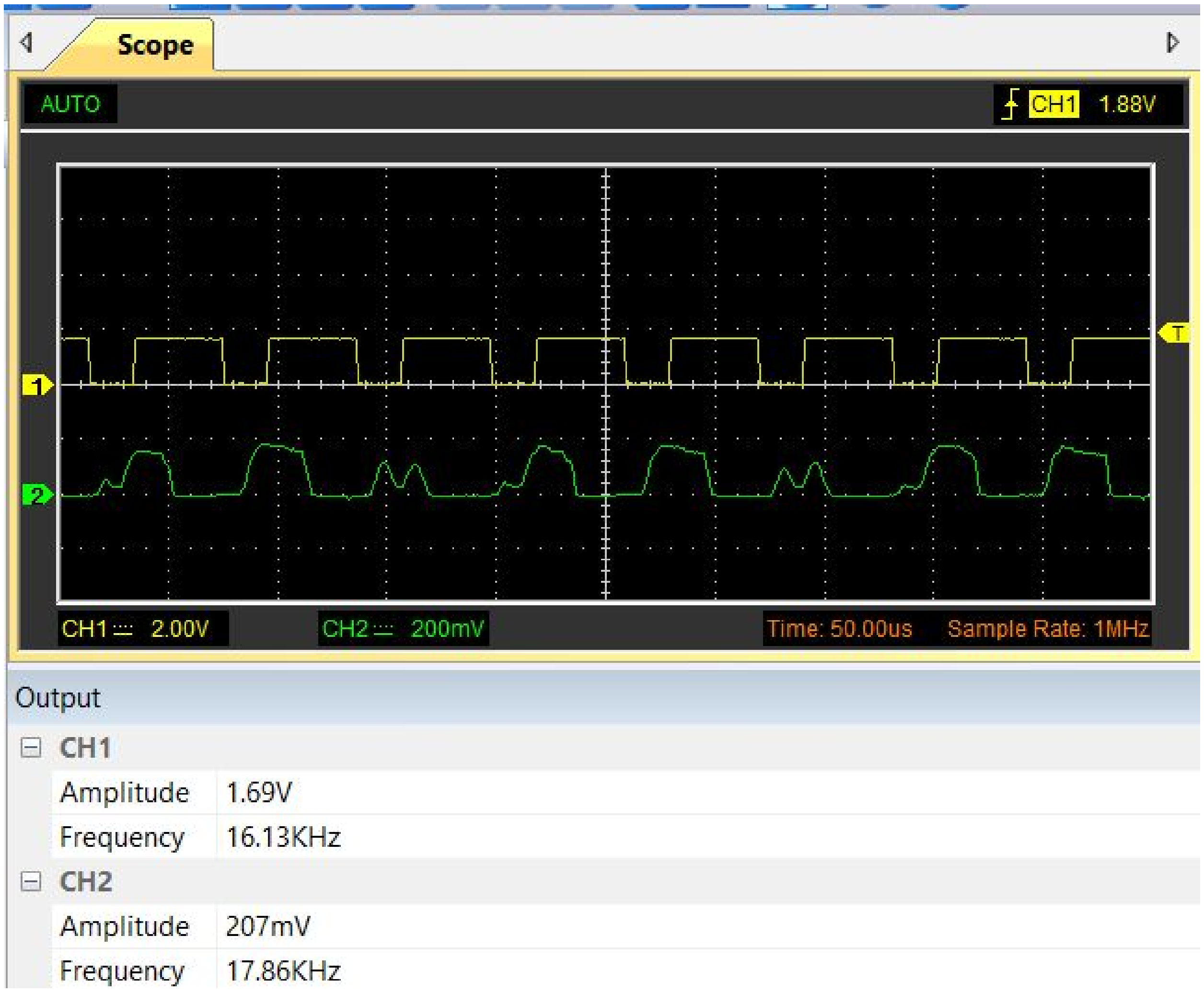
Silicon Labs, “EFM8 Laser Bee Family”, EFM8LB1 Reference Manual datasheet, 2018

Electronic Oscaldas, “MG90S”, MG90S Metal Gear Servo datasheet, 2018.

CUI Devices, “CEM-1203”, Magnetic Buzzer Transducer specification, 2019.

8.0 Appendices

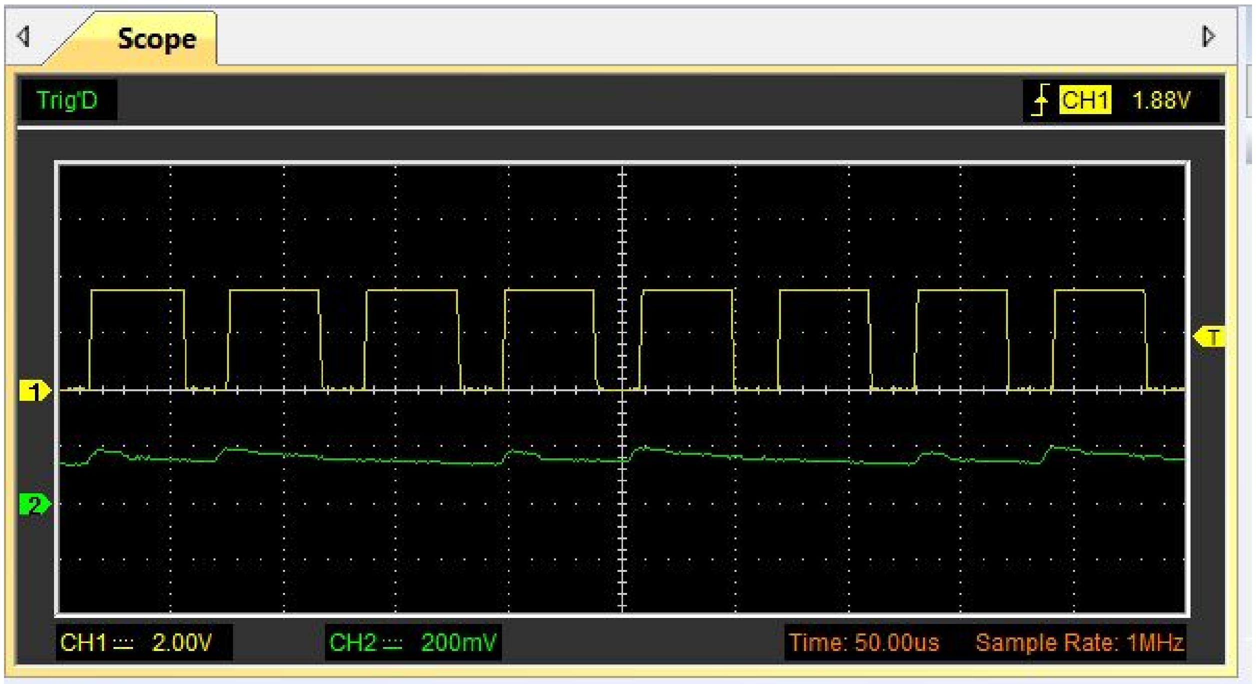
Appendix A: Perimeter Wire Op-Amp Output



Output

CH1	
Amplitude	1.69V
Frequency	16.13KHz
CH2	
Amplitude	207mV
Frequency	17.86KHz

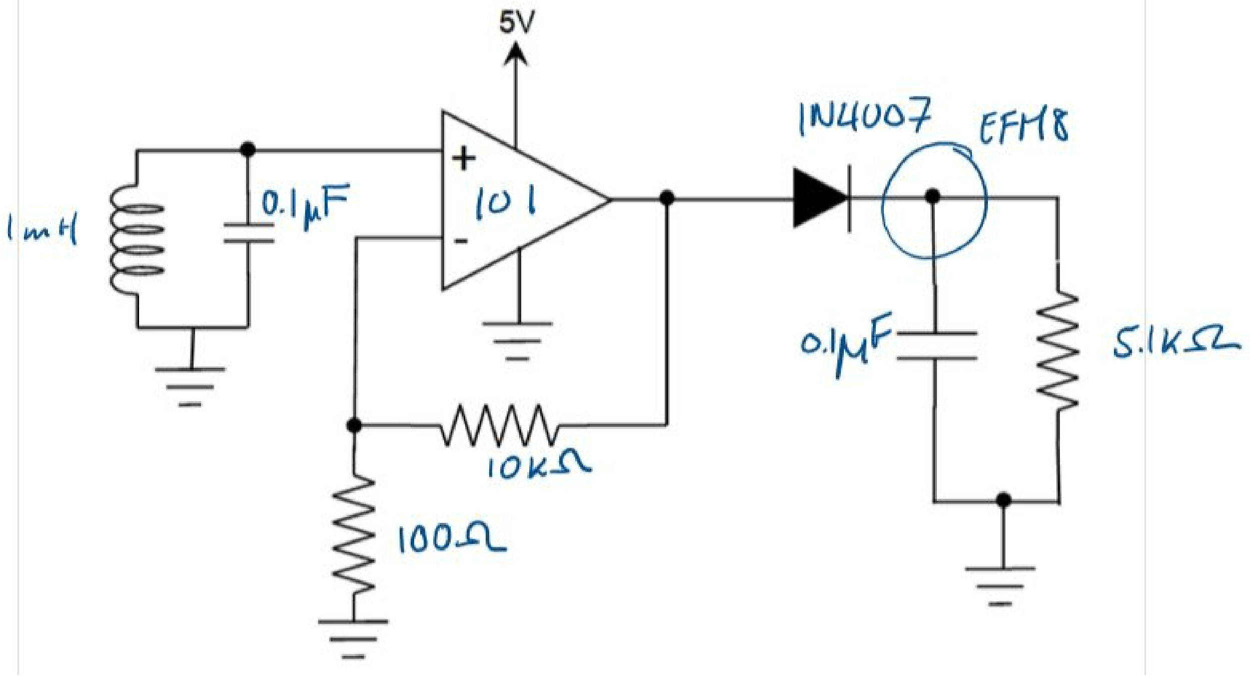
Appendix B: Peak Detector Diode Output



Output

CH1	
Amplitude	3.51V
Frequency	16.67KHz
CH2	
Amplitude	87.8mV
Frequency	51.44Hz

Appendix C: Peak Detector Circuit

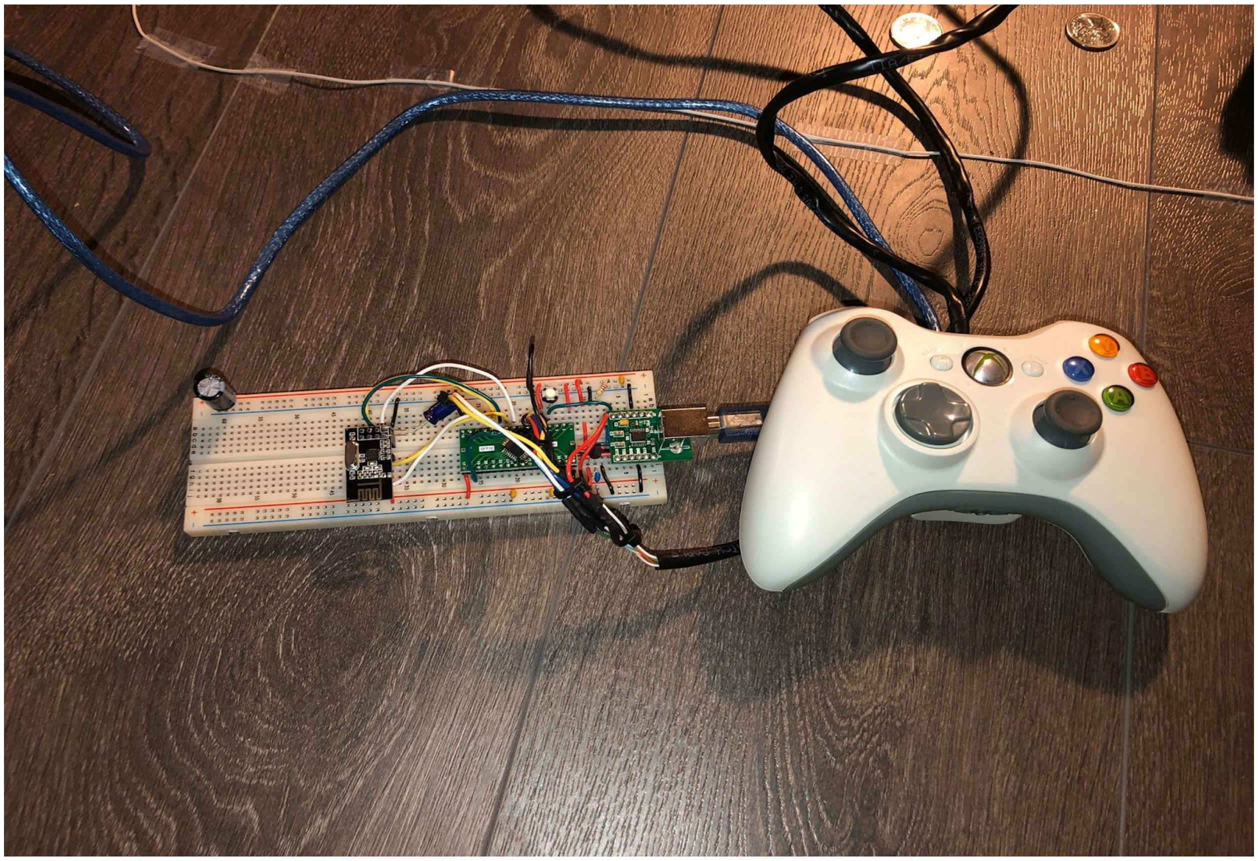


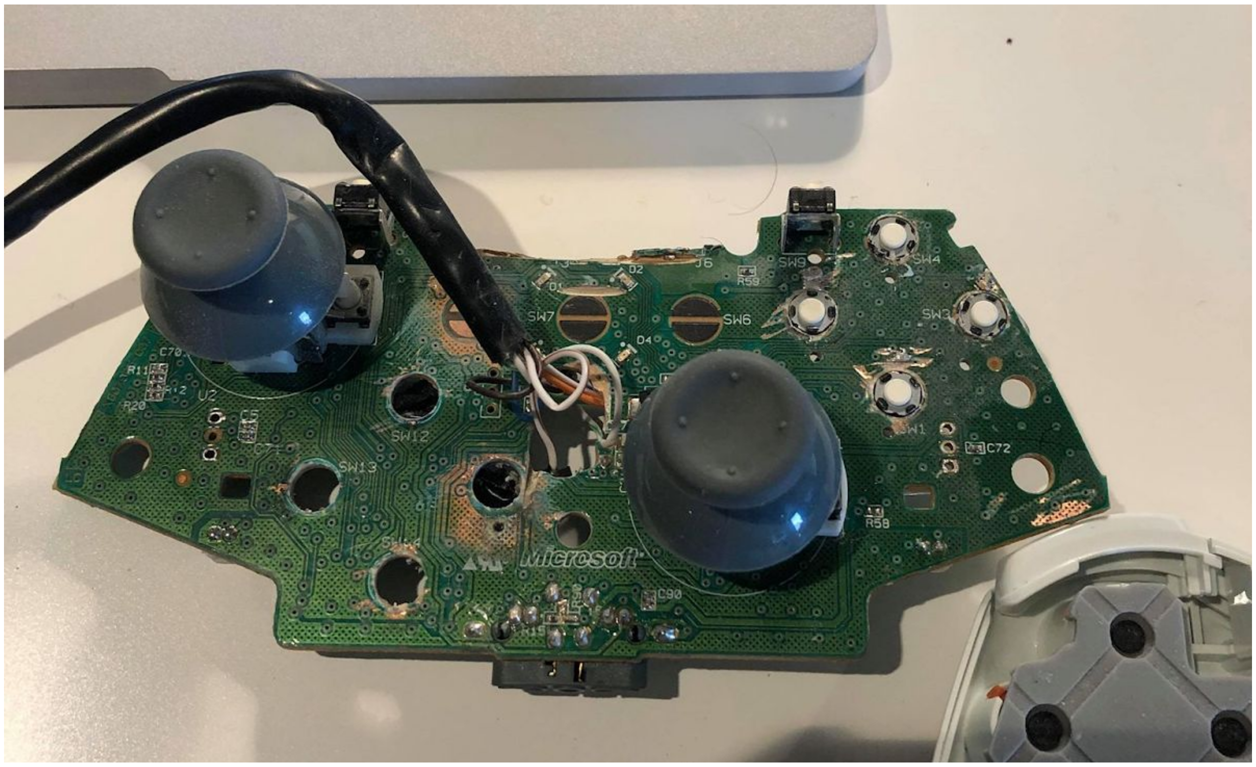
Provided by Professor Jesus Calvino-Fraga, ELEC 291

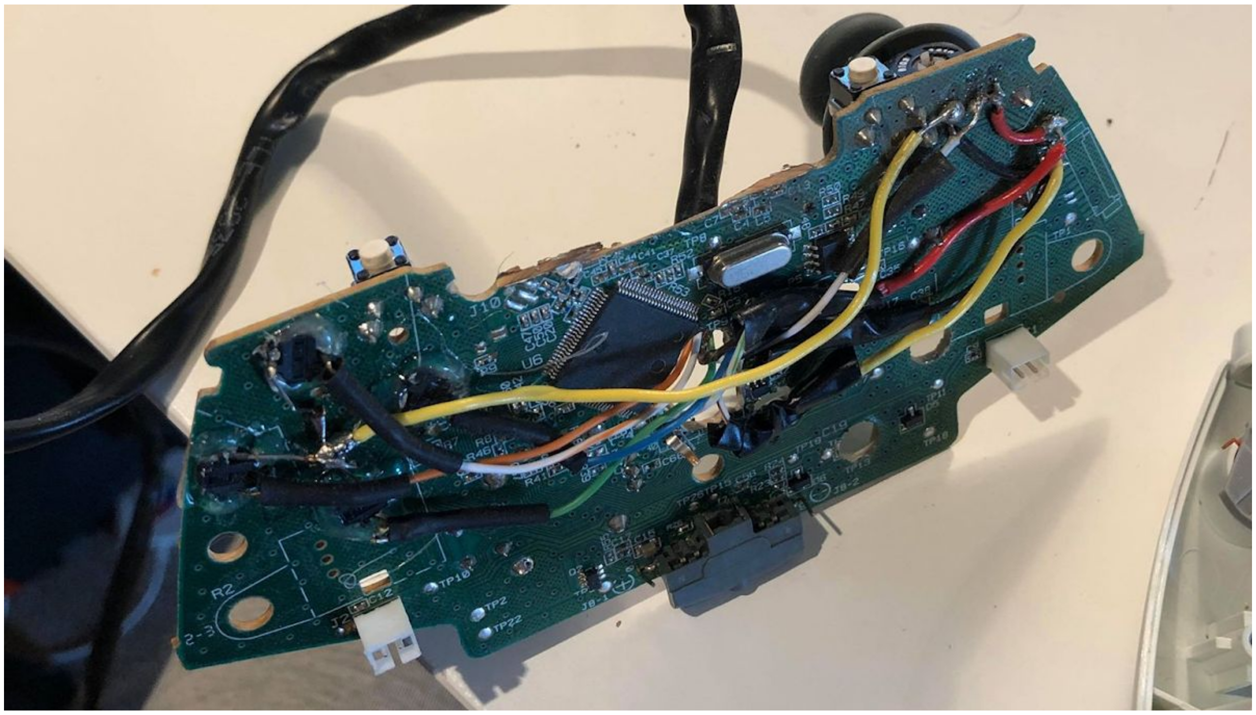
Appendix D: Analog Stick Testing Data

Direction:	Xaxis:	Yaxis:
Center	1.62	0.99
Right	0	0.99
Left	2.75	0.99
Up	1.62	1.95
Down	1.62	0
Up-Right	0	0.99
Up-Left	2.75	0.99
Down-Right	0	0
Down-Left	2.75	0

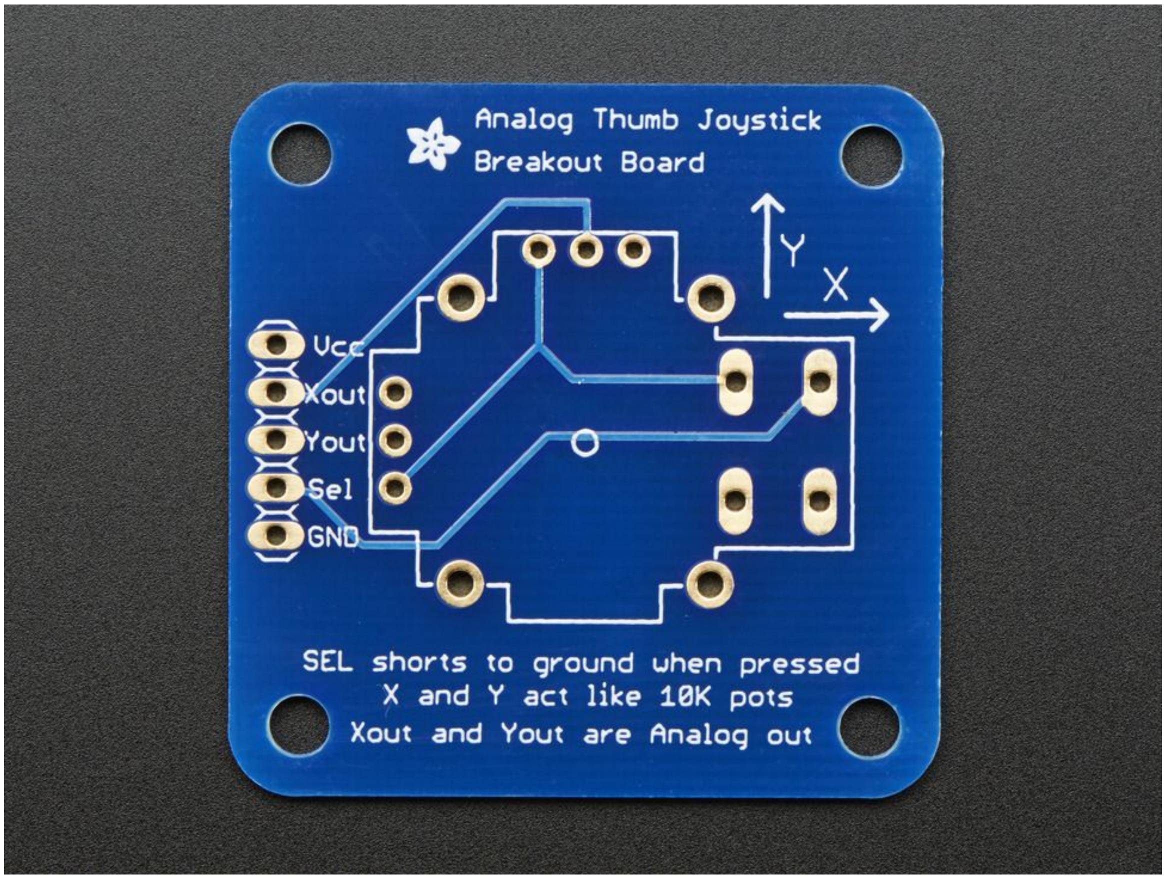
Appendix E: Transmitter Hardware







Appendix F: Adafruit Diagram


Analog Thumb Joystick
Breakout Board
Y
X
Vcc
Xout
Yout
Sel
GND
SEL shorts to ground when pressed
X and Y act like 10K pots
Xout and Yout are Analog out

Appendix G: Main Code Robot

```
//  square.c: Uses timer 2 interrupt to generate a square wave in pin
//  P2.0 and a 75% duty cycle wave in pin P2.1
//  Copyright (c) 2010-2018 Jesus Calvino-Fraga
//  ~C51~

#include <stdlib.h>
#include <stdio.h>
#include <EFM8LB1.h>
#include <ctype.h>
#include <string.h>
#include "Tunes.h"
#include "nrf24.h"

#define     PAUSE 0
#define     BASE_C1
#define     BASE_Cs     2
#define     BASE_D3
#define     BASE_Ds     4
#define     BASE_E5
#define     BASE_F6
#define     BASE_Fs     7
#define     BASE_G8
#define     BASE_Gs     9
#define     BASE_A10
#define     BASE_As     11
#define     BASE_B12

#define     NORMAL      7
#define     LEGATO      8
#define     STACCATO    3

#define SYSCLK 72000000L
#define BAUDRATE 115200L
#define RELOAD_10MS (0x10000L-(SYSCLK/(12L*100L)+1))
#define F_SCK_MAX 2000000L  // Max SCK freq (Hz)

#define FORWARD   8
#define BACKWARD 2
#define LEFT 4
#define RIGHT 6
#define FORWARD_RIGHT 9
#define FORWARD_LEFT 7
#define BACKWARD_RIGHT 3
#define BACKWARD_LEFT 1
#define NO_MOVEMENT 5
```

```
#define EQ(A,B) !strcmp((A),(B))

void ParseMDL(char * music);

#define BUFFSIZE 15
xdata char buff[BUFFSIZE+1];

xdata unsigned char style, octave, note, tempo;
xdata int actLen, defLen, cur;
extern volatile int timer_count;

// timer 0 used for systemclock
#define TIMER0_RELOAD_VALUE (65536L-((SYSCLK/12L)/1000L))     ////!!!!!!!!!!!!

#define SOUNDPIN P2_6

volatile int timer_count;

extern const float FTone[];

#define OUT0 P2_4        //motor 1
#define OUT1 P2_3

#define OUT2 P2_2        //motor 2
#define OUT3 P2_1

#define PWMOUT0 P2_0           //bottom motor  //P2_0
#define PWMOUT1 P1_7           //top motor

#define PWMMAG P3_0                   //electromagnet

#define VDD 3.3035 // The measured value of VDD in volts

volatile unsigned char pwm_count0=0;
volatile unsigned char pwm_count1=0;

volatile unsigned int in0 = 50;
volatile unsigned int in1 = 50;

volatile unsigned int in2 = 50;
volatile unsigned int in3 = 50;

volatile unsigned int pwm_reload0;
volatile unsigned int pwm_reload1;

volatile unsigned char pwm_state0 = 0;
volatile unsigned char pwm_state1 = 0;
volatile unsigned char count20ms;
xdata volatile unsigned int arm_flag = 0;
xdata volatile unsigned int sound_flag = 0;

unsigned char overflow_count;

char _c51_external_startup (void)
{
      // Disable Watchdog with key sequence
      SFRPAGE = 0x00;
      WDTCN = 0xDE; //First key
      WDTCN = 0xAD; //Second key

      VDM0CN=0x80;       // enable VDD monitor
      RSTSRC=0x02|0x04;  // Enable reset on missing clock detector and VDD

      #if (SYSCLK == 48000000L)
            SFRPAGE = 0x10;
            PFE0CN  = 0x10; // SYSCLK < 50 MHz.
            SFRPAGE = 0x00;
      #elif (SYSCLK == 72000000L)
            SFRPAGE = 0x10;
            PFE0CN  = 0x20; // SYSCLK < 75 MHz.
            SFRPAGE = 0x00;
      #endif

      #if (SYSCLK == 12250000L)
            CLKSEL = 0x10;
            CLKSEL = 0x10;
            while ((CLKSEL & 0x80) == 0);
      #elif (SYSCLK == 24500000L)
            CLKSEL = 0x00;
            CLKSEL = 0x00;
            while ((CLKSEL & 0x80) == 0);
      #elif (SYSCLK == 48000000L)
            //  Before  setting  clock  to  48  MHz,  must  transition  to  24.5  MHz
first
            CLKSEL = 0x00;
            CLKSEL = 0x00;
            while ((CLKSEL & 0x80) == 0);
            CLKSEL = 0x07;
            CLKSEL = 0x07;
            while ((CLKSEL & 0x80) == 0);
      #elif (SYSCLK == 72000000L)
            //  Before  setting  clock  to  72  MHz,  must  transition  to  24.5  MHz
first
            CLKSEL = 0x00;
            CLKSEL = 0x00;
```

```
		while ((CLKSEL & 0x80) == 0);
		CLKSEL = 0x03;
		CLKSEL = 0x03;
		while ((CLKSEL & 0x80) == 0);
	#else
		#error SYSCLK must be either 12250000L, 24500000L, 48000000L, or 
72000000L
	#endif

	#if ( ((SYSCLK/BAUDRATE)/(12L*2L)) > 0x100)
		#error Can not configure baudrate using timer 1
	#endif

		// Configure Uart 0
	#if (((SYSCLK/BAUDRATE)/(2L*12L))>0xFFL)
		#error   Timer   0   reload   value   is   incorrect   because 
(SYSCLK/BAUDRATE)/(2L*12L) > 0xFF
	#endif

	// Configure the pins used for square output
	P1MDOUT|=0b_1000_0000;
	P2MDOUT|=0b_0110_0011;
	P0MDOUT |= 0x10; // Enable UART0 TX as push-pull output
	XBR0     = 0x01; // Enable UART0 on P0.4(TX) and P0.5(RX)
	XBR1     = 0X10; // Enable T0 on P0.0
	XBR2     = 0x40; // Enable crossbar and weak pull-ups

	// initialize timer0 for system clock
	TR0=0; // stop timer 0
	TMOD =(TMOD&0xf0)|0x01; // T0=16bit timer
	TMR0=TIMER0_RELOAD_VALUE;
	TR0=1; // start timer 0
	ET0=1; // enable timer 0 interrupt

	// Configure Uart 0
	SCON0 = 0x10;
	CKCON0 |= 0b_0000_0000 ; // Timer 1 uses the system clock divided by 12.
	TH1 = 0x100-((SYSCLK/BAUDRATE)/(2L*12L));
	TL1 = TH1;      // Init Timer1
	TMOD &= ~0xf0;  // TMOD: timer 1 in 8-bit auto-reload
	TMOD |=  0x20;
	TR1 = 1; // START Timer1
	TI = 1;  // Indicate TX0 ready

	// SPI inititialization
	// SPI0CKR = (SYSCLK/(2*F_SCK_MAX))-1;
	// SPI0CFG = 0b_0100_0000; //SPI in master mode
	// SPI0CN0 = 0b_0000_0001; //SPI enabled and in three wire mode
	// Initialize timer 2 for periodic interrupts
	TMR2CN0=0x00;   // Stop Timer2; Clear TF2;
	CKCON0|=0b_0001_0000; // Timer 2 uses the system clock
	TMR2RL=(0x10000L-(SYSCLK/10000L)); // Initialize reload value
	TMR2=0xffff;   // Set to reload immediately
	ET2=1;         // Enable Timer2 interrupts
	TR2=1;         // Start Timer2 (TMR2CN is bit addressable)

	// Initialize timer 5 for periodic interrupts
	SFRPAGE=0x10;
	TMR5CN0=0x00;   // Stop Timer5; Clear TF5;
	pwm_reload0=0x10000L-(SYSCLK*1.5e-3)/12.0;  //  1.5  miliseconds  pulse  is
the center of the servo
	pwm_reload1=0x10000L-(SYSCLK*1.5e-3)/12.0;  //  1.5  miliseconds  pulse  is
the center of the servo
	TMR5=0xffff;   // Set to reload immediately
	EIE2|=0b_0000_1000; // Enable Timer5 interrupts
	TR5=1;         // Start Timer5 (TMR5CN0 is bit addressable)

	EA=1; // Enable interrupts

	SFRPAGE=0x00;

	return 0;
}

void InitADC (void)
{
	SFRPAGE = 0x00;
	ADC0CN1 = 0b_10_000_000; //14-bit,  Right justified no shifting applied,
perform and Accumulate 1 conversion.
	ADC0CF0 = 0b_11111_0_00; // SYSCLK/32
	ADC0CF1 = 0b_0_0_011110; // Same as default for now
	ADC0CN0 = 0b_0_0_0_0_0_00_0; // Same as default for now
	ADC0CF2 = 0b_0_01_11111 ; // GND pin, Vref=VDD
	ADC0CN2  =  0b_0_000_0000;   //  Same  as  default  for  now.  ADC0  conversion
initiated on write of 1 to ADBUSY.
	ADEN=1; // Enable ADC
}

void Timer3us(unsigned char us)
{
	unsigned char i;               // usec counter

	// The input for Timer 3 is selected as SYSCLK by setting T3ML (bit 6) of
CKCON0:
	CKCON0|=0b_0100_0000;
```

```
		TMR3RL = (-(SYSCLK)/1000000L); // Set Timer3 to overflow in 1us.
		TMR3 = TMR3RL;                 // Initialize Timer3 for first overflow

		TMR3CN0 = 0x04;                 // Sart Timer3 and clear overflow flag
		for (i = 0; i < us; i++)       // Count <us> overflows
		{
			while (!(TMR3CN0 & 0x80));  // Wait for overflow
			TMR3CN0 &= ~(0x80);         // Clear overflow indicator
			if (TF0)
			{
			   TF0=0;
			   overflow_count++;
			}
		}
		TMR3CN0 = 0 ;                   // Stop Timer3 and clear overflow flag
}

void waitms (unsigned int ms)
{
		unsigned int j;
		for(j=ms; j!=0; j--)
		{
			Timer3us(249);
			Timer3us(249);
			Timer3us(249);
			Timer3us(250);
		}
}

void TIMER0_Init(void)
{
		TMOD&=0b_1111_0000; // Set the bits of Timer/Counter 0 to zero
		TMOD|=0b_0000_0101; // Timer/Counter 0 used as a 16-bit counter
		TR0=0; // Stop Timer/Counter 0
}


void InitPinADC (unsigned char portno, unsigned char pinno)
 {
		unsigned char mask;

		mask=1<<pinno;

		SFRPAGE = 0x20;
		switch (portno)
		{
			case 0:
```

```
                P0MDIN &= (~mask); // Set pin as analog input
                P0SKIP |= mask; // Skip Crossbar decoding for this pin
                break;
                case 1:
                P1MDIN &= (~mask); // Set pin as analog input
                P1SKIP |= mask; // Skip Crossbar decoding for this pin
                break;
                case 2:
                P2MDIN &= (~mask); // Set pin as analog input
                P2SKIP |= mask; // Skip Crossbar decoding for this pin
                break;
                default:
                break;
            }
            SFRPAGE = 0x00;
        }

 unsigned int ADC_at_Pin(unsigned char pin)
        {
        ADC0MX = pin;   // Select input from pin
        ADBUSY=1;       // Dummy conversion first to select new pin
        while (ADBUSY); // Wait for dummy conversion to finish
        ADBUSY = 1;     // Convert voltage at the pin
        while (ADBUSY); // Wait for conversion to complete
        return (ADC0);
 }

 float Volts_at_Pin(unsigned char pin)
 {
        return ((ADC_at_Pin(pin)*VDD)/0b_0011_1111_1111_1111);
 }


void Timer0_ISR (void) interrupt INTERRUPT_TIMER0
{
        TMR0=TIMER0_RELOAD_VALUE;
        timer_count++;
}

void Timer2_ISR (void) interrupt INTERRUPT_TIMER2
{
        TF2H = 0; // Clear Timer2 interrupt flag

        if (sound_flag == 1) {
            SOUNDPIN=!SOUNDPIN;
        } else {
            pwm_count0++;
            if(pwm_count0>100) pwm_count0=0;
            OUT0=pwm_count0>in0?0:1;
            OUT1=pwm_count0>in1?0:1;

            pwm_count1++;
            if(pwm_count1>100) pwm_count1=0;

            OUT2=pwm_count1>in2?0:1;
            OUT3=pwm_count1>in3?0:1;
      }
}

void Timer5_ISR (void) interrupt INTERRUPT_TIMER5
{
      SFRPAGE=0x10;
      TF5H = 0; // Clear Timer5 interrupt flag
      // Since the maximum time we can achieve with this timer in the
      // configuration above is about 10ms, implement a simple state
      // machine to produce the required 20ms period.
      if (arm_flag == 0){
            switch (pwm_state0)
            {
               case 0:
                  PWMOUT0=1;
                  TMR5RL=RELOAD_10MS;
                  pwm_state0=1;
                  count20ms++;
               break;
               case 1:
                  PWMOUT0=0;
                  TMR5RL=RELOAD_10MS-pwm_reload0;
                  pwm_state0=2;
               break;
               default:
                  PWMOUT0=0;
                  TMR5RL=pwm_reload0;
                  pwm_state0=0;
               break;
            }
      }     else if (arm_flag == 1) {
                  switch (pwm_state1)
                  {
                     case 0:
                        PWMOUT1=1;
                        TMR5RL=RELOAD_10MS;
                        pwm_state1=1;
                        count20ms++;
                     break;
```

```
                    case 1:
                       PWMOUT1=0;
                       TMR5RL=RELOAD_10MS-pwm_reload1;
                       pwm_state1=2;
                    break;
                    default:
                       PWMOUT1=0;
                       TMR5RL=pwm_reload1;
                       pwm_state1=0;
                    break;
                 }
      }

}

unsigned char AsciiToHex(char * buff)
{
      return ((buff[0]-'0')*0x10)+(buff[1]-'0');
}


// Frequencies for equal-tempered scale, A4 = 440 Hz
// http://pages.mtu.edu/~suits/notefreqs.html

//2020 - April 01: pitch adjusting
const float FTone[] =
{
         30.87,32.70,34.65,36.71,38.89,41.20,43.65,46.25,49.00,51.91,
         55.00,58.27,
         61.74,   65.41,   69.30,   73.42,   77.78,   82.41,   87.31,   92.50,
         98.00,  103.83,  110.00,  116.54,  123.47,  130.81,  138.59,  146.83,
        155.56,  164.81,  174.61,  185.00,  196.00,  207.65,  220.00,  233.08,
        246.94,  261.62,  277.18,  293.66,  311.13,  329.63,  349.23,  369.99,
        391.99,  415.30,  440.00,  466.16,  493.88,  523.25,  554.36,  587.33,
        622.25,  659.25,  698.45,  739.98,  783.99,  830.60,  879.99,  932.32,
        987.76, 1046.50, 1108.72, 1174.65, 1244.50, 1318.50, 1396.90, 1479.97,
       1567.97, 1661.21, 1759.99, 1864.64, 1975.52, 2092.99, 2217.45, 2349.30,
       2489.00, 2637.00, 2793.81, 2959.94, 3135.94, 3322.42, 3519.98, 3729.29,
       3951.04};

//Use timer 2 to play the note.
void PlayNote(void)
{
      int tmsec, toff;

      // Compute in milliseconds the duration of a note or silence using:
      tmsec= ( (60000L/tempo)*400L ) / actLen;
```

```
	// Set the time when the sound will be turned off:
	toff=(tmsec*style)/8L;

	// Start the sound
	if (note!=0)
	{
		TMR2=TMR2RL=(unsigned
int)(65536.0-((72.0e6)/(FTone[note]*2*12.0)));
		TR2=1;
		//turn flag on when sound out
		//sound_flag = 1;
    }
    else //It is a silence...
    {
      TR2=0; //Turn off timer 2
      //sound_flag = 0;
    }

	//Count the milliseconds for the note or silence
	timer_count=0;
	while(timer_count<tmsec)
	{
		if(timer_count>toff) TR2=0; //Turn off timer 2
	}
	TR2=0; //Turn off timer 2
}

int GetNumber(char * music)
{
	int n=0;
	/*Get the number*/
	while (isdigit(music[cur]) && music[cur]) n=(n*10)+(music[cur++]-'0');
	return n;
}

void ParseMDL(char * music)
{
	bit getout;

	cur=0;
	style=NORMAL;
	TR2=0;

	while(music[cur] && (RI==0))
	{
	    //putchar(music[cur]);
		switch (toupper(music[cur]))
		{
```

```
case '>':
      cur++;
      octave++;
break;

case '<':
      cur++;
      octave--;
break;

case 'O':
      cur++;
      octave=GetNumber(music);
break;

/*Choose a note or pause*/
case 'A': case 'B': case 'C': case 'D':
case 'E': case 'F': case 'G': case 'P':

      /*Select the note number from name and octave*/
      note=(octave*12);
      switch (toupper(music[cur])-'A')
      {
            case 0:  note+=BASE_A; break;
            case 1:  note+=BASE_B; break;
            case 2:  note+=BASE_C; break;
            case 3:  note+=BASE_D; break;
            case 4:  note+=BASE_E; break;
            case 5:  note+=BASE_F; break;
            case 6:  note+=BASE_G; break;
            default: note =PAUSE ; break;
      }

      cur++;
      actLen=defLen;

      getout=0;
      while(!getout)
      {
            switch (toupper(music[cur]))
            {
                  case '+': case '#':
                        cur++;
                        note++;
                  break;

                  case '-':
                        cur++;
```

```
                                    note--;
                              break;

                              case '.':
                                    cur++;
                                    actLen=(actLen*2)/3;
                              break;

                              /*Get note duration*/
                              case  '0':  case  '1':  case  '2':  case  '3':
case '4':
                              case  '5':  case  '6':  case  '7':  case  '8':
case '9':

actLen=GetNumber(music)*100;/*Increase resolution*/
                              break;

                              default:
                                    /*Play note and continue*/
                                    PlayNote();
                                    getout=1;
                              break;
                        }
                  }
            break;

            /*Play a note by its number*/
            case 'N':
                  cur++;
                  note=GetNumber(music);
                  actLen=defLen;
                  PlayNote();
            break;

            /*Set the default note duration*/
            case 'L':
                  cur++;
                  defLen=GetNumber(music)*100;/*Increase resolution*/
            break;

            /*Choose the playing style. Ignore foreground and
            background commands, since only background works here*/
            case 'M':
                  cur++;
                  switch (toupper(music[cur]))
                  {
                        case 'N': style=NORMAL;   break;
                        case 'L': style=LEGATO;   break;
```

```
                              case 'S': style=STACCATO; break;
                        }
                        cur++;
                  break;

                  /*Select the tempo*/
                  case 'T':
                        cur++;
                        tempo=GetNumber(music);
                  break;

                  /*Ignore and discard substrings.*/
                  case 'X':
                        cur++;
                        /*Discard substring*/
                        while ((music[cur]!='$') && music[cur]) cur++;
                        cur++;/*Discard also the "$"*/
                  break;

                  /*Unknown   commands   and   blanks   are   just   ignored   and
discarded*/
                  default:
                        cur++;
                  break;
            }
      }
      //TR2=1;
}

void arm_pick_up(void) {            //picks up coins
      PWMMAG = 0;
      waitms(500);
      arm_flag = 1;
      pwm_reload1=0x10000L-(SYSCLK*2.3*1.0e-3)/12.0;        //down
      PWMMAG = 1;                               //electromagnet on
      waitms(500);
      arm_flag = 0;
      pwm_reload0=0x10000L-(SYSCLK*2.0*1.0e-3)/12.0;        //sweep left
      waitms(500);
      arm_flag = 1;
      pwm_reload1=0x10000L-(SYSCLK*0.6*1.0e-3)/12.0;        //pick up
      waitms(500);
      arm_flag = 0;
      pwm_reload0=0x10000L-(SYSCLK*0.9*1.0e-3)/12.0;        //carry right
      waitms(500);
      arm_flag = 1;
      pwm_reload1=0x10000L-(SYSCLK*1.0*1.0e-3)/12.0;        //drop
      PWMMAG = 0;
//Electromagnet off
      waitms(500);
      arm_flag = 0;
      pwm_reload0=0x10000L-(SYSCLK*1.2*1.1e-3)/12.0;         //centered
      waitms(500);
}

void arm_reset(void) {         //resets and centers arm
      PWMMAG = 0;
      //Electromagnet off
      arm_flag = 1;
      pwm_reload1=0x10000L-(SYSCLK*1.0*1.0e-3)/12.0;         //up
      waitms(500);
      arm_flag = 0;
      pwm_reload0=0x10000L-(SYSCLK*1.3*1.0e-3)/12.0;         //centered
      waitms(500);
}

 uint8_t spi_transfer(uint8_t tx)
 {

    SPI0DAT=tx;
    while(!SPIF);
    SPIF=0;
      return SPI0DAT;
 }

/*********** MAIN CODE ***********/
void main (void)
{
      int dir=5;
      int ctlIn;
      int Abutton=0;
      int Bbutton=0;
      int Xbutton=0;
      int Ybutton=0;
      uint8_t temp;
      xdata uint8_t data_array[32];
      const uint8_t tx_address[] = "TXADD";
      const uint8_t rx_address[] = "RXADD";

      unsigned long frequency;
      unsigned long freq_init;
      float volt_init[2];
      float v[2];
      int coin_count = 0;
      char c;
      int mode_flag=0;

      xdata int Abutton_flag=0;
      xdata int Bbutton_flag=0;

      //SOUND
      sound_flag = 1;
      ParseMDL(starttune);           //mario start song
      sound_flag = 0;
      TR2=1;

      in0 = 50;
      in1 = 50;
      in2 = 50;
      in3 = 50;

      //TIMER
      TIMER0_Init();

      //ADC
      InitPinADC(1, 1);
      InitPinADC(1, 2);
    InitADC();

    count20ms=0;         //  Count20ms  is  an  atomic  variable,  so  no  problem
sharing with timer 5 ISR
    waitms(500);         //wait for putty to start

    /*********** ADC Voltage **********/
      volt_init[0] = Volts_at_Pin(QFP32_MUX_P1_1);
      volt_init[1] = Volts_at_Pin(QFP32_MUX_P1_2);
      //printf("\rPerimeter     Detector:     P1.1=%7.5fV,     P1.2=%7.5fV\r",
volt_init[0], volt_init[1]);


    /*********** FREQUENCY ***********/
      //initial frequency
      TL0=0;
      TH0=0;
      overflow_count=0;
      TF0=0;
      TR0=1; // Start Timer/Counter 0

      waitms(200);
      TR0=0; // Stop Timer/Counter 0
      freq_init=overflow_count*0x10000L+TH0*0x100L+TL0;
      //printf("\rMetal Detector: f=%luHz\n", freq_init);
```

```
	//while(freq_init< 50000);	//ensures that frequency readings are
correct

	arm_reset();

	in0 = 80;
	in1 = 20;
	in2 = 80;
	in3 = 20;

	while(1)
	{
		TL0=0;
		TH0=0;
		overflow_count=0;
		TF0=0;
		TR0=1; // Start Timer/Counter 0

		waitms(200);//required!!
		TR0=0; // Stop Timer/Counter 0
		frequency=overflow_count*0x10000L+TH0*0x100L+TL0;

		v[0] = Volts_at_Pin(QFP32_MUX_P1_1);
		v[1] = Volts_at_Pin(QFP32_MUX_P1_2);


		printf("\x1b[2J"); // Clear screen using ANSI escape sequence.
		printf("\033[%d;%dH", 1, 1);	//set cursor
		printf("\rMetal Detector: f=%luHz\n", frequency);
		printf("\rPerimeter  Detector:  P1.1=%7.5fV,  P1.2=%7.5fV\n",  v[0],
v[1]);
		printf("\rCoins: %d", coin_count);

		//CHECK METAL DETECTOR
		if (frequency >= freq_init + 30) {		//30
			in0 = 20;
			in1 = 80;
			in2 = 20;
			in3 = 80;
			waitms(420);
			in0 = 50;
			in1 = 50;
			in2 = 50;
			in3 = 50;

			// sound_flag = 1;
		 //	ParseMDL(cointune);
		 //	sound_flag = 0;
```

```
             //   TR2=1;

             //   in0 = 50;
                  // in1 = 50;
                  // in2 = 50;
                  // in3 = 50;

                  arm_pick_up();

                  coin_count++;
            } else {
                  in0 = 80;
                  in1 = 20;
                  in2 = 80;
                  in3 = 20;
            }

            //CHECK PERIMETER DETECTOR
            if  ((v[0]  >=  volt_init[0]  +  1.200)  ||  (v[1]  >=  volt_init[1]  +
1.200)){
                  in0 = 20;
                  in1 = 80;
                  in2 = 20;
                  in3 = 80;
                  waitms(1000);
                  in0 = 80;
                  in1 = 20;
                  in2 = 20;
                  in3 = 80;
                  waitms(1500);
                  in0 = 60;
                  in1 = 40;
                  in2 = 60;
                  in3 = 40;
            }

            /********** SWITCH TO REMOTE CONTROL ************/
             while(coin_count >= 20){

                  if(mode_flag==0){

                        ADEN=0;
                        XBR2=0;

                  /*SPI Setup, overwrites frequency detector stuff*/
                        //P0MDOUT=0b_0001_1101;//SCK,   MOSI,   P0.3,   TX0   are
puspull, all others open-drain
                        P0SKIP=0b_0000_0001;
                        P0MDOUT=0b_0101_1010;
                        //P0MDIN=0;
                        //P1MDOUT=0;
                        //P2MDOUT=0;
                        XBR0=0b_0000_0011;//SPI0E=1, URT0E=1
                        XBR1=0b_0000_0000;
                        XBR2=0b_0100_0000; // Enable crossbar and weak pull-ups

                              // SPI inititialization
                        SPI0CKR = (SYSCLK/(2*F_SCK_MAX))-1;
                        SPI0CFG = 0b_0100_0000; //SPI in master mode
                        SPI0CN0 = 0b_0000_0001; //SPI enabled and in three wire
mode


                        //RECIEVER
                  nrf24_init();                              //  init  hardware
pins  (if  there  are  problems  comment  out  line  modifying  P0MDOUT  in  this
function)
                        nrf24_config(120,32);                //       Configure
channel and payload size
                  nrf24_tx_address(rx_address);  //set device as reciever
                        nrf24_rx_address(tx_address);
                        printf("manual mode");
                        mode_flag=1;
                  }
                                          //once 3 coins are picked up
                  if(nrf24_dataReady())
         {
             nrf24_getData(data_array);
             ctlIn =atoi(data_array);      //converts   string   recieved   from
radio into an integer


  //           /*ctlIn is formated as follows:
  //               the 5 bit positions carry the values for the push buttons and
the analog stick direction
  //             Starting from the leftmost bit:
  //             A button  (0 or 1)
  //             B button  (0 or 1)
  //             X button  (0 or 1)
  //             Y button  (0 or 1)
  //             direction (1-9) */

             if(ctlIn>=10000){
                  Abutton=1;
                  ctlIn=ctlIn%10000; //removes the leftmost bit if set
             } else
```

```
                    Abutton=0;

               if(ctlIn>=1000){
                    Bbutton=1;
                    ctlIn=ctlIn%1000;  //removes the leftmost bit if set
               } else
                    Bbutton=0;

               if(ctlIn>=100){
                    Xbutton=1;
                    ctlIn=ctlIn%100;   //removes the leftmost bit if set
               } else
                    Xbutton=0;
               if(ctlIn>=10){
                    Ybutton=1;
                    ctlIn=ctlIn%10;          //removes the leftmost bit if set
               } else
                    Ybutton=0;

              dir=ctlIn;
                printf("IN: %s\r\n", data_array); //Prints data that is recieved,
may want to remove
              //printf("IN: %i%i%i%i%i",Abutton,Bbutton,Xbutton,Ybutton,dir);

  //           /* FOR DEBUGGING ONLY
  //           if(inputs==0)
  //              printf("A button pressed");
  //           printf("ctlIn:%i",ctlIn);
  //           printf("button=%i,dir=%i",button,dir);

          }

           if(RI) //Other radio junk, stuff to do with handling lost messages and
such
          {
             //safe_gets(data_array, sizeof(data_array), 2000);
             gets(data_array);
                 //printf("\r\n");
                nrf24_send(data_array);
                  while(nrf24_isSending());
                  temp = nrf24_lastMessageStatus();
                   if(temp == NRF24_MESSAGE_LOST)
                  {
                       //printf("> Message lost\r\n"); //for debugging, may want
to remove
                        dir = NO_MOVEMENT; //makes robot not move if message is
lost
                     Abutton=0;
                  }
                    nrf24_powerDown();
             nrf24_powerUpRx();
              }


             // /*Motor Control:
             //    Directions are defined at the top of the file and look like
this:
             //    7     8     9
             //    4     5     6
             //    1     2     3
             //    This is based of a standard dial pad, with the analog stick
centered at 5

                        if (dir == FORWARD) {
                              in0 = 80;
                              in1 = 20;
                              in2 = 80;
                              in3 = 20;
                        } else if (dir == BACKWARD) {
                              in0 = 20;
                              in1 = 80;
                              in2 = 20;
                              in3 = 80;
                        } else if (dir == FORWARD_RIGHT) {
                              in0 = 70;
                              in1 = 30;
                              in2 = 50;
                              in3 = 50;
                        } else if (dir == FORWARD_LEFT) {
                              in0 = 50;
                              in1 = 50;
                              in2 = 70;
                              in3 = 30;
                        } else if (dir == RIGHT) {
                              in0 = 70;
                              in1 = 30;
                              in2 = 30;
                              in3 = 70;
                        } else if (dir == LEFT) {
                              in0 = 30;
                              in1 = 70;
                              in2 = 70;
                              in3 = 30;
                        } else if (dir == BACKWARD_RIGHT) {
                              in0 = 30;
                              in1 = 70;
```

```
                                    in2 = 50;
                                    in3 = 50;
                            } else if (dir == BACKWARD_LEFT) {
                                    in0 = 50;
                                    in1 = 50;
                                    in2 = 30;
                                    in3 = 70;
                            } else if (dir == NO_MOVEMENT) {
                                    in0 = 50;
                                    in1 = 50;
                                    in2 = 50;
                                    in3 = 50;
                            }


                            if(Abutton){
                                    Abutton_flag=1;
                            }
                            else{
                                    if(Abutton_flag){
                                            arm_pick_up();
                                            printf("pickup");
                                            Abutton_flag=0;
                                    }
                            }


                            if(Bbutton){
                                    Bbutton_flag=1;
                            }
                            else{
                                    if(Bbutton_flag){
                                            sound_flag = 1;
                                            ParseMDL(starttune);           //mario
start song

                                            sound_flag = 0;
                                            TR2=1;
                                            Bbutton_flag=0;
                                    }
                            }



              }
    }
}
```