TABLE OF CONTENTS

1 Introduction

1.1 Design Objective

The project aimed to design, build, program, and test a Coin Picking Robot and Controller using two separate microcontrollers. The robot is used to pick up coins manually by taking inputs from the controller as well as automatically with no inputs.

1.2 Design Specification

Hardware Component Specifications:

General:

- EFM8 Microcontroller
- ATMega348 Microcontroller
- USB Adapter: BO230X
- Various Resistors of 5% Tolerance (Refer to Hardware Circuit)
- Tactile Push Buttons
- CEM-1203 Buzzer
- JDY-40
- LCD
- Various Capacitors (Refer to Hardware Circuit)
- LM7805 5V Voltage Regulator
- MCP1700 3.3V Voltage Regulator
- PS2 Joystick
- White LEDs

Robot Arm + Sensors

- MG90S Micro Servos
- Electromagnetic Wire
- LTV846 Optocoupler
- M8275-ND 1mH Inductors
- LM358 Op Amp

H-Bridge

- DC Motors
- LTV846 Optocoupler
- NTD3055L104 - N-Type-Mosfet
- NTD2955-1G - P-Type-Mosfet

Software Specifications:

Automatic Mode:

- detects and picks up 20 coins in a set area
- never leave the set area
- after done picking up 20 coins, start receiving inputs from controller again

Manual Mode:

- able to move forward, backward, turn left, turn right, and pick coin up
- receives data from controller
- display magnet strength on lcd and using a buzzer

2 Investigation:

2.1 Idea Generation

Before generating ideas and formulating working hypotheses, our group focused on thoroughly understanding the project requirements and the components provided. We began by assembling the robot's body according to the provided instructions, constructing the chassis, sensor module, and electromagnet. Using only the power supply, we tested and observed the basic operation of the DC motors and the electromagnet, ensuring we understood how each component functioned individually. To prepare for system integration, we reviewed the lecture materials related to JDY-40 communication between microcontrollers. Once we had a solid grasp of each subsystem, we conducted a brainstorming session and developed our working hypothesis: by implementing proper optocoupling to isolate grounds and ensuring a reliable JDY-40 connection between the microcontrollers—along with a functional button interface—we could successfully implement both automatic and manual modes while minimizing electrical noise within the robot system.

2.2 Investigation Design

To ensure our robot would function as intended, we planned to investigate three main areas: sensor behavior, wireless communication, and power delivery.

For sensing, we aimed to understand how oscillating currents affected the frequency of the perimeter-detecting inductors and how metal objects disrupted the period of the metal-sensing inductor. A specific challenge we anticipated was detecting the perimeter wire when it ran parallel to the robot. To explore this, we planned to use the professor's circuit schematics and lab tools like oscilloscopes and multimeters to analyze signal behavior and optimize sensor placement.

For communication, we intended to study how the JDY-40 modules handled data transmission between the controller and the robot. We planned to use a separate EFM8 microcontroller to simulate the robot and monitor data exchange through the Putty interface, helping us verify that commands were properly sent, received, and interpreted.

Finally, we recognized the importance of verifying power requirements as we integrated each hardware component. We planned to test voltage rails and system performance incrementally, identifying whether power delivery was sufficient or if optocouplers were necessary to isolate high-power components and prevent interference.

2.3 Data Collection

To collect data from the inductors, we used an oscilloscope, metal coins, and a short segment of perimeter wire. A function generator simulated the perimeter signal by generating an oscillating voltage through the wire. We gradually increased the voltage output to ensure a clear and consistent response from the inductors. Frequency changes in the perimeter-detecting inductors and period changes in the metal-detecting inductor were closely observed. To complement the visual data from the oscilloscope, we also printed real-time sensor readings to the Putty interface for a clearer numerical analysis. While testing

the metal detector, we replaced the perimeter wire with metal coins to observe how the presence of metal altered the inductor's signal period.
For JDY-40 communication testing, we first verified that both transmitter and receiver circuits were correctly built and free of hardware faults. Using the base code provided, we established a working wireless link between two microcontrollers. To monitor and understand the data transmission, we ran Putty on two separate computers—one for the controller and one for the robot. We sent commands from the controller and used a connected LCD on the robot side to confirm receipt and correct interpretation of messages. To improve transmission reliability, we fine-tuned the sending frequency, lowering it to a rate that maximized communication accuracy and stability.

2.4 Data Synthesis

After confirming that each individual component functioned as expected, we began integrating the system step by step. Each component—such as the H-bridge for the DC motors, two servos, electromagnet, perimeter detector, and metal detector—was added to the breadboard one at a time. After every addition, we powered the circuit and verified that all previously installed components continued to operate correctly.
During this process, we identified significant electrical noise when high-power components were active. To address this, we opted to isolate the servos, electromagnet, and H-bridge using optocouplers. Additionally, we powered these components with an external 6V battery to ensure they received sufficient current without overloading the microcontroller's supply.
Once all hardware was integrated across both breadboards, we verified full system functionality, including successful wireless communication via the JDY-40 modules. We also confirmed that the controller's button input reliably switched between manual and automatic modes, ensuring consistent system responsiveness across all subsystems.

2.5 Analysis of Results

Our final tests confirmed that the robot was able to reliably pick up 20 coins in both automatic and manual modes, with seamless switching between the two via the JDY-40 wireless connection. We validated these results through repeated trials conducted over multiple days, ensuring consistent performance across varying conditions. Throughout the project, we encountered and overcame several challenges. One key issue was battery reliability—our 9V power source would gradually drain, leading to inconsistent results in tests conducted on subsequent days. Over the course of the project, we had to replace the battery five times to maintain stable performance. Another challenge was the limited space on the breadboard, which required us to find creative and efficient ways to fit all the components within the constraints. Despite these hurdles, we successfully built a fully functional robot capable of switching between manual and autonomous operation, detecting and picking up metal coins, and navigating using perimeter detection.

3 Design:

3.1 Use of Process:

The first thing our group did when starting the project was to look at the project requirements. After making sure we understood the need and scope, we broke the project down into smaller parts. The parts included:

- hardware of the robot
- hardware of controller
- software of controller (master)
- software of robot manual mode (slave)
- software of robot automatic mode

We used a combination of the knowledge we received from the lecture, as well as the base code files provided as a starting point. Our team was "split" into a hardware and software side, but we all contributed to both parts. We took advantage of makefiles and Github to manage our files for the project. This allowed for code to be updated easily and organized. Of course, throughout the project there were many issues that our team had to overcome through countless revisions. After completing the base requirements for the project, we started to think about what bonus features we could implement. After brainstorming features that were both feasible and practical, we worked together to add them.

3.2 Need and Constraint Identification:

To identify the needs and limitations of the project, our group looked at the Project 2 pdf on Canvas. By reading this document as well as what we were told in class, we found these requirements:

- We had to use two different microcontroller systems from different families. One for the controller and one for the robot.
- Both the robot and controller had to be powered by external batteries.
- We had to detect 0.05$, 0.1$, 0.25$, 1$, and 2$ Canadian coins via a magnet detector
- We had to be able to pick up all of the current Canadian coins in circulation and place them in a container carried by the robot
- The programming of both the robot and controller had to be done in C
- We had to use MOSFETS to create the H bridge for the motors
- The robot had to have an automatic mode that would pick up and collect 20 coins in a region bounded by a wire without leaving the region. The wire had AC current going through it. After picking up 20 coins, the robot had to stop and wait for instructions from the controller.
- The robot had to have a manual mode that is controlled by a controller. In manual mode the robot had to be able to move forward, backwards, turn left, turn right, stop, and pick up a coin. The robot and controller had to receive and transmit through a set of JDY-40 radios.
- The controller had to give instructions to the robot to perform the actions listed above. Additionally, the controller had to display the strength of magnet detection on an LCD screen as well as audio in relation to the magnet strength.

3.3 Problem Specification:

Our group first created our robot and slave to cover all of the needs and constraints listed above. Our robot consisted of two wheels connected to an h-bridge of optocouplers allowing movement in the directions needed. We had an electromagnet connected to two servo motors which allowed us to pick up coins and store them in a basket that was mounted on the robot. We also had inductors on the bottom of the robot that were connected to circuit configurations that let us detect the perimeter as well as coins. The robot was controlled using an EFM8LB1 microcontroller that communicated to the ATMEGA328P microcontroller on our controller using the JDY-40 radios. Our controller had the LCD, speaker, joystick, as well as push buttons to match all design requirements.

3.4 Solution Generation:

For the physical design of our robot, we chose to use the materials provided in the project kit. This included the robot chassis, DC and servo motors, electromagnet, wheels, and inductors. This allowed our robot to function efficiently to meet the project requirements. Some key design features of the final product were:

- Having the electromagnet, servo motors, and DC motors connected to optocouplers to reduce noise to our microcontroller
- Soldering wires together to ensure consistent wiring connections
- Using a 5V and 3.33V regulator so that we could power multiple components using the same source
- Shortening wires for a cleaner and more compact design

3.5 Solution Evaluation:

Our final design was able to meet the requirements stated above. Here is a list of the different components that met the design criteria:

- Use of the ATMEGA328P and the EFM8LB1 met the criteria for two different microcontrollers
- the dc motors met the movement requirements
- the electromagnet along with the two servo motors met the criteria to pick coins
- The set of JDY40 radios met the communication requirements
- The controller LCD and speaker met the criteria of showing magnet strength
- The inductors along with the peak detection and frequency circuits met the requirements for perimeter and magnet detection
- The push buttons on the controller allowed us to toggle between the automatic and manual modes

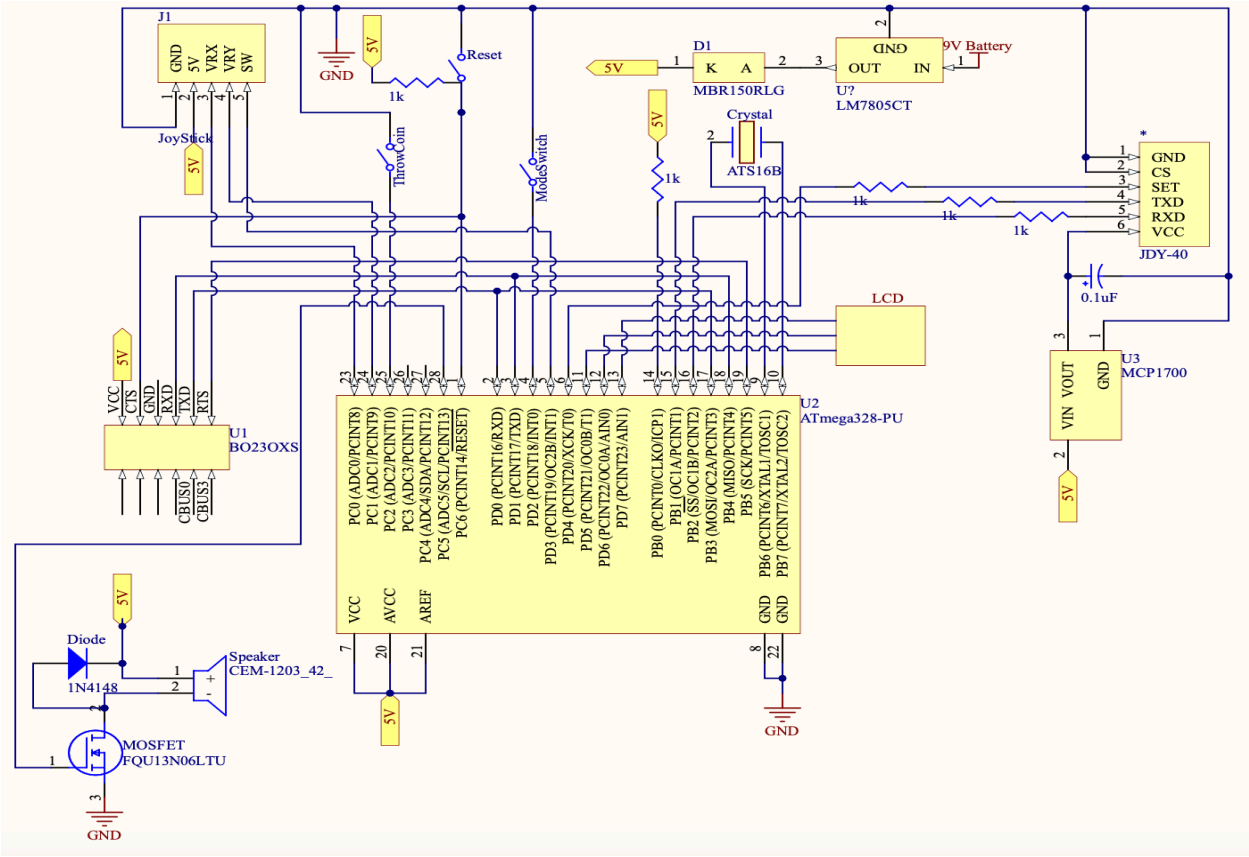
Overall, our final design was developed to efficiently meet the project requirements as well as create room to add bonus features.

3.6 Safety and Professionalism

This project involved dealing with high temperatures, low voltage, and current. There were also fumes involved from the solder and flux. To satisfy proper safety, all members took precautions.
1. Safety Goggles were worn at all times when dealing with live voltage, and soldering.
2. Ventilation was prioritized when soldering to avoid the inhalation of fumes.
3. Hands were washed thoroughly after soldering activities.
4. No food and water were present in the lab or around lab activities outside the lab.
5. Proper attire was worn, covering any exposed skin. Loose hair was tied back, and closed-toed shoes were worn at all times.
6. Any flammable substance or clothing was removed and replaced with more appropriate equipment.
7. The station was cleaned after use, and lab equipment was powered off and returned to their appropriate slots.
8. Any safety issues that were noticed were brought to the lab TA's attention.
All members adhered to the protocols set by the professor before the start of the project. Members worked in groups, being accountable for each other for safety reasons.

3.7 Detailed Design

3.7.1 Hardware Block Diagram



Controller Hardware Block Diagram

3.7.2 Detailed Hardware Specifications

Controller

We powered the controller with a 9V battery which we stepped down to 5V using an LM7805. 5V was used to power the ATMega, speaker, LCD, and the Joystick. The JDY-40 requires 3.3V, therefore we used an MCP1700 to step down the 5V.

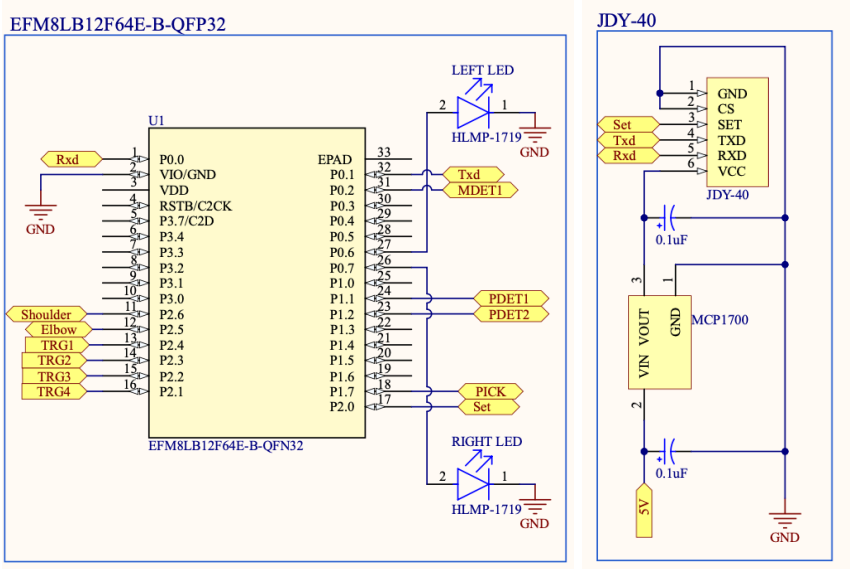
A PS2 Joystick for the robot's movement was used in manual mode. It has pin VRX for turning left and right, and pin VRY for moving forward and backwards. The joystick also comes with a switch which was configured for the coin picking movement. The LCD is simply connected to the ATMega for configuration, and its necessary pins were connected to ground and power.

The speaker is connected with a diode and a MOSFET. The diode is there to protect any flyback voltage going back into the MOSFET. The microcontroller will send the signal to the MOSFET, signalling whether the speaker is on or off. The diode will regulate the voltage back to the MOSFET, thus ensuring its safety. The signal to the MOSFET is sent from pin 28 of the microcontroller.

The JDY-40's RXD, TXD, and SET pins were connected to the microcontroller for transmission and receiving of data from the robot. It was powered by the stepped down 3.3V and had its GND and CS pins connected to ground. A capacitor was connected to ground from the output of the MCP1700 to protect the JDY-40 from noise.

A button was connected to the microcontroller for resetting purposes, and another button was used for switching between automatic and manual mode. Additionally, an extra button was integrated into the controller for a bonus feature.

Robot

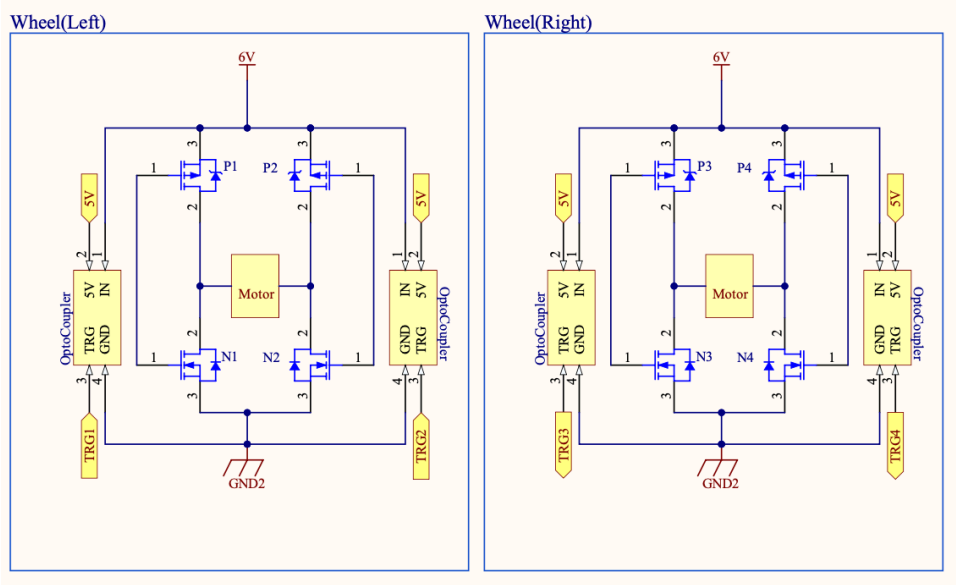


The robot itself was powered by two different power supplies. The DC Motors, Micro-Servos, and the electromagnet were powered by a 6V power source. The EFM8 microcontroller, perimeter detector, and

the metal detector were powered by an external 9V battery which was regulated down to 5V by a LM7805 voltage regulator. The 5V from this regulator was further stepped down to 3.3V to power the JDY-40 through a MCP1700. The components that required power from the 6V power source were separated to a different ground port through the optocouplers to minimize noise. As the components requiring lower voltage would be subjected to excessive noise from the higher voltage being cycled in the same ground.

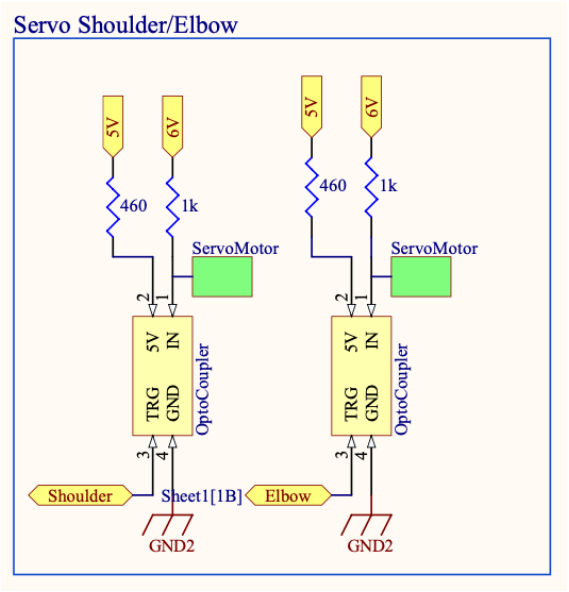
The MCP1700 had capacitors which were connected to ground to regulate noise. The JDY-40 itself had its SET, RXD, and TXD pins connected to the microcontroller for transmission and retrieval of data from the controller. Two white LEDs were used to indicate moving backwards, and rotating left and right.

H-Bridge and DC Motors



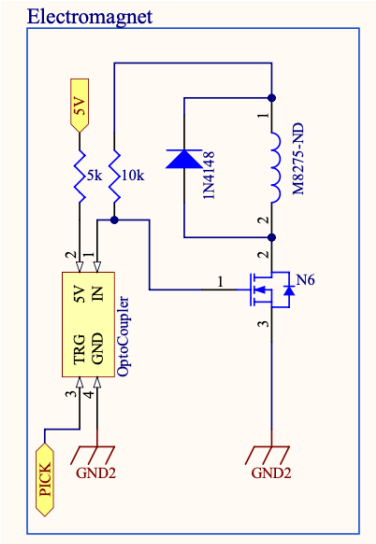
Each of the two DC motors used for the robot's locomotion were controlled by an H-Bridge. The H-bridge's receive digital input from the microcontroller that determines the direction that the motor will spin. Each H-bridge has two inputs, in order to activate the motor one input needs to be set high and the other to ground. To switch the direction of the motor the high and low inputs need to be switched. Both motors and H-bridge's were connected through an optocoupler to prevent noise from the motor from interfering with the microcontroller. The two motors are controlled independently so we can move the robot forwards, backwards, or rotate left and right.

Micro-Servos



There are two servo motors that are used to control the arm that carries the electromagnet. Both servos are connected to the microcontroller through an optocoupler and are powered by the same 6 volt supply as the DC motors and the electromagnet. The servos receive a PWM signal from the microcontroller that determines the precise movement of the motors.

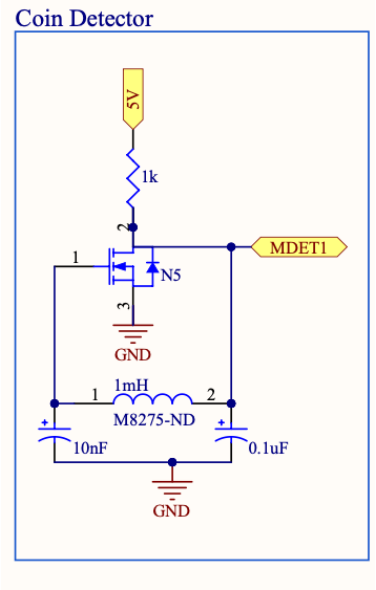
Electromagnet



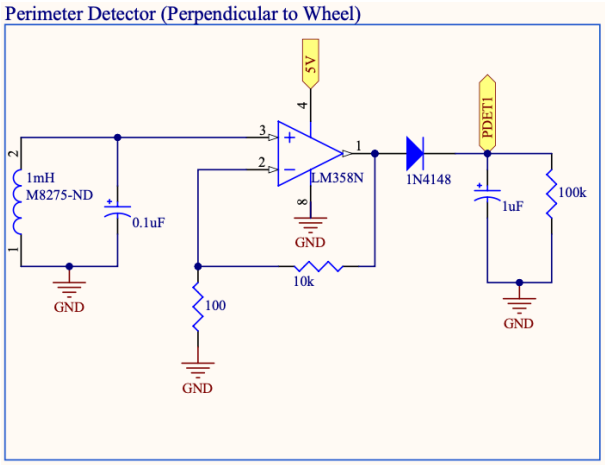
The electromagnet was made by a coil of insulated wire which was then connected to the microcontroller through an optocoupler. Our electromagnet had a resistance of approximately 10 ohms. Our electromagnet would be activated by the microcontroller only when needed to pick up a coin. If it was left on longer than needed it would heat up and drain the battery faster.

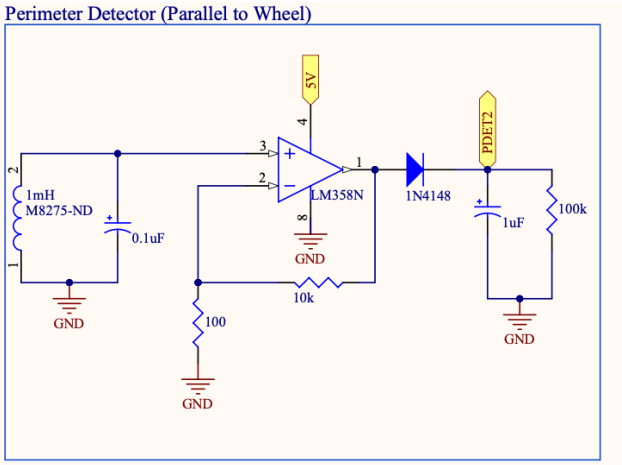
Metal Detector

The metal detector is an inductor at the bottom of the robot which is connected to a Colpitts oscillator circuit on the breadboard. When metal goes near the inductor causing the magnetic permeability of the surrounding space to change, the frequency of the oscillator circuit changes which is then read by the microcontroller. This circuit is powered by the same power and ground as the microcontroller.



Perimeter Detector

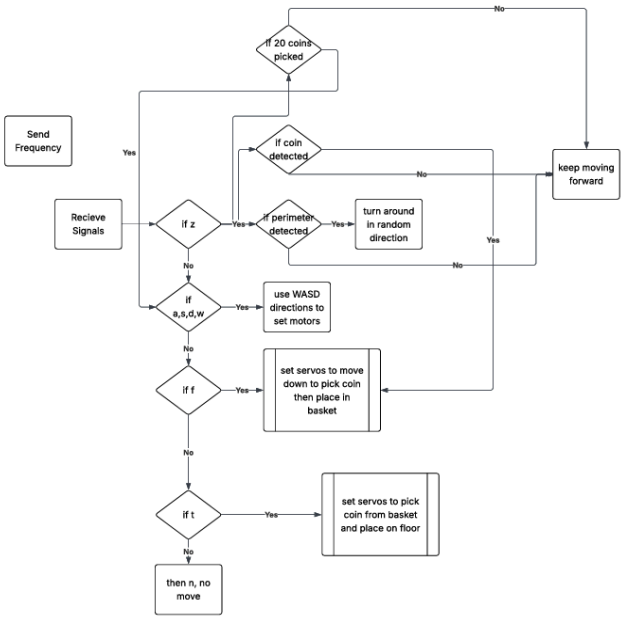




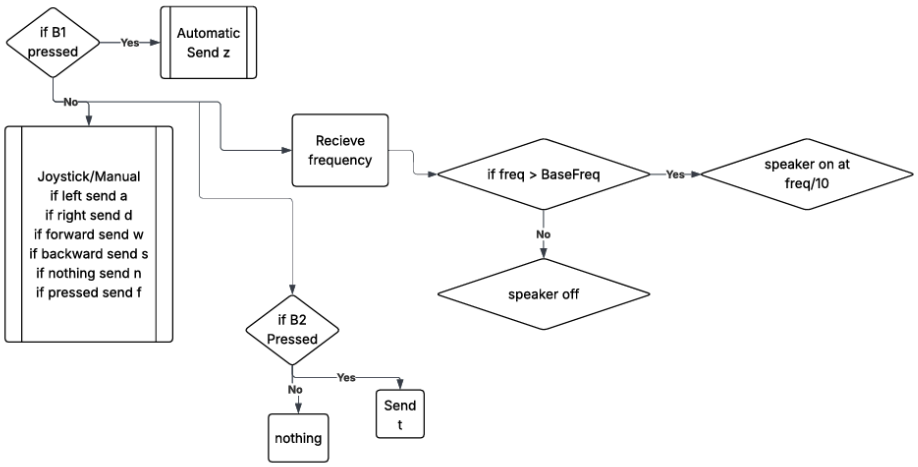
The perimeter detector consists of two inductors each in parallel with a capacitor that are placed perpendicular to one another at the bottom of the robot. The frequency of the current in the inductors is then sent to an amplifier circuit on the breadboard. Without amplification, the signal will not be strong enough to be read by the microcontroller. The perimeter detector circuit is connected to the same power and ground of the microcontroller. The perimeter wire is being supplied with a frequency of 15kHz that is close to the actual resonant frequency of the inductors in parallel with the capacitors.

3.7.3 Software Block Diagram

Slave Software Diagram



Master Software Diagram



3.7.4 Detailed Software Specifications

The coin-collecting robot operates through a the use of two microcontrollers. The **ATmega328P** hardware and software acts as the master transmitting joystick position along with mode switching between automatic and manual. While the **EFM8LB1** acts as the slave and does tasks such as motor control, metal detection, and servo movement.

Mode Switching Manual/ Automatic

In order to switch between the manual and automatic mode for the robot an external pushbutton was used. This pushbutton was configured to switch to manual when it was pressed in automatic mode, and to switch to automatic when pressed in manual mode. A singular button was used due to the lack of pins on the Atmega.

Slave/Master Communication

The slave and master communicated using the **JDY40** chips, in software the channel was adjusted in order to avoid interference with other groups radio channels. There were different strings sent through the master for different movements and modes.

Metal Detector

The metal detector software is simply measuring the period of the transmitted signal from the circuit and if the period lowers we know a coin is detected.

Perimeter Detector

Voltage is received at two pins (since there are two perimeter detectors) of the EMF8 controller and if the received voltage is above a threshold value then the perimeter is detected and a flag is set to indicate that.

Automatic Mode

In automatic mode the only communication made through the radio channels was the sending of ‘z’ from the master to the slave in order to communicate that the slave was in automatic mode. In automatic mode the functions detect coin, detect perimeter and detect while spinning were called constantly and only executed an action if the intended detection had happened which would cause the slave to do the neccessary action.

In detect coin anytime a coin was detected the robot sets the motors to move back then sweep the servo and turn the magnet on then once positioned over the basket the magnet was turned off.

If a perimeter was detected the robot would turn for a random amount of time using a random number function which lead to a function with turns for different amounts of time.

If the robot is spinning it is also configured to stop once it detects a coin and pick it up and then continue with its actions.

The requirement of picking 20 coins then transitioning to manual mode is executed by sending the string ‘k’ to the controller once 20 coins are picked, to which the controller stops sending ‘z’ and instead sends the signals for manual mode which are detailed below. Before this is executed to robot is coded to do a “money dance” to indicate that the requirement is met.

Manual Mode

In manual mode the radio is used more frequently as the controller send signals. If the joystick is in the resting position and no external buttons are pressed then the controller sends ‘n’ indicating no movement. The WASD commands are used to indicate turns and movement. The controller sends ‘f’ when the joystick is pressed to pick a coin. An extra feature is implemented through the use of an external button

which send ‘t’ everytime it is pressed. The function of this feature is to take coins out of the basket of the robot and lay them out so the coins can be picked up once again. A table is provided below summarizing these.

Signal	Action	Signal	Action
‘n’	Indicates no movement	‘d’	Indicates movement right
‘w’	Indicates movement forwards	‘f’	Indicates pick coin servo and magnet
‘a’	Indicates movement left	‘t’	Indicates re-destribution of coins
‘s’	Indicates movement backwards	‘z’	Indicates that the system is automatic mode

As these actions go along the LCD is set to display the strength of the magnet detected by the coin detector by showing the frequency. A buzzer/speaker is connected to a pin on the ATmega and is turned on once a coin is detected this is done through setting a base frequency and the frequency changes based on the strength of the coin detected.

3.8 Solutions Assessment:

Multiple design requirements and constraints need to be met for the coin picking robot to work as expected.

Frequency Sensing Validation/Coin Detection:

In order to determine if there is a coin, along with the different tones in terms of the strength produced by each coin, the frequencies sensed from the coins must be different from the frequency being sensed without any coins. To test the frequencies of each coin (Nickel, dime, quarter, loonie, toonie), we would have test trials with multiple replicates (4 of each) of the same coin to test for accuracy from 1 centimeter away from the metal inductor which is generally where the robot would sense and pick up a coin. After determining the frequency readings from each type of coin along with its replicates, we would determine the frequency of the lowest and highest reading of the coin replicates to know at set range should produce said tone/be this specific type of coin. Additionally, if we noticed a coin holds a different frequency relative to the other same replicates, we would exchange the coin for another version of it that matches the range of frequency we are expecting for the specific coin. This allows for us to have consistency between each coin meaning all coins of the same variation should hold similar frequencies. Moreover, we would determine the frequency sensed with no coin present near the inductor to ensure the frequency readings read on PuTTY have an average that is different from the frequency of each coin. This allows for proper coin pickup at automatic mode to prevent false pickups due to wrong frequency readings.

Table:

Coins @ 1 cm away	Trial 1 (Hz):	Trial 2 (Hz):	Trial 3 (Hz):
Toonie A	41721	41738	41742
Toonie B	41756	41783	41772
Toonie C	41710	41721	41715
Toonie D	41742	41735	41744
Loonie A	41093	41064	41086
Loonie B	41213	41233	41243
Loonie C	41334	41323	41322
Loonie D	41353	41357	41344
Quarter A	41389	41392	41376
Quarter B	41532	41523	41522
Quarter C	41643	41621	41609
Quarter D	41589	41604	41603
Dime A	40530	40521	40534
Dime B	40583	40582	40577
Dime C	40559	40553	40549
Dime D	40592	40586	40593
Nickel A	41032	41025	41022
Nickel B	41076	41076	41073
Nickel C	41075	41073	41073
Nickel D	41053	41055	41052
No Coin	39423	39424	39422

A, B, C, D - Same type of valued but different coin

Claw Pick Up:

The project required the robot to use the electromagnet to pick up coins in manual and automatic mode. When in automatic mode we would use the coins we selected from testing and have the robot powered on moving in a straight line with coins in a line that is close enough (1 centimeter away when robot approaches to pickup) to sense and pick up the coin. Doing this allowed for us to ensure consistency in when the robot should pick up and with what ranges it would sense and pick the coin up. Additionally, this allowed us to test out each type of coin the robot struggled to pick up allowing for more tests and changes made into the code. As for the manual mode, we would activate the electromagnet to pick up on command with all 20 coins to ensure there is no deterioration of the pick up in which we found was consistent with pick ups done manually. This was due to the electromagnet already in an ideal position done by the user before pickups rather than automatic mode.

Wire Perimeter Detection:

The project required a perimeter wire which had to have an area of $0.5m^2$ for the robot set in automatic mode to not exit out of. In order to test out the frequency detected from the perimeter wire, we would test out the frequency wire which was set as a magnitude of 16,000 Hz and when our robot senses this, it would turn away from the wire and move in a different direction. We would test the wire detection by positioning the robot to move at different angles going near contact to the perimeter of the wire including corners. Doing this allowed us to know which parts of the perimeter had trouble sensing while testing the angle at which our robot moved near to the perimeter allowing us to change and adjust the range of frequency read to change directions.

Responsiveness of Controller

In order to determine the responsiveness of the controller in manual mode, we had to test out the toggling between manual and automatic, coin pick up, and direction control. For changing the modes between manual and automatic, pressing onto the pushbutton would show the change made onto PuTTY for us to validate the change consistently. As for coin pick up, pressing onto the joystick would send an output onto PuTTY allowing us to know the controller pick up works on command. Additionally, testing the movement of the robot made us have to test and change delays made from the code. Adding less delay allowed for sharper turning and felt overall much more responsive whereas adding more delay made the robot turn more despite positioning the joystick for the same duration of when we tested with having less delay. Additionally, we would be able to see each input made from the joystick and read it onto PuTTY to validate the correct movements of the controller on command.

Weakness:

A weakness in our design is sensing the coins from the metal inductors. If the coin was around 2 centimeters or more away from the robot and depending on the size of the coin would struggle to detect and pick up coins. We found the toonie was the best for the robot to sense and detect coins due to more frequency along with having a bigger surface area making it hard for the robot to miss and not pick up the coin. Additionally, we found the dime to be the worst for the robot to sense and pick up and due to the surface area of the coin, this made it easier for the robot to miss pickups making it seem inconsistent in pickups with other types of coins.

Strengths:

A strength we had from the project is the responsiveness of the remote. Because we removed as much delay written in the code, it felt very easy to make turns moving forward and reverse, left or right, and pick up coins. Additionally, toggling between manual and automatic worked consistently as expected making the remote work consistently overall.

4 Live Long Learning:

At the beginning of the project, we were less familiar with the concepts required to build the coin-picking robot and controller. We needed to learn how to design a master-slave system with wireless communications and we needed to learn about battery and power management for the robot hardware. By working on the project and collaborating effectively as a team we were able to bridge the knowledge gap and learn everything we needed to construct the robot and controller. By the end of the project, we had gained confidence in the required concepts and enhanced our knowledge of communication protocols, battery management, and complex software and hardware integration. We worked on every part of the project collaboratively, learning together and helping each other when needed.

All members of our team went into the project with a strong foundational knowledge of electronics provided by the other courses we are taking and took last semester. A course that helped us complete the project was ELEC 211. It taught us about magnetism and inductors, concepts which were needed to understand the electromagnet and perimeter and metal detectors. The previous projects in ELEC 291 were also helpful since they gave us practice using different types of microcontrollers which were used in project 2.

5 Conclusion:

Our team was successfully able to create a coin-picking robot and controller. We used a master-slave communication system with microcontrollers from different families for each. (EFM8 for the robot and ATMEGA for the master). Our robot was able to detect and pick up coins while staying within the boundary in automatic mode. In manual mode was able to pick up coins, respond to the controller, and activate a speaker when a coin was under the inductor. We included all base functionality described in the project document, and added some bonus features: LEDs that signify the direction the robot is moving in manual mode, the ability to detect and pick up coins while turning, an extra button that makes the robot take coins from the container and place them back on the ground in a circle, and a “happy dance” that occurs after all the coins have been picked up in automatic mode. We also used a separate timer for each servo motor, allowing the two motors in the arm to move at the same time making the movement much smoother.

A problem we encountered while working on the project was managing the voltages and batteries on the robot, we had to make sure all the devices that could drain power were optocoupled. We initially had the electromagnet connected to the microcontroller voltage and ground and it caused a drain in voltage that interfered with the whole system. It was also challenging to fit all the components onto one breadboard, we had to bend up unused pins on one of the optocouplers to save space for other components. Our team spent a total of approximately 130 hours working on the project spread across a 19 day period.

BIBLIOGRAPHY

[1]“ATmega328P 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash DATASHEET.” https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

[2]“EFM8LB1 by Silicon Labs Datasheet | DigiKey,” *Digikey.com*, 2021. https://www.digikey.com/en/htmldatasheets/production/1934067/0/0/1/efm8lb1-datasheet

[3] J. Calvino-Fraga, Project 2 - Coin Picking Robot, University of British Columbia, 2025.

[4] J. Calvino-Fraga. ELEC 281. Lecture, “Coin Picking Robot.” Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, March. 14, 2025.

[5] J. Calvino-Fraga. ELEC 281. Lecture, “Microcomputer Interfacing using transistors” Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Mar. 09, 2024.

[6] J. Calvino-Fraga, Robot assembly, University of British Columbia, 2025.

[7] J. Calvino-Fraga, Coin picker assembly, University of British Columbia, 2025.

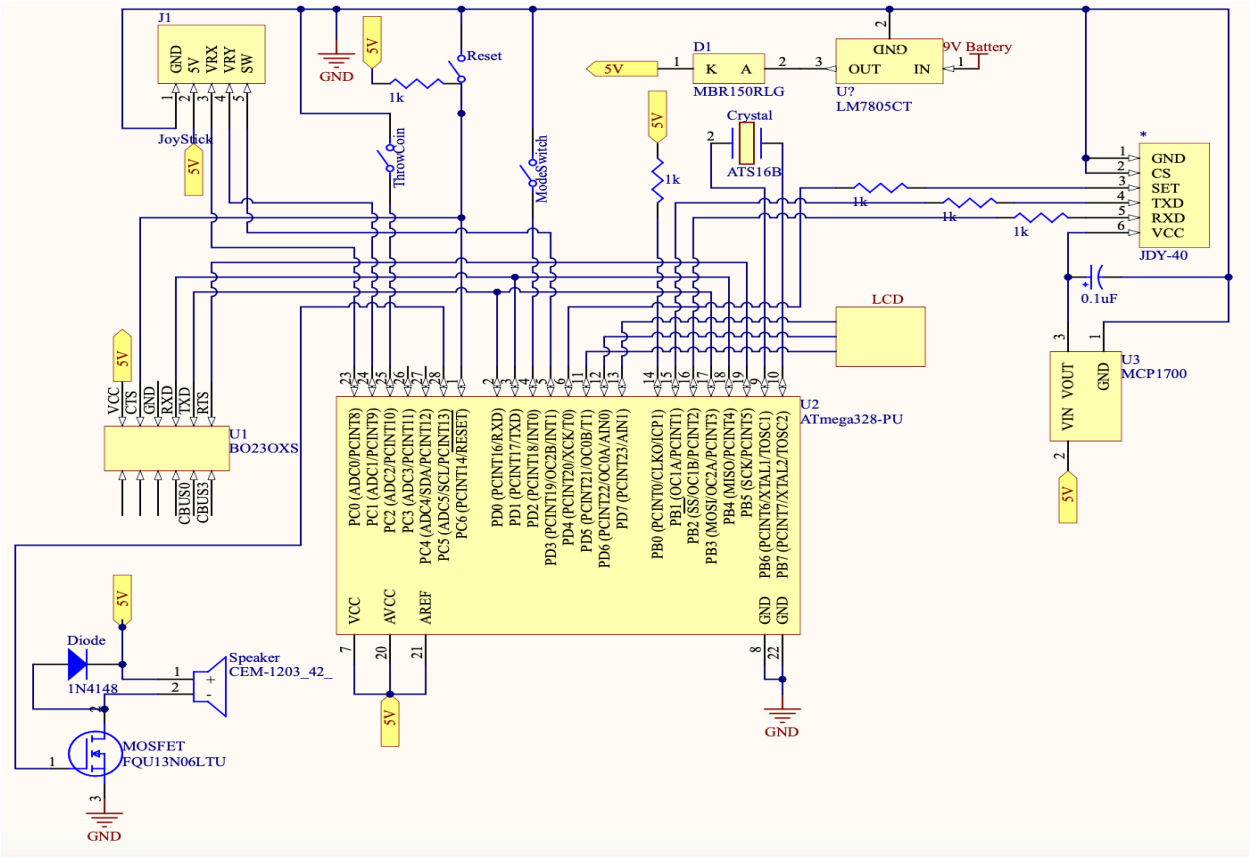
[8] J. Calvino-Fraga, Making an Electromagnet, University of British Columbia, 2025.

[9] J. Calvino-Fraga, ATMega328_JDY40_test, University of British Columbia, 2025.

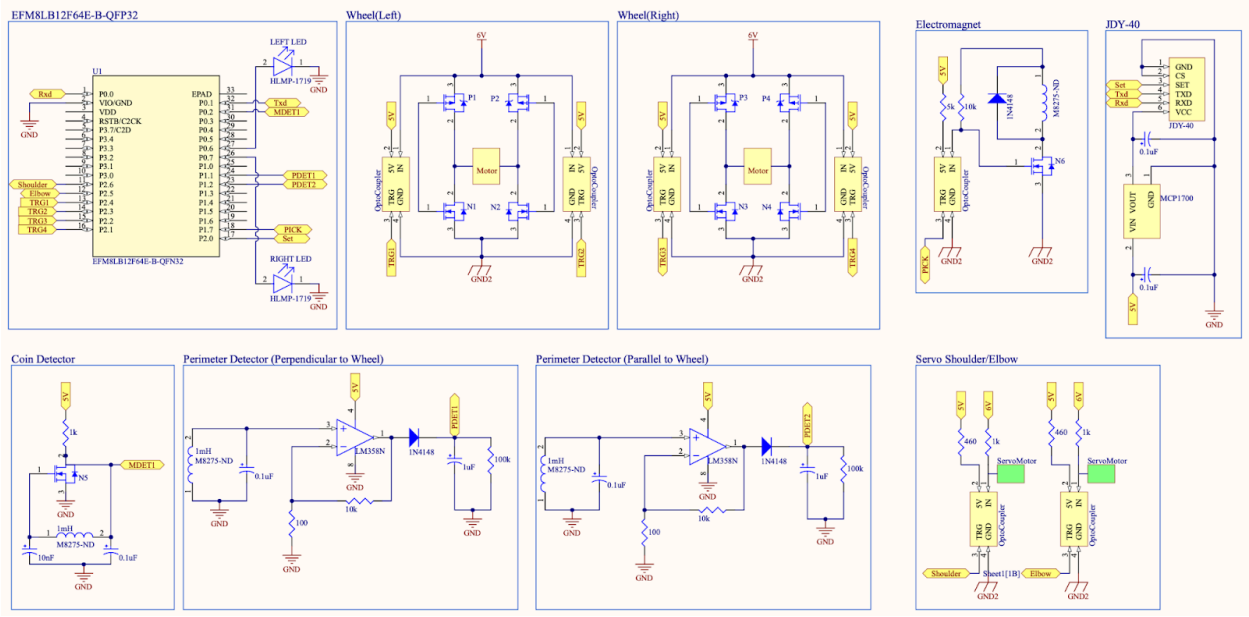
[10] J. Calvino-Fraga, EFM8LB1_JDY40_test, University of British Columbia, 2025.

Appendix:

Remote Circuit:



Robot Circuit:



Remote Circuit:

Slave Code:

```
#include <EFM8LB1.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


#define SYSCLK 72000000
#define BAUDRATE 115200L
#define MAGNET  P1_7
#define MOTOR_R1   P2_1
#define MOTOR_R2   P2_2
#define MOTOR_L1   P2_3
#define MOTOR_L2   P2_4
#define SHOULDER_SERVO   P2_5
#define ELBOW_SERVO   P2_6
#define DETECT_METAL P0_2
#define DETECT_Wire1 P1_1
#define DETECT_Wire2 P1_2
#define RELOAD_10MS (0x10000L-(SYSCLK/(12L*100L)))
#define SARCLK 18000000L
#define A 3800L

volatile unsigned int elbow_reload;
volatile unsigned int shoulder_reload;
volatile unsigned char pwm_state4=0;
volatile unsigned char pwm_state5=0;
volatile unsigned char count20ms;
volatile unsigned int overflow_count;
volatile unsigned int start_period;
unsigned int select;
unsigned int motor_r_select;
unsigned int motor_l_select;
unsigned int magnet_select;
unsigned int servo_select;
unsigned int test_servo_flag=1;
unsigned int magnet_detector_flag=1;
unsigned int perimeter_detector_flag=1;
unsigned int coins_picked=0;
float pulse_width;
float period, frequency;
unsigned int i;
unsigned int j;

idata char buff[20];
idata char buff3[20];

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
P0MDOUT |= 0b_1100_0000;
P2MDOUT |= 0b_0110_0000;
P0MDOUT |= 0x11; // Enable UART0 TX (P0.4) and UART1 TX (P0.0) as push-pull outputs

XBR0     = 0x01; // Enable UART0 on P0.4(TX) and P0.5(RX)
XBR1     = 0X00;
XBR2     = 0x41; // Enable crossbar and uart 1

// Configure Uart 0
#if (((SYSCLK/BAUDRATE)/(2L*12L))>0xFFL)
        #error Timer 0 reload value is incorrect because (SYSCLK/BAUDRATE)/(2L*12L) > 0xFF
#endif
SCON0 = 0x10;
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

// Initialize timer 5 for periodic interrupts
SFRPAGE=0x10;
TMR5CN0=0x00;
TMR5RL=RELOAD_10MS-0x10000L-(SYSCLK*0.5*1.0e-3)/12.0; // intialize the elbow servo to
0.5ms
```

```
	TMR5=0xffff;   // Set to reload immediately
	EIE2|=0b_0000_1000; // Enable Timer5 interrupts
	TR5=1;         // Start Timer5 (TMR5CN0 is bit addressable)

	// Initialize timer 4 for periodic interrupts
	SFRPAGE=0x10;
	TMR4CN0=0x00;
	TMR4RL=RELOAD_10MS-0x10000L-(SYSCLK*0.9*1.0e-3)/12.0; // intialize the shoulder servo to
0.9ms
	TMR4=0xffff;   // Set to reload immediately
	EIE2|=0b_0000_0100; // Enable Timer4 interrupts
	TR4=1;         // Start Timer4 (TMR4CN0 is bit addressable)

	EA=1;

	SFRPAGE=0x00;

	P2_0=1; // 'set' pin to 1 is normal operation mode.

	return 0;
}

void Timer4_ISR (void) interrupt INTERRUPT_TIMER4 // ISR for the elbow servo
{
	SFRPAGE=0x10;
	TF4H = 0; // Clear Timer4 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.
	switch (pwm_state4)
	{
		case 0:
			ELBOW_SERVO=1;
			TMR4RL=RELOAD_10MS;
			pwm_state4=1;
			count20ms++;
		break;
		case 1:
			ELBOW_SERVO=0;
			TMR4RL=RELOAD_10MS-elbow_reload;
			pwm_state4=2;
		break;
		default:
			ELBOW_SERVO=0;
			TMR4RL=elbow_reload;
			pwm_state4=0;
		break;
	}
}

void Timer5_ISR (void) interrupt INTERRUPT_TIMER5 // ISR for the shoulder servo
{
	SFRPAGE=0x10;
	TF5H = 0; // Clear Timer5 interrupt flag
	switch (pwm_state5)
	{
		case 0:
		SHOULDER_SERVO=1;
			TMR5RL=RELOAD_10MS;
			pwm_state5=1;
			count20ms++;
		break;
```

```
                case 1:
                SHOULDER_SERVO=0;
                        TMR5RL=RELOAD_10MS-shoulder_reload;
                        pwm_state5=2;
                break;
                default:
                SHOULDER_SERVO=0;
                        TMR5RL=shoulder_reload;
                        pwm_state5=0;
                break;
        }
}


// Uses Timer3 to delay <us> micro-seconds.
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
        }
        TMR3CN0 = 0 ;                   // Stop Timer3 and clear overflow flag
}

void TIMER0_Init(void)
{
        TMOD&=0b_1111_0000; // Set the bits of Timer/Counter 0 to zero
        TMOD|=0b_0000_0001; // Timer/Counter 0 used as a 16-bit timer
        TR0=0; // Stop Timer/Counter 0
}

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

void InitADC (void)
{
        SFRPAGE = 0x00;
        ADEN=0; // Disable ADC

        ADC0CN1=
                (0x2 << 6) | // 0x0: 10-bit, 0x1: 12-bit, 0x2: 14-bit
         (0x0 << 3) | // 0x0: No shift. 0x1: Shift right 1 bit. 0x2: Shift right 2 bits. 0x3:
Shift right 3 bits.
```

```
                (0x0 << 0) ; // Accumulate n conversions: 0x0: 1, 0x1:4, 0x2:8, 0x3:16, 0x4:32

        ADC0CF0=
            ((SYSCLK/SARCLK) << 3) | // SAR Clock Divider. Max is 18MHz. Fsarclk = (Fadcclk) /
(ADSC + 1)
                (0x0 << 2); // 0:SYSCLK ADCCLK = SYSCLK. 1:HFOSC0 ADCCLK = HFOSC0.

        ADC0CF1=
                (0 << 7)   | // 0: Disable low power mode. 1: Enable low power mode.
                (0x1E << 0); // Conversion Tracking Time. Tadtk = ADTK / (Fsarclk)

        ADC0CN0 =
                (0x0 << 7) | // ADEN. 0: Disable ADC0. 1: Enable ADC0.
                (0x0 << 6) | // IPOEN. 0: Keep ADC powered on when ADEN is 1. 1: Power down when
ADC is idle.
                (0x0 << 5) | // ADINT. Set by hardware upon completion of a data conversion. Must
be cleared by firmware.
                (0x0 << 4) | // ADBUSY. Writing 1 to this bit initiates an ADC conversion when
ADCM = 000. This bit should not be polled to indicate when a conversion is complete. Instead, the
ADINT bit should be used when polling for conversion completion.
                (0x0 << 3) | // ADWINT. Set by hardware when the contents of ADC0H:ADC0L fall
within the window specified by ADC0GTH:ADC0GTL and ADC0LTH:ADC0LTL. Can trigger an interrupt.
Must be cleared by firmware.
                (0x0 << 2) | // ADGN (Gain Control). 0x0: PGA gain=1. 0x1: PGA gain=0.75. 0x2: PGA
gain=0.5. 0x3: PGA gain=0.25.
                (0x0 << 0) ; // TEMPE. 0: Disable the Temperature Sensor. 1: Enable the
Temperature Sensor.

        ADC0CF2=
                (0x0 << 7) | // GNDSL. 0: reference is the GND pin. 1: reference is the AGND pin.
                (0x1 << 5) | // REFSL. 0x0: VREF pin (external or on-chip). 0x1: VDD pin. 0x2:
1.8V. 0x3: internal voltage reference.
                (0x1F << 0); // ADPWR. Power Up Delay Time. Tpwrtime = ((4 * (ADPWR + 1)) + 2) /
(Fadcclk)

        ADC0CN2 =
                (0x0 << 7) | // PACEN. 0x0: The ADC accumulator is over-written.  0x1: The ADC
accumulator adds to results.
                (0x0 << 0) ; // ADCM. 0x0: ADBUSY, 0x1: TIMER0, 0x2: TIMER2, 0x3: TIMER3, 0x4:
CNVSTR, 0x5: CEX5, 0x6: TIMER4, 0x7: TIMER5, 0x8: CLU0, 0x9: CLU1, 0xA: CLU2, 0xB: CLU3

        ADEN=1; // Enable ADC
}

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

        mask=1<<pin_num;

        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
        ADINT = 0;
        ADBUSY = 1;     // Convert voltage at the pin
        while (!ADINT); // Wait for conversion to complete
        return (ADC0);
}




void UART1_Init (unsigned long baudrate)
{
    SFRPAGE = 0x20;
        SMOD1 = 0x0C; // no parity, 8 data bits, 1 stop bit
        SCON1 = 0x10;
        SBCON1 =0x00;   // disable baud rate generator
        SBRL1 = 0x10000L-((SYSCLK/baudrate)/(12L*2L));
        TI1 = 1; // indicate ready for TX
        SBCON1 |= 0x40;   // enable baud rate generator
        SFRPAGE = 0x00;
}

void putchar1 (char c)
{
    SFRPAGE = 0x20;
        while (!TI1);
        TI1=0;
        SBUF1 = c;
        SFRPAGE = 0x00;
}

void sendstr1 (char * s)
{
        while(*s)
        {
                putchar1(*s);
                s++;
        }
}

char getchar1 (void)
{
        char c;
    SFRPAGE = 0x20;
        while (!RI1);
        RI1=0;
        // Clear Overrun and Parity error flags
        SCON1&=0b_0011_1111;
        c = SBUF1;
        SFRPAGE = 0x00;
```

```
        return (c);
}

char getchar1_with_timeout (void)
{
        char c;
        unsigned int timeout;
    SFRPAGE = 0x20;
    timeout=0;
        while (!RI1)
        {
                SFRPAGE = 0x00;
                Timer3us(20);
                SFRPAGE = 0x20;
                timeout++;
                if(timeout==25000)
                {
                        SFRPAGE = 0x00;
                        return ('\n'); // Timeout after half second
                }
        }
        RI1=0;
        // Clear Overrun and Parity error flags
        SCON1&=0b_0011_1111;
        c = SBUF1;
        SFRPAGE = 0x00;
        return (c);
}

void getstr1 (char * s, unsigned char n)
{
        char c;
        unsigned char cnt;

        cnt=0;
        while(1)
        {
                c=getchar1_with_timeout();
                if(c=='\n')
                {
                        *s=0;
                        return;
                }

                if (cnt<n)
                {
                        cnt++;
                        *s=c;
                        s++;
                }
                else
                {
                        *s=0;
                        return;
                }
        }
}

// RXU1 returns '1' if there is a byte available in the receive buffer of UART1
bit RXU1 (void)
{
        bit mybit;
```

```
    SFRPAGE = 0x20;
       mybit=RI1;
       SFRPAGE = 0x00;
       return mybit;
}

void waitms_or_RI1 (unsigned int ms)
{
       unsigned int j;
       unsigned char k;
       for(j=0; j<ms; j++)
       {
              for (k=0; k<4; k++)
              {
                     if(RXU1()) return;
                     Timer3us(250);
              }
       }
}

void SendATCommand (char * s)
{
       printf("Command: %s", s);
       P2_0=0; // 'set' pin to 0 is 'AT' mode.
       waitms(5);
       sendstr1(s);
       getstr1(buff, sizeof(buff)-1);
       waitms(10);
       P2_0=1; // 'set' pin to 1 is normal operation mode.
       printf("Response: %s\r\n", buff);
}

void ReceptionOff (void)
{
       P2_0=0; // 'set' pin to 0 is 'AT' mode.
       waitms(10);
       sendstr1("AT+DVID0000\r\n"); // Some unused id, so that we get nothing in RXD1.
       waitms(10);
       // Clear Overrun and Parity error flags
       SCON1&=0b_0011_1111;
       P2_0=1; // 'set' pin to 1 is normal operation mode.
}

void CheckConfiguration(void) {

       SendATCommand("AT+VER\r\n");
       SendATCommand("AT+BAUD\r\n");
       SendATCommand("AT+RFID\r\n");
       SendATCommand("AT+DVID\r\n");
       SendATCommand("AT+RFC\r\n");
       SendATCommand("AT+POWE\r\n");
       SendATCommand("AT+CLSS\r\n");

       // We should select an unique device ID.  The device ID can be a hex
       // number from 0x0000 to 0xFFFF.  In this case is set to 0xABBA
       SendATCommand("AT+DVIDDCAB\r\n");
       SendATCommand("AT+RFC117\r\n");
}

void PickCoin(void){
       P0_7=1;
       P0_6=1;
```

```
        shoulder_reload=0x10000L-(SYSCLK*1.6*1.0e-3)/12.0;
        elbow_reload=0x10000L-(SYSCLK*2.2*1.0e-3)/12.0;
        MAGNET=1;
        waitms(250);
        shoulder_reload=0x10000L-(SYSCLK*2.3*1.0e-3)/12.0;
        waitms(250);
        for(i=1; i<17; i++){
                elbow_reload=0x10000L-(SYSCLK*(2.2-(0.1*i))*1.0e-3)/12.0;
                waitms(25);
        }
        waitms(250);
        for(i=1;i<12;i++){
                shoulder_reload=0x10000L-(SYSCLK*(2.3-(0.1*i))*1.0e-3)/12.0;
                waitms(25);
        }
        elbow_reload=0x10000L-(SYSCLK*1*1.0e-3)/12.0;
        waitms(500);
        MAGNET=0;
        elbow_reload=0x10000L-(SYSCLK*0.5*1.0e-3)/12.0;
        P0_7=0;
        P0_6=0;
        waitms(50);
        coins_picked++;

}

void PickCoinManual(void){
        P0_7=1;
        P0_6=1;
        shoulder_reload=0x10000L-(SYSCLK*1.6*1.0e-3)/12.0;
        elbow_reload=0x10000L-(SYSCLK*2.2*1.0e-3)/12.0;
        MAGNET=1;
        waitms(250);
        shoulder_reload=0x10000L-(SYSCLK*2.3*1.0e-3)/12.0;
        waitms(250);
        for(i=1; i<17; i++){
                elbow_reload=0x10000L-(SYSCLK*(2.2-(0.1*i))*1.0e-3)/12.0;
                waitms(25);
        }
        waitms(250);
        for(i=1;i<12;i++){
                shoulder_reload=0x10000L-(SYSCLK*(2.3-(0.1*i))*1.0e-3)/12.0;
                waitms(25);
        }
        elbow_reload=0x10000L-(SYSCLK*1*1.0e-3)/12.0;
        waitms(500);
        MAGNET=0;
        elbow_reload=0x10000L-(SYSCLK*0.5*1.0e-3)/12.0;
        coins_picked++;
        P0_7=0;
        P0_6=0;
        waitms(50);
        CheckConfiguration();
}

void PlaceCoin(void){
        shoulder_reload=0x10000L-(SYSCLK*1.1*1.0e-3)/12.0;
        waitms(250);
        elbow_reload=0x10000L-(SYSCLK*1.8*1.0e-3)/12.0;
        MAGNET=1;
        waitms(250);
        shoulder_reload=0x10000L-(SYSCLK*0.9*1.0e-3)/12.0;
```

```
	waitms(500);
	shoulder_reload=0x10000L-(SYSCLK*1.1*1.0e-3)/12.0;
	waitms(1000);
	elbow_reload=0x10000L-(SYSCLK*1.6*1.0e-3)/12.0;
	waitms(250);
	elbow_reload=0x10000L-(SYSCLK*1.5*1.0e-3)/12.0;
	waitms(250);
	elbow_reload=0x10000L-(SYSCLK*1.4*1.0e-3)/12.0;
	waitms(250);
	elbow_reload=0x10000L-(SYSCLK*0.8*1.0e-3)/12.0;
	waitms(250);
	shoulder_reload=0x10000L-(SYSCLK*1.6*1.0e-3)/12.0;
	waitms(250);
	elbow_reload=0x10000L-(SYSCLK*1.7*1.0e-3)/12.0;
	waitms(250);
	MOTOR_R1=1;
	MOTOR_R2=0;
	MOTOR_L1=0;
	MOTOR_L2=1;
	waitms(1000);
	MOTOR_R1=0;
	MOTOR_R2=0;
	MOTOR_L1=0;
	MOTOR_L2=0;
	waitms(250);
	shoulder_reload=0x10000L-(SYSCLK*2.4*1.0e-3)/12.0;
	waitms(100);
	MAGNET=0;
	waitms(250);
	elbow_reload=0x10000L-(SYSCLK*0.5*1.0e-3)/12.0;
	waitms(250);

	shoulder_reload=0x10000L-(SYSCLK*1.1*1.0e-3)/12.0;
	CheckConfiguration();
}



float GetPeriod(void){
	// Reset the counter
	TL0=0;
	TH0=0;
	TF0=0;
	overflow_count=0;

	while(DETECT_METAL!=0); // Wait for the signal to be zero
	while(DETECT_METAL!=1); // Wait for the signal to be one
	TR0=1; // Start the timer
	while(DETECT_METAL!=0) // Wait for the signal to be zero
	{
		if(TF0==1) // Did the 16-bit timer overflow?
		{
			TF0=0;
			overflow_count++;
		}
	}
	while(DETECT_METAL!=1) // Wait for the signal to be one
	{
		if(TF0==1) // Did the 16-bit timer overflow?
		{
			TF0=0;
			overflow_count++;
```

```
                }
        }
        TR0=0; // Stop timer 0, the 24-bit number [overflow_count-TH0-TL0] has the period!
        period=(overflow_count*65536.0+TH0*256.0+TL0)*(12.0/SYSCLK);
        return period;
}
void InitServo(void){
        elbow_reload=0x10000L-(SYSCLK*0.5*1.0e-3)/12.0; // intialize the elbow servo to 0.5ms
        waitms(500);
        shoulder_reload=0x10000L-(SYSCLK*1.1*1.0e-3)/12.0; // intialize the shoulder servo to
1.1ms
}
void stop_moving(void){
        MOTOR_R1=0;
        MOTOR_R2=0;
        MOTOR_L1=0;
        MOTOR_L2=0;
}

void start_moving(void){
        MOTOR_R1=1;
        MOTOR_R2=0;
        MOTOR_L1=1;
        MOTOR_L2=0;
}

void reverse_moving(void){
        MOTOR_R1=0;
        MOTOR_R2=1;
        MOTOR_L1=0;
        MOTOR_L2=1;
}

void move_back_a_lil(void){
        MOTOR_R1=0;
        MOTOR_R2=1;
        MOTOR_L1=0;
        MOTOR_L2=1;
        waitms(200); // change this value
        stop_moving();
}



void WaitWhileDetectingMagnet(int time){ // to check for a magnet while turning
        for(j=0; j<time/50; j++){ // every 50ms check for coin and pick it up if its there
                DetectAndPickWhileSpinning();
                waitms(50);
        }
}

void move_back_a_lil_more(void){
        MOTOR_R1=0;
        MOTOR_R2=1;
        MOTOR_L1=0;
        MOTOR_L2=1;
        WaitWhileDetectingMagnet(400); // change this value
        stop_moving();
}

void SwitchDirection(int direction){
        move_back_a_lil_more();
```

```
if(direction==1){
        MOTOR_R1=1;
        MOTOR_R2=0;
        MOTOR_L1=0;
        MOTOR_L2=1;
        WaitWhileDetectingMagnet(600);
}
else if(direction==2){
        MOTOR_R1=1;
        MOTOR_R2=0;
        MOTOR_L1=0;
        MOTOR_L2=1;
        WaitWhileDetectingMagnet(550);
}
else if(direction==3){
        MOTOR_R1=1;
        MOTOR_R2=0;
        MOTOR_L1=0;
        MOTOR_L2=1;
        WaitWhileDetectingMagnet(750);
}
else if(direction==4){
        MOTOR_R1=1;
        MOTOR_R2=0;
        MOTOR_L1=0;
        MOTOR_L2=1;
        WaitWhileDetectingMagnet(1000);
}
else if(direction==5){
        MOTOR_R1=1;
        MOTOR_R2=0;
        MOTOR_L1=0;
        MOTOR_L2=1;
        WaitWhileDetectingMagnet(350);
}
else if(direction==6){
        MOTOR_R1=1;
        MOTOR_R2=0;
        MOTOR_L1=0;
        MOTOR_L2=1;
        WaitWhileDetectingMagnet(300);
}
else if(direction==7){
        MOTOR_R1=1;
        MOTOR_R2=0;
        MOTOR_L1=0;
        MOTOR_L2=1;
        WaitWhileDetectingMagnet(500);
}
else if(direction==8){
        MOTOR_R1=1;
        MOTOR_R2=0;
        MOTOR_L1=0;
        MOTOR_L2=1;
        WaitWhileDetectingMagnet(550);
}
else if(direction==9){
        MOTOR_R1=1;
        MOTOR_R2=0;
        MOTOR_L1=0;
        MOTOR_L2=1;
        WaitWhileDetectingMagnet(600);
```

```
	}
	else if(direction==10){
		MOTOR_R1=1;
		MOTOR_R2=0;
		MOTOR_L1=0;
		MOTOR_L2=1;
		WaitWhileDetectingMagnet(700);
	}
	start_moving();
}

unsigned int DetectPerimeter(void){
	//printf("%d, %d\n\r", ADC_at_Pin(QFP32_MUX_P1_1), ADC_at_Pin(QFP32_MUX_P1_2));
	if((ADC_at_Pin(QFP32_MUX_P1_1)>50) || (ADC_at_Pin(QFP32_MUX_P1_2)>50)){ // change
parameters here
		return 1;
	}
	else{
		return 0;
	}

}
void DetectAndPick(void){
	period=GetPeriod()*1000000.0; // multiply period by 10^6 cuz its kinda small

	if(start_period-period>0.0255){
		waitms(2);
		period=GetPeriod()*1000000.0; // multiply period by 10^6 cuz its kinda small

		if(start_period-period>0.0255){
				stop_moving(); // stop moving if metal is detected
				waitms(50);
				move_back_a_lil(); // move so coin is in the picking area
				waitms(50);
				PickCoin(); // pick coin if metal is detected
				waitms(50);
				start_moving(); // start moving again
			}
		}

}

void DetectAndPickWhileSpinning(void){
	period=GetPeriod()*1000000.0; // multiply period by 10^6 cuz its kinda small

	if(start_period-period>0.0255){
		waitms(2);
			period=GetPeriod()*1000000.0; // multiply period by 10^6 cuz its kinda
small
			if(start_period-period>0.0255){
				stop_moving(); // stop moving if metal is detected
				waitms(50);
				move_back_a_lil(); // move so coin is in the picking area
				waitms(50);
				PickCoin(); // pick coin if metal is detected
				waitms(50);
				start_moving(); // start moving again
			}
		}
}

void VictoryDance(void){
```

```
        for(i=0; i<10; i++){
                reverse_moving();
                waitms(100);
                start_moving();
                waitms(100);
        }
        stop_moving();
}

void main (void)
{
    unsigned int cnt=0;
    char c;
    char dirX;

    count20ms = 0;
       MAGNET = 0;
       P0_6=0;
       P0_7=0;
       stop_moving();

       TIMER0_Init();

       waitms(500);
       printf("\r\nEFM8LB12 JDY-40 Slave Test.\r\n");
       UART1_Init(9600);
       InitPinADC(1, 1); // Configure P1.1 as analog input
       InitPinADC(1, 2); // Configure P1.2 as analog input
       InitADC();
       InitServo();
    waitms(2000); // Wait a second to give PuTTy a chance to start

       printf("\x1b[2J\x1b[1;1H");

       ReceptionOff();

       // To check configuration
       SendATCommand("AT+VER\r\n");
       SendATCommand("AT+BAUD\r\n");
       SendATCommand("AT+RFID\r\n");
       SendATCommand("AT+DVID\r\n");
       SendATCommand("AT+RFC\r\n");
       SendATCommand("AT+POWE\r\n");
       SendATCommand("AT+CLSS\r\n");

       // We should select an unique device ID.  The device ID can be a hex
       // number from 0x0000 to 0xFFFF.  In this case is set to 0xABBA
       SendATCommand("AT+DVIDDCAB\r\n");
       SendATCommand("AT+RFC117\r\n");

       cnt=0;
       start_period=GetPeriod()*1000000.0;
       while(1)
       {
               if(RXU1()) // Something has arrived
               {
                       c=getchar1();

                       if(c=='!') // Master is sending message
                       {
                               getstr1(buff, sizeof(buff)-1);
                               if(strlen(buff)<=3)
```

```
{
//      printf("Master Says: %s\r\n", buff);

        dirX = buff[0];

        //printf("%c\n", dirX);



        if (dirX == 'w') {
        MOTOR_R1=1;
        MOTOR_R2=0;
        MOTOR_L1=1;
        MOTOR_L2=0;
        P0_6=0;
        P0_7=0;
        }

        if (dirX == 'a') {
        MOTOR_R1=1;
        MOTOR_R2=0;
        MOTOR_L1=0;
        MOTOR_L2=1;
        P0_6=1;
        P0_7=0;
        }

        if (dirX == 's') {
        MOTOR_R1=0;
        MOTOR_R2=1;
        MOTOR_L1=0;
        MOTOR_L2=1;
        P0_6=1;
        P0_7=1;
        }

        if (dirX == 'd') {
        MOTOR_R1=0;
        MOTOR_R2=1;
        MOTOR_L1=1;
        MOTOR_L2=0;
        P0_6=0;
        P0_7=1;
        }

        if (dirX == 'n') {
        MOTOR_R1=0;
        MOTOR_R2=0;
        MOTOR_L1=0;
        MOTOR_L2=0;
        P0_6=0;
        P0_7=0;
        }
        //code for pic coin
        if (dirX == 'f') {
        PickCoinManual();
        }
        if( dirX == 't'){
                PlaceCoin();
        }

        if(dirX == 'z'){ // automatic routine
```

```
                                        start_moving();
                                        waitms(5);
                                        DetectAndPick(); // check for coins and pick them up
                                        waitms(5);
                                        if(DetectPerimeter()){ // check if we reached the
perimeter
                                        SwitchDirection((rand() % 10) + 1); // random turn
                                        }
                                        if(coins_picked==20){ // if we picked up 20 coins,
stop moving
                                        coins_picked=0;
                                        VictoryDance();
                                        stop_moving();
                                        waitms(50);
                                        sprintf(buff3, "k");
                                        waitms(5); // The radio seems to need this delay...
                                        sendstr1(buff3);
                                        waitms(250);
                                        CheckConfiguration();
                                        }
                                        waitms(5);
                                }

                        }

                }

                if(c=='@') // Master wants slave data
                {
                        frequency = 1/ GetPeriod();
                        sprintf(buff3, "%f\n", frequency);
                        cnt++;
                        waitms(5); // The radio seems to need this delay...
                        sendstr1(buff3);

                }


        }
    }
}
```

Master Code:

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <util/delay.h>
#include "usart.h"
#include "Software_UART.h"

/* Pinout for DIP28 ATMega328P:

                          -------
     (PCINT14/RESET) PC6 -|1    28|- PC5 (ADC5/SCL/PCINT13)
       (PCINT16/RXD) PD0 -|2    27|- PC4 (ADC4/SDA/PCINT12)
       (PCINT17/TXD) PD1 -|3    26|- PC3 (ADC3/PCINT11)
      (PCINT18/INT0) PD2 -|4    25|- PC2 (ADC2/PCINT10)
 (PCINT19/OC2B/INT1) PD3 -|5    24|- PC1 (ADC1/PCINT9)
```

```
    (PCINT20/XCK/T0) PD4 -|6    23|- PC0 (ADC0/PCINT8)
                     VCC -|7    22|- GND
                     GND -|8    21|- AREF
(PCINT6/XTAL1/TOSC1) PB6 -|9    20|- AVCC
(PCINT7/XTAL2/TOSC2) PB7 -|10   19|- PB5 (SCK/PCINT5)
   (PCINT21/OC0B/T1) PD5 -|11   18|- PB4 (MISO/PCINT4)
 (PCINT22/OC0A/AIN0) PD6 -|12   17|- PB3 (MOSI/OC2A/PCINT3)
      (PCINT23/AIN1) PD7 -|13   16|- PB2 (SS/OC1B/PCINT2)
  (PCINT0/CLKO/ICP1) PB0 -|14   15|- PB1 (OC1A/PCINT1)
                          -------
*/
/*
stagnant x 330
stagnant y 390
y 678 left x 330
y 21 left x 330
x 670 y 390 down
x 0 y 390 up

ntn 39473
5c 41095
10c 40540
25c 41379(maybe)-41666
1$ 41379-41095
2$  417

*/
#define DEF_FREQ 15000L
#include <stdint.h>

#include "lcd.h"
#include "lcd.c"
/* Pinout remains the same as provided */

#define DEF_FREQ 15000L
#define PRESCALER 64
#define OCR2_RELOAD ((F_CPU/(PRESCALER*2*DEF_FREQ))+1)

volatile uint8_t reload; // 8-bit reload value for Timer2

// Timer2 Compare A Interrupt Service Routine



// Timer2 Compare A Interrupt Service Routine
ISR(TIMER2_COMPA_vect)
{
//      TIMSK2 = (1 << OCIE2A); // i.e. TIMSK2 = 0x02
    OCR2A += reload; // Update OCR2A with 8-bit addition
    PORTC ^= 0b00100000; // Toggle PB0 and PB1
}

// 'Timer 1 output compare A' Interrupt Service Routine


void SendATCommand (char * s)
{
        char buff[40];
        printf("Command: %s", s);
        PORTD &= ~(BIT4); // 'set' pin to 0 is 'AT' mode.
        _delay_ms(10);
        SendString1(s);
```

```
        GetString1(buff, 40);
        PORTD |= BIT4; // 'set' pin to 1 is normal operation mode.
        _delay_ms(10);
        printf("Response: %s\r\n", buff);
}

void ReceptionOff (void)
{
        PORTD &= ~(BIT4); // 'set' pin to 0 is 'AT' mode.
        _delay_ms(10);
        SendString1("AT+DVID0000\r\n"); // Some unused id, so that we get nothing from the radio.
        _delay_ms(10);
        PORTD |= BIT4; // 'set' pin to 1 is normal operation mode.
}
void initUART(void)
{
        // Not necessary; initialize anyway
        DDRD |= _BV(PD1);
        DDRD &= ~_BV(PD0);

        // Set baud rate; lower byte and top nibble
        UBRR0H = ((_UBRR) & 0xF00);
        UBRR0L = (uint8_t) ((_UBRR) & 0xFF);

        TX_START();
        RX_START();

        // Set frame format = 8-N-1
        UCSR0C = (_DATA << UCSZ00);
        //UCSR0A bits: RXC0 TXC0 UDRE0 FE0 DOR0 UPE0 U2X0 MPCM0
        UCSR0A|=0x02; // U2X0 double speed
}

uint8_t getByte(void)
{
        // Check to see if something was received
        while (!(UCSR0A & _BV(RXC0)));
        return (uint8_t) UDR0;
}

void putByte(unsigned char data)
{
        // Stay here until data buffer is empty
        while (!(UCSR0A & _BV(UDRE0)));
        UDR0 = (unsigned char) data;
}

void writeString(char *str)
{
        while (*str != '\0')
        {
                putByte(*str);
                ++str;
        }
}

// delay functions
unsigned int cnt = 0;

void wait_1ms(void)
{
        unsigned int saved_TCNT1;
```

```
        saved_TCNT1=TCNT1;

        while((TCNT1-saved_TCNT1)<(F_CPU/1000L)); // Wait for 1 ms to pass
}

void waitms(int ms)
{
        while(ms--) wait_1ms();
}

void adc_init(void)
{
    ADMUX = (1<<REFS0); //Sets to 10 bit ADC, using AVCC as reference voltage
    ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}

uint16_t adc_read(int channel)
{
    channel &= 0x7;
    ADMUX = (ADMUX & 0xf8)|channel;

    ADCSRA |= (1<<ADSC);

    while(ADCSRA & (1<<ADSC)); //as long as ADSC pin is 1 just wait.

    return (ADCW);
}

void PrintNumber(long int N, int Base, int digits)
{
        char HexDigit[]="0123456789ABCDEF";
        int j;
        #define NBITS 32
        char buff[NBITS+1];
        buff[NBITS]=0;

        j=NBITS-1;
        while ( (N>0) | (digits>0) )
        {
                buff[j--]=HexDigit[N%Base];
                N/=Base;
                if(digits!=0) digits--;
        }
        usart_pstr(&buff[j+1]);
}

void Configure_Pins(void)
{
        DDRB|=0b00000001; // PB0 is output.

        DDRB  &= 0b11111101; // Configure PB1 as input
        PORTB |= 0b00000010; // Activate pull-up in PB1

        DDRD  &= 0b11111011; // Configure PB1 as input
        PORTD |= 0b00000100; // Activate pull-up in PB1

        DDRD|=0b11111000; //  PD3, PD4, PD5, PD6, and PD7 are outputs.
        PORTD |= 0x02; // PD2 as speaker output

        DDRB  &= 0b11111011; // Configure PB2 as input
        PORTB |= 0b00000100; // Activate pull-up in PB2
```

```
        DDRC  &= 0b11111101; // PC1 as input
        PORTC |= 0b00000010; // pullup in PC1

        DDRB|=0b00000001; // PB0 is output.
        DDRD|=0b11110000; // PD3, PD4, PD5, PD6, and PD7 are outputs.
//      DDRC|=0b00011000; // PD3, PD4, PD5, PD6, and PD7 are outputs.

        DDRC|=0b00111000;

                // Configure PB1 for input.  Information here:
        // http://www.elecrom.com/avr-tutorial-2-avr-input-output/
        DDRD  &= 0b11111011;
        PORTD |= 0b00000100;
//      DDRD  &= 0b11110111;
//      PORTD |= 0b00001000;


    PORTC |= 0b00100000; // Initialize PB0 high, PB1 low

}
void ConfigurePins(void){
        SendATCommand("AT+VER\r\n");
        SendATCommand("AT+BAUD\r\n");
        SendATCommand("AT+RFID\r\n");
        SendATCommand("AT+DVID\r\n");
        SendATCommand("AT+RFC\r\n");
        SendATCommand("AT+POWE\r\n");
        SendATCommand("AT+CLSS\r\n");

        // We should select an unique device ID.  The device ID can be a hex
        // number from 0x0000 to 0xFFFF.  In this case is set to 0xABBA
        SendATCommand("AT+DVIDDCAB\r\n");
        SendATCommand("AT+RFC117\r\n");

}



void main(void)
{

        unsigned int adcX, adcY;
        int offsetX, offsetY;
        unsigned long int vX, vY;

        char buff[80];


        char sendX;
        char sendY;
        char radbuff[80];
    int timeout_cnt=0;
    int cont1=0, cont2=100;

        float frequency = 0;
        float period;
        int check = 0;
        int pickcoin = 0;
        char buffc[32];
        //LCD_4BIT();
```

```
        TCCR2B |= _BV(CS22) | _BV(CS21); // Prescaler 64
    TIMSK2 |= _BV(OCIE2A); // Enable Timer2 Compare A interrupt

        sei();
        usart_init (); // configure the usart and baudrate

        adc_init();
        Configure_Pins(); // configure pins as inputs & outputs
        LCD_4BIT(); // initialize LCD
        Init_Software_Uart(); // Configure the sorftware UART

        // Turn on timer with no prescaler on the clock.  We use it for delays and to measure
period.
        TCCR1B |= _BV(CS10); // Check page 110 of ATmega328P datasheet
        TIMSK1 |= _BV(OCIE1A); // output compare match interrupt for register A

        // TIMSK2 = 0;

         char buffspeak[32];
    unsigned long newF;

    reload = OCR2_RELOAD;

  //  DDRC = 0b00110000; // Set PB0 and PB1 as outputs
    PORTD |= 0b0001000; // Initialize PB0 high, PB1 low


    _delay_ms(500);


        waitms(500); // Wait for putty to start   // configure the hardware usart and baudrate

        _delay_ms(500); // Give putty a chance to start before we send information...
        printf("\r\nATMega328 JDY-40 Master Test\r\n");

        ReceptionOff();

        // To check configuration
        ConfigurePins();
//      reload=(F_CPU/(2L*100))+1;
//      LCDprint("hello world",2,1);

        offsetX=0;
        offsetY=0;

        while (1) // forever loop
        {


                if((PIND&0b00000100)==0){
      check = !check;
    }

          newF = atol(buff)/10;
          period = (1.0/newF)*1000000;
          printf("%f",period);

       if(  newF > 4300){
     TIMSK2 |= _BV(OCIE2A);

      }
     if(  newF < 4300 ){
```

```
TIMSK2 = 0;

 }

   if(newF > (F_CPU / (PRESCALER * 2 * 1)))
   {

   }
   if(newF > 0)
   {
       uint32_t calc_reload = (F_CPU / (PRESCALER * 2UL * newF)) + 1;
       if (calc_reload > 255)
       {
           calc_reload = 255;

       }
       reload = (uint8_t)calc_reload;

  }



         adcX=adc_read(0);
         adcX=adcX-offsetX;
         vX=(adcX*5000L)/1023L;


         adcY=adc_read(1);
         adcY=adcY-offsetY;
         vY=(adcY*5000L)/1023L;

          pickcoin = (PIND&0b00001000)?1:0;
         if(adcX <20){
                sendX = 'w';
         }
         else if(adcX >650){
                sendX = 's';
         }

         else if(adcY < 41){
                sendX = 'd';
         }
         else if(adcY >668){
                sendX = 'a';
         }
         else{
         if(pickcoin == 0){
   sendX ='f';
  // waitms(1250);

  }
  else{
         sendX = 'n';
         }
         }

         if(check){
         sendX = 'z';

         }
         if(buff[0] == 'k'){
         check = 0;
```

```
                ConfigurePins();
                }



        sprintf(buff, "%c\n", sendX); // Construct a test message
                SendByte1('!'); // Send a message to the slave. First send the 'attention'
character which is '!'
                // Wait a bit so the slave has a chance to get ready
                _delay_ms(5); // This may need adjustment depending on how busy is the slave
                SendString1(buff); // Send the test message

                if(++cont1>200) cont1=0; // Increment test counters for next message
                if(++cont2>200) cont2=0;

                _delay_ms(5); // This may need adjustment depending on how busy is the slave

                SendByte1('@'); // Request a message from the slave
                if(  newF > 4300){
          TIMSK2 |= _BV(OCIE2A);

       }
         if(  newF < 4300){
         TIMSK2 = 0;

       }

                timeout_cnt=0;
                while(1)
                {
                        if(RXD_FLAG) break; // Something has arrived
                        if(++timeout_cnt>250) break; // Wait up to 15ms for the repply
                        _delay_us(100); // 100us*250=25ms
                }

                if(RXD_FLAG) // Something has arrived from the slave
                {


                        GetString1_timeout(buff, sizeof(buff)-1);
                        if(strlen(buff)<=80) // Check for valid message size (5 characters)
                        {
                                printf("Slave says: %s\r\n", buff);

                                LCDprint(buff,2,1);
                        //      LCDprint("MAGNET DETECTOR",1,1);

                        }
                        else
                        {
                                printf("*** BAD MESSAGE ***: %s\r\n", 1/newF);
                        }
                }
                else // Timed out waiting for reply
                {
                        printf("NO RESPONSE\r\n", buff);
                }

                _delay_ms(5);  // Set the information interchange pace: communicate about every
50ms
        }
```

```
}
```