GRAND VALLEY STATE UNIVERSITY
SCHOOL OF ENGINEERING

Project Design Document

Project:
Gibson-Millwood Marble Sorter

Prepared By:
Joe Gibson – gibsjose@mail.gvsu.edu
Jesse Millwood – millwooj@mail.gvsu.edu

Supervisor: Dr. Robert Bossemeyer
EGR 326 Term Project
Fall Term 2013

Contents

Abstract (or Executive Summary)....3
1.0Introduction and Design Background....4
2.0Requirements and Functional Specifications....4
3.0System Architecture....4
3.1Hardware Specifications....4
3.1.1Hardware Design....4
3.1.2Hardware Implementation....4
3.2Software Specifications....4
3.2.1Software Design....4
3.2.2Software Implementation....5
4.0Verification....5
4.1Hardware verification....5
4.2Software verification....5
5.0Validation....6
6.0Conclusions....6
7.0Project Budget and Schedule....6
7.1Budget....6
7.2Schedule....6
8.0Revision History....6
Appendix A – Electrical Schematics....7
Appendix B – Bill of Materials....8
Appendix C – Detailed Mechanical Drawings....9
Appendix D – Source Code....10

Executive Summary

The objective of this project is to construct a standalone marble sorting device. The only interface to the outside world that this device requires is 9V DC power and human initiation. This device can differentiate between black marbles and white marbles. Once the color of the marble is identified the marbles are diverted into bins of the respective color. The marbles are diverted without human interference. This is accomplished by the following steps: The marble rolls into the opening of the wishbone shaped diverters, the marble is scanned by the infrared color sensors, the microcontroller decides which marble is which color, the microcontroller sends the correct controlling PWM signal to the servos, the servos rotate the diverters, the marbles are diverted into the correct bins. There are two diverters and two servos. This allows for near simultaneous sorting of two marbles. The operator can view the status of the sorting operation via a 20x4 character LCD screen. If the operator wishes to start, stop, or reset the sorting process, he or she can do so with the push of the designated button. Further status indication is available to the operator via an LED that will be one of three colors: Error Free Sorting State (green), Powering Down State (yellow), or End Of Sorting State (red). This device tracks the marbles sorted and stores this information for use between power cycles. This marble sorting device is easy to use, fast, and efficient.

1.0 Introduction and Design Background

A marble sorter was to be designed to efficiently sort black and white marbles into separate bins.

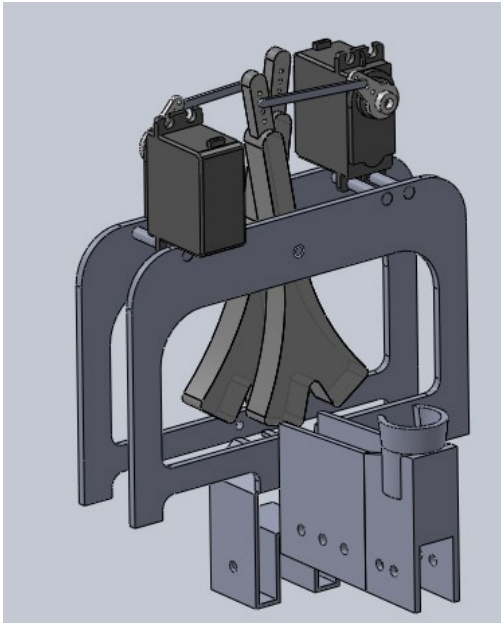


Figure 1: Sorting Machine

The sorting machine can be seen in Figure 1. The objectives of the design were to store 25 or more marbles in a hopper and to then sort the marbles using infrared photodiodes and infrared phototransistors. The method chosen for this project was to implement a lambda-shaped diverter system, using two diverters, to sort two marbles at one time. Marbles are gravity-fed from the hopper in one continuous line of marbles into the sorting chamber, where they are diverted to either one side or the other based on their color. Non-continuous servo motors are used to rotate the diverters about a central axis, knocking the marbles off the track on which they are fed into the diverting chamber. The servo motors are connected to the top of the diverters via a linkage arm. Both the diverters and the linkage are to be 3-D printed using the MakerBot printers. The servo motors are controlled by outputting a PWM signal from a microcontroller. The ATmega328P microcontroller was used, and the PWM signal is diverted through a four-position switch, which acts as a demultiplexer to select which servo the signal is routed to. The ATmega328P is an 8-bit microcontroller, typically used on the Arduino platform. However, in the proposed design, the ATmega328P was implemented from the ground up, with the supporting

electronics built into the manufactured PCB. Finally, the infrared diode and phototransistors were chosen as sensors, since noise from visible light was reduced. The infrared phototransistor outputs a voltage at V_{CE} inversely proportional to the amount of light sensed by the transistor. This characteristic, coupled with the visible light rejection, made the infrared diodes and phototransistors an ideal choice.

2.0 Requirements and Functional Specifications

The marble sorting system is to be designed such that it will sort black and white marbles, fed through a hopper, into bins corresponding to their color. The system will use the ATmega328P microcontroller to interface with servo motors, infrared sensors , and an analog SP3T IC switch. The goal of the system is to sort marbles accurately and rapidly.

Constraints

1. Purchased items should not exceed $50 USD.
2. The sorting system will be fed 25 marbles via a hopper
3. The sorting system must recognize what color each marble is and displace it to a bin containing other marbles of the same color
4. The operator must be able to start, stop, and reset sorting via a button interface
5. The operator must be able to see the progress and status of the sorting via an LCD screen
6. Sorting information must be stored by the sorting machine that will be accessible even after a power cycle.
7. The operator will be notified of successful sorting, errors, or warnings via LEDs of appropriate colors.
8. The system will be powered from an AC-DC “wall wart” power supply.

3.0 System Architecture

3.1 Hardware Specifications

This marble sorting machine consists mainly of two diverters and two servos controlled by a single microcontroller. The PWM signal is switched between the two servos via a 3PST analog switch IC. The block diagram in Figure 2 shows how the hardware components interact with one another.

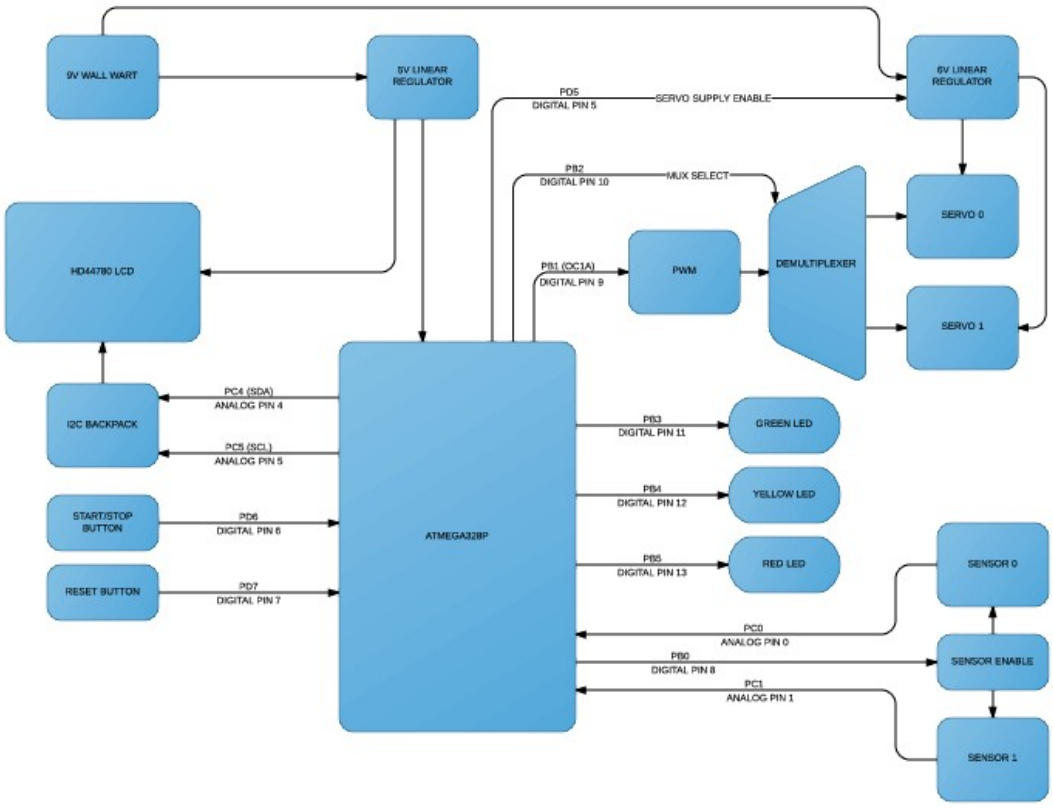


Figure 2: Hardware Block Diagram

3.1.1 Hardware Design

The final sorting machine consist of two Axle Mounts that support the axles for the Diverters, the rods for the Servos, and the marble track. The servos are controlled via a pwm signal that is switched between one servo and the other via a SP3T analog switch IC. This design will simultaneously sort two marbles. The operator will be able to view the system status via a 20x4 character LCD display that communicates with the microcontroller via i2c protocol.

Alternatives were briefly explored and were fond to be unfit for the requirements at hand. One alternative was using a stepper motor instead of servos. These motors are much more expensive than the servos and servos meet the requirements to move the marbles, so needless to say, servos were used. The diverters were decided upon because of their simplicity. One other design that was considered to move the marbles was a room that the marble would roll into that was moved from one side to the other, much like the opening of the diverter. However the room would be moved by a linear actuator. The downfall of this design was that common linear actuators do not have the travel distance needed to move a marble far enough off of the track and into a bin. The analog switch IC was chosen because two PWM signals are required to control the two servos. The other PWM pins on the ATmega328P are not fast enough to properly control the servos in this case. The alternative to the switch was changing the crystal on the ATmega328P. This was decided against because other code snippets and libraries that may be used may require a certain clock frequency buried deep in the code. This was not a hunt that either team member wanted to embark on.

3.1.2 Hardware Implementation

The sorting machine consists of the following main hardware components:

1. Axle Mounts

The Axle Mounts are really the frame of the sorting machine. They hold the rods that the servos are mounted to, the track, and the axle that the diverters rotate about. These Axle Mounts were cut out of clear 1/8th inch thick plastic on the Haas CNC mills. There are two axle mounts one is designed so that the marbles are able to roll on to the track. The other is designed so that the marbles stop at the end of the track and can then be evaluated and sorted. One of the Axle Mounts that allow the marble onto the track is shown in Figure 3.

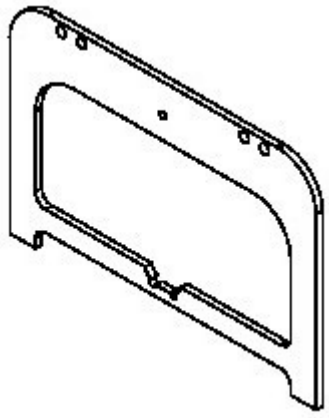


Figure 3: One of the Axle Mounts

2. Diverters

The Diverters are wishbone shaped objets that pivot around an axle that is mounted between two Axle Mounts. The Diverter is swung one way or the other via linkage to a nearby servo. When the Diverter is swung from one side to the other a marble will be diverted off of the track to one side or the other. These Diverters were constructed on a MakerBot 3d printer. A drawing of one can be seen in Figure 4.

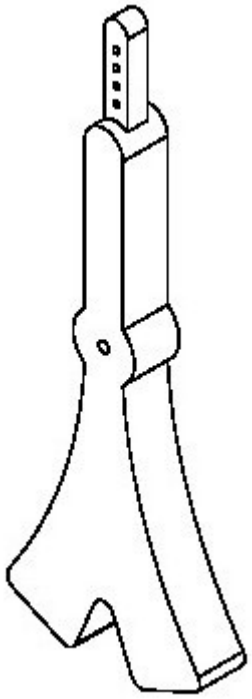


Figure 4:
Diverter Drawing

3. Servos

The Hi-Tec 311 PWM controlled servos were used to rotate the Diverters. The servos will receive power and the controlling PWM signal from the Microcontroller Board. There are designated pads on the Mircocontroller Board for both of these. They are designated by silk screen labels.

4. Microcontroller Board

The Microcontroller Board layout can be seen in Figure 5. This board consists of the ATmega328P, a SP3T analog IC, power regulation, serial communication converters and pads to interface to off board sensors. The ATmega328P will read the analog value from the infrared sensors and decide which color each marble is. It will then produce a PWM signal and select line signal to divert the PWM signal to the correct servo. The select lines will then be changed and the other servo will receive a PWM signal that will move the marble off of the track and into the correct bin.

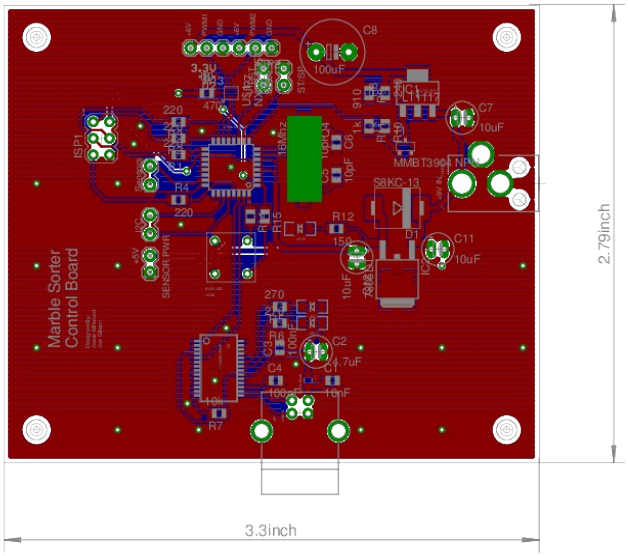


Figure 5: Microcontroller Board

The power regulation circuitry on this board regulates the 9V switched input to 6V for the Servos and 5V for the other IC's on the board. The power regulation is achieved by using two low dropout linear regulators.

5. Sensor Board

The sensor board will be built on peg board, a pcb board will not be manufactured separately. The sensor board will sit under the marble track and will interface to the Microcontroller Board via ribbon cable, sending signal and receiving power. The ribbon cable will be soldered to the designated pads on the Microcontroller Board. The sensors on this board will be the infrared LEDs and the infrared transistors. The peg board that this will be built on offers much more flexibility than a specially manufactured pcb board. The sensor board schematic can be seen in Figure 6.

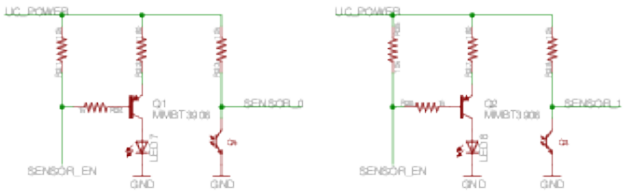


Figure 6: Color Sensing Circuit Off Board

6. LCD Screen

The LCD module that was used is a 20x4 character SainSmart module. This module has a i2c backpack that interfaces to the Microcontroller Board. This was used because the i2c protocol only uses two wires and microcontroller resources were needed elsewhere.

7. Hopper Track Section

The track that feeds the sorting mechanism is split into two sections. The first part interfaces with a funnel to hold all of the marbles. Figure 7 shows the section of track that interfaces with the funnel and the intermediate section of track that then interfaces with the sorting mechanism.

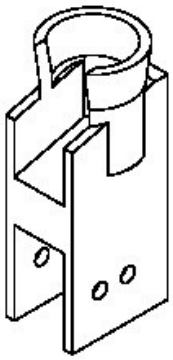


Figure 7: Hopper Track Section

8. Intermediate Track Section

The second section of track is the intermediate section between the hopper track section and the sorting mechanism. This track section is shown in Figure 8. Both track sections have a slight slope to aid in feeding the sorting mechanism.

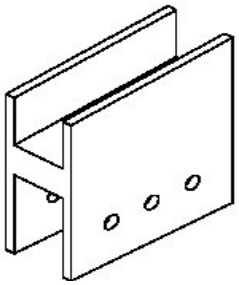


Figure 8: Intermediate Track Section

9. Sensor Enclosure

The sensor enclosure, shown in Figure 9, was mounted in between the two axle mounts. This served as a staging area for the marbles to be diverted from and as an enclosure to house the infrared sensors. The sensors were housed beneath the marbles that rolled in from the track sections.

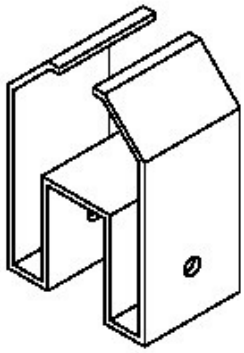


Figure 9: Sensor Enclosure

3.2 Software Specifications

The top level UML diagram in Figure 10 shows the top-level software approach. The sorter enters into an initialization state, followed by the idle state in which the sorter waits for user input to begin sorting, recall previous information, or reset the previous information. This simple four state approach minimizes errors associated with entering/exiting states, or being in an invalid state.

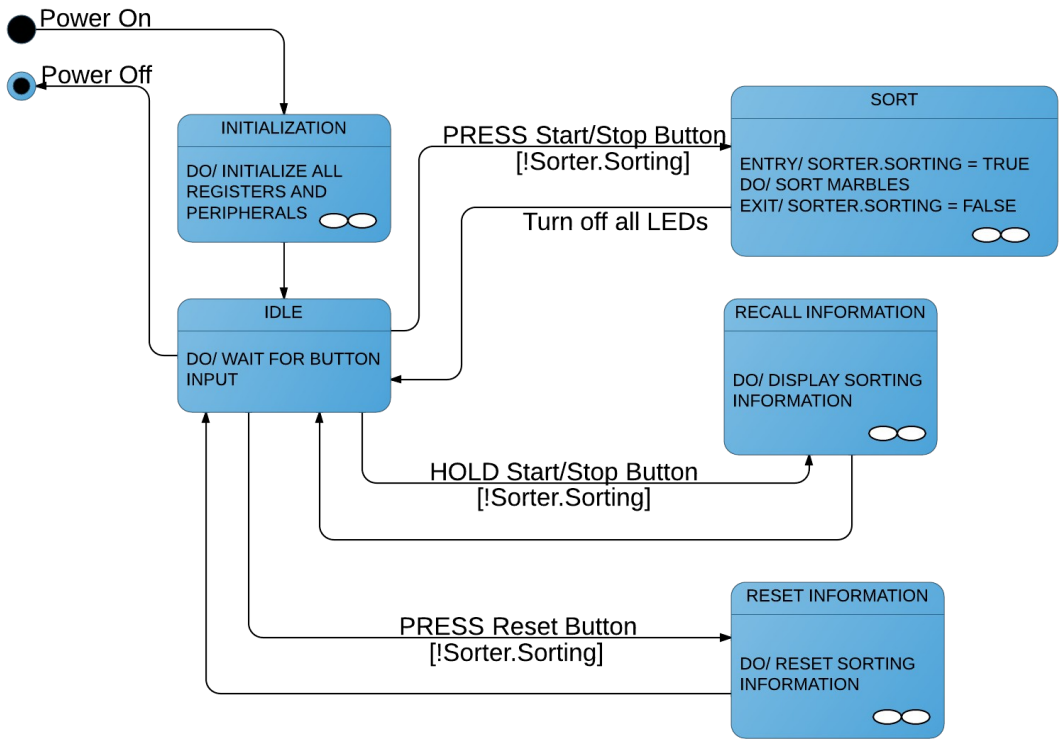


Figure 10: Top Level UML

3.2.1 Software Design

The initialization state performs the initialization of all the peripherals, including the timers, ADC, watchdog timer, USART, LCD, sensors, servos, etc. The initialization state is shown below in **Figure 11**, and the code implementing each of the functions is shown in the appendix. The marble, servo and sorter classes are initialized by their own constructor methods, which are found in their respective classes, also located in the appendix.

```
/*************************************************************************/
/* Setup                                                                 */
/*************************************************************************/
void setup(void)
{
    //Reset WDT right away
    wdt_reset();

    //Initialize everything
    InitPortDirections();
    InitTimers();
    InitADC();
    InitWDT();
    InitEEPROM();
    InitSorter();
    InitLCD();

    //Enable global interrupts
    sei();
}
```

Figure 11: Setup/Initialization State

The initialization state also prints the project name and version number to the screen and shows a loading bar animation, and then sets the sorter state to the idle state for the following loop.

The idle state simply waits for user input, continuously checking for and debouncing button presses, and polling flags to know which state to transfer into. A sample of entering into a state from the idle state is shown below in Figure 12.

```
/*********************/
/* Reset Information */
/*********************/
if((sorter.GetResetButtonAction() == Press) && (sorter.State == IdleState))
{
    sorter.ButtonActionCompleted();
    sorter.State = ResetState;

    ClearLine(LINE_4);
    lcd.setCursor(0, LINE_4);
    lcd.print("Reset");

    for(int i = 0; i < 3; i++)
    {
        _delay_ms(200);
        lcd.print(".");
    }

    sorter.SecondsElapsed = 0;
    sorter.MinutesElapsed = 0;
```

Figure 12: Sample of code showing requirements to enter a given state

Button debouncing and state control is done in a fairly simple yet elegant manner. There are two variables and one flag associated with the buttons -- *ResetButtonAction*, *StartStopButtonAction*, and *ButtonActionReady*. The two button action variables are of type *T_ButtonAction*, which represents a press, hold or no action on a button. The *ButtonActionReady* flag was used to determine when a new action was ready to be processed, which was checked automatically when calling either *Sorter::GetResetButtonAction()* and *Sorter::GetStartStopButtonAction()* from the sorter class. Only if a valid button action was found on a button (press or hold) and the *ButtonActionReady* flag was true did the button get processed as valid input. Furthermore, the buttons were debounced by using Timer 2 in CTC mode with a 1ms output compare interrupt. Every 1ms the state of both buttons was checked, and only if the buttons remained pressed for the entire duration of a press or hold was the state of the button changed to represent the newest action. Both buttons were not allowed to have actions present at the same time, so whichever button achieved a valid action first was processed first. Whenever an action was processed, *Sorter::ButtonActionCompleted()* was called, allowing for the next button action to be ready for processing.

The sort state continuously senses and sorts new marbles until either the stop button is pressed or the watchdog timer senses a timeout. The sort state also pushes an update of all the count information and updates the LCD with the elapsed time and marble counts. The sort state also lights the LED a color corresponding to whether it is still sorting (green), has been halted by the user (yellow), or has timed out/has encountered an error (red/blinking

red). The code for the *Sorter::SetLEDColor()* routine and *T_Color* enumeration can be found in the appendix. Some of the code associated with the sort state is shown below in **Figure 13.**

```
/*****************************************************************************/
/* Perform one sorting cycle                                                 */
/*****************************************************************************/
T_ErrorCode Sort(void)
{
    /*********************************/
    /* DEBUG: ONLY SORTING ONE MARBLE */
    /*********************************/
    //Declare error codes
    T_ErrorCode errorCodeChannelZero = ERR_NO_ERROR;
    //T_ErrorCode errorCodeChannelOne = ERR_NO_ERROR;

    //Check sensor 0
    errorCodeChannelZero = CheckSensorOnChannel(CHANNEL_0, MarbleZero);

    //Check sensor 1
    //errorCodeChannelOne = CheckSensorOnChannel(CHANNEL_1, MarbleOne);

    //Update counts
    UpdateCount(MarbleZero.GetMarbleType());
    //UpdateCount(MarbleOne.GetMarbleType());

    //IF SORTING ONE MARBLE
    //Discern if channel detected no marble
    if(errorCodeChannelZero == WAR_NO_MARBLE)
    {
        //this->MoreMarbles = false;
        return WAR_NO_MARBLE;
    }

    //Marble was detected
    //this->MoreMarbles = true;

    //Set servo to sort marble based on type
    ServoZero.SetServo(MarbleZero.GetMarbleType());

    //Delay and return to nominal
    _delay_ms(500);
    ServoZero.SetServo(NoMarble);
```

Figure 13: Code from Sorter::Sort() for one marble

Entering into the sort state requires a press of the sort button, being previously in the idle state, and performing the actual sort operation requires that the *Sorter::MoreMarbles* flag is set. If not set (there are no more marbles to sort), the sorter displays a message to the user and blinks the red LED twice to alert the user that there is nothing to sort.

Note that the code for two sensors/marbles/servos at a time is still in the *Sorter::Sort()* function, and throughout the code, although it has been commented out, as we opted in the end to only use one sensor/servo. However, one can see the functionality that would be gained by using two and how the code would have implemented it.

The recall information state can only be accessed from the idle state. In the recall information state, the EEPROM is read from to obtain count information

and timing information from a previous sorting session. Part of the recall information state can be seen below in Figure 14.

```
/**********************/
/* Recall Information */
/**********************/
if((sorter.GetStartStopButtonAction() == Hold) && (sorter.State == IdleState))
{
    static uint8_t min = 0;
    static uint8_t sec = 0;
    static uint8_t whiteCount = 0;
    static uint8_t blackCount = 0;
    sorter.ButtonActionCompleted();
    sorter.State = RecallState;

    lcd.clear();
    lcd.home();
    lcd.print("Recall Information");

    min = eeprom_read_byte((uint8_t *)MIN_ADDR);
    sec = eeprom_read_byte((uint8_t *)SEC_ADDR);
    whiteCount = eeprom_read_byte((uint8_t *)WHITE_COUNT_ADDR);
    blackCount = eeprom_read_byte((uint8_t *)BLACK_COUNT_ADDR);
```

Figure 14: Sample of code from the recall information state

Below this sample code, after reading the EEPROM to variables local to the function, the variables were checked against the default EEPROM value of 0xFF (255), in which case they were assumed to be zero. The information was then printed to the LCD screen for viewing.

The reset information state can only be accessed from the idle state. In the reset information state, the EEPROM is written to, clearing out the previous count information. A temporary message on the screen alerts the user of the reset operation. The code for the reset state is shown below in Figure 15.

```
/*********************/
/* Reset Information */
/*********************/
if((sorter.GetResetButtonAction() == Press) && (sorter.State == IdleState))
{
    sorter.ButtonActionCompleted();
    sorter.State = ResetState;

    ClearLine(LINE_4);
    lcd.setCursor(0, LINE_4);
    lcd.print("Reset");

    for(int i = 0; i < 3; i++)
    {
        _delay_ms(200);
        lcd.print(".");
    }

    sorter.SecondsElapsed = 0;
    sorter.MinutesElapsed = 0;

    sorter.MarbleCount.WhiteCount = 0;
    sorter.MarbleCount.BlackCount = 0;
    sorter.MarbleCount.TotalCount = 0;
    sorter.SetLEDColor(Off);
    eeprom_update_byte((uint8_t *)MIN_ADDR, 0);
    eeprom_update_byte((uint8_t *)SEC_ADDR, 0);
    eeprom_update_byte((uint8_t *)WHITE_COUNT_ADDR, 0);
    eeprom_update_byte((uint8_t *)BLACK_COUNT_ADDR, 0);

    _delay_ms(1000);

    //Return to idle state
    sorter.State = IdleState;
    printIdleScreen = true;
}
```

Figure 15: Reset Information State

3.2.2 Software Implementation

The software used in the design was implemented in C++, rather than C, to take advantage of the object-oriented characteristics of the language. Classes were created to implement the marble, servo, and sorter structures. The classes contain relevant members like the marble type (white, black, no marble), the marble and servo indices, and sorter information like the two marbles being sorted, the two servos performing the sorting, error codes, flags, etc. The object-oriented design approach allows for encapsulation of the class members and methods. Methods for the servo class include setting the servo to a specific angle, while those of the marble class simply allow for access to the marble members. Methods of the sorter class include those to perform the sort functionality, choose the proper sensor for input, and choose the correct servo to engage, among others.

The development environment used was Atmel Studio 6.1. In addition to the standard Atmel libraries, some libraries like the I2C LCD library were imported from the Arduino environment. This was accomplished through configuring extensive link and include file hierarchies, which are specific to a users machine and computing environment. The entire process can be found here: http://www.engblaze.com/tutorial-using-atmel-studio-6-with-arduino-projects/, but note that the process is completely dependent on your folder structures, which libraries you are using, and many other factors. Thus, explaining this process in detail for my machine is not worthwhile, since another user would not follow the same steps on their machine.

4.0 Verification

The tests outlined in the Hardware Verification section of this document will be performed. The first test will be considered a success when the range of motion of each diverter can divert a marble off of the track in the intended direction. The second test will be considered a success if the slope of the track sections is enough to feed the marbles from the funnel to the end of the sensor enclosure. The third test will be considered a success if the angle and distance from the marble of the color sensors allow for successful reads. The fourth test will be considered a success if the sorting machine can successfully sort the intended 25 marbles correctly.

4.1 Hardware verification

The main hardware components that require verification is the range of motion of the diverters, the PWM switching mechanism/circuitry, and the reading/placement of the color sensors.

Test 1: Individual Diverter Movement

The diverters and the servos were mounted to the axle mounts. The servos were then connected to an Arduino development board. The Arduino was then used to test the range of motion of each individual servo and diverter. The range of motion was considered a success if it could divert a marble off of the sensor enclosure and into a bin. The servos were controlled by a PWM signal of 20ms. The values that were found to successfully divert the marble off of the sensor enclosure was 1ms 'ON' time for diverting the marble left and 2ms 'ON' time for diverting the marble right.

Test 2: Track Section Slope

The slope of the track section was verified by feeding marbles into the funnel and making sure that they traveled all the way to the end of the sensor enclosure. This was considered a success when a marble could freely travel the entire distance from the funnel to the end of the sensor enclosure. The slope of the Hopper Section of the track was about 14.36°. The slope of the intermediate track section was 3.43°.

Test 3: Placement of Color Sensors

This test will ensured that the sensors can distinguish between the presence of a marble and what color that marble is. A marble was loaded into the sensing area, the track section of the sensor enclosure. The ADC value of the sensor was printed out to the LCD screen for different cases and the results can be seen in Table 1 . These results were used in the code to distinguish between the different cases. The test was run 10 times and the ADC value for each case changed by no more than 1 ADC value.

Case	ADC Value
Black Marble	11
White Marble	7
No Marble	146

Table 1: ADC Values For Test Cases

Test 4: Final Assembly and Sorting of Full Load of Marbles

This test verified that the hardware, the custom pcb boards, and the software performed as intended. The control board was originally powered from a bench top power supply. This was to ensure that there was not any excess current drawn, indicating a problem with the power regulation circuit. Once this was verified to operate correctly the the mechanical pieces of the sorting machine were assembled and the peripherals were wired to the control board. The sensor board was built on a protoboard and also wired to the control board. The software was then flashed to the Atmega 328P that was on the control board. The sorting machine was loaded with 25 marbles. Three test runs were performed and the results can be seen in Table 2. It should be noted that the 58 second run was longer because of a jam that occurred in the track section and the funnel. The other two test runs indicated that it took about 1 second to sort each individual marble.

Run Number	Sort Successful?	Time
Run 1	Yes	58 seconds
Run 2	Yes	27 seconds
Run 3	Yes	26 seconds

Table 2: Full Sorting Load Test Results

4.2 Software verification

Verifying the following modules allowed for proper software testing. Proper software testing, in this case, refers to good repeatability. This means different things for different modules of the software, but in general repeating a process four to five times was sufficient for verification.

- Sensors:
  - The sensors input was verified to be within an acceptable range for the presence of a white marble (<= 7), black marble (>= 8, <= 11), and the absence of a marble (120-150).
- Servos
  - The servos were verified by applying the proper PWM signal to them and visibly verifying that they move to the correct position relative to the diverters. The proper offsets were also introduced by verifying that the servos indeed move to the correct angle corresponding to the PWM signal.

- Display, Buttons and Timers
  - The display, buttons and timers were all tested together. The display was loaded, and a simple timer remained stopped at 00:00. Upon pressing any of the buttons, after debouncing them (as described extensively above in section 3.2), the timer was started/stopped.
  - The buttons were also tested extensively to ensure proper debouncing. This was done by entering into test states, each of which printed a unique string to the screen, and displayed a unique LED color.
- RBG LED
  - The RBG LED was tested when the buttons/states were tested. Entering the sort state changed the LED to green, the recall information state to yellow, and the reset state to red.
- Recall Information
  - The EEPROM read/write cycles was verified by simply using one button, which was pressed once to set a byte of EEPROM memory to a certain hex value, and pressed again to clear it. Upon either writing the hex value or clearing the memory, the memory was read and printed to the LCD.
- Watchdog Timer
  - The watchdog timer verification was attempted, but conflict with the I2C protocol on the board caused (partial) abandonment of the WDT for the scope of this project. The code was written and maintained for

the WDT (as seen in the appendix), but instead of using the actual WDT, an interrupt driven output compare match system was used to set the WDT flag and mimic the process exactly.

5.0 Validation

Requirements:

1. Hopper to hold at least 25 marbles
    a. The hopper will have 25 marbles place inside, and it will be verified that all marbles successfully exit the hopper and enter the sorting chamber.
    b. Risks include marbles being stuck in the hopper. Using a hopper with a large enough exit hole will minimize the risk of stuck marbles.
2. Marbles must be sorted by color
    a. Once the servos have been attached to the diverters, a test run of 5-10 marbles will be performed.
    b. Risks include the sensors not properly identifying a marble vs. no marble. Coloring the diverters if necessary will allow for the proper ADC reading when no marble is present.
3. LCD will show the count of black and white marbles and time elapsed
    a. See section 4.2: Software verification.
4. Two momentary push buttons will be used to start/stop the process, recall past information and reset the system
    a. See section 4.2: Software verification.
    b. Risks include the buttons producing noise. Debouncing the buttons by checking if the button has been pressed for 10ms continuously will minimize the noise introduced.
5. Upon stopping the sorting process, the process should resume from the same place it left off
    a. This is inherently part of the design. A flag will be set in EEPROM which will only be cleared upon a power cycle. This flag will allow the software to maintain and display the previous count and elapsed time while the system is still powered on, regardless of entering/exiting the sort state.
    b. Risks include reading/writing from the EEPROM. This will be reduced per the software tests performed in section 4.2: Software verification.
6. Sorting information should be retained even after the system has been powered down
    a. See section 4.2: Software verification.
    b. Risks include reading and writing properly to the EEPROM. This will be achieved by drawing from the exampled code provided in the ATmega328P datasheet.
7. An RGB LED indicator will show a green light when the process is running, yellow if the process is halted, and red when the process is complete

   a. When verifying the actual sorting of the marbles per section 5.2, the LED's will be configured to work as expected, and their functionality will be verified.
   b. Risks include wiring the LED up incorrectly. This will be solved by careful placement and soldering of all components during the board population.
8. Process completion will be determined by the use of a watchdog timer. If, by the end of the process, less than 10 marbles were sorted, the LED will turn RED and blink on and off in 1 second intervals until the start/stop button is pressed
   a. Again, this will be verified during the sorting of the marbles. Once more than 10 marbles are verified to have been sorted correctly, 5 marbles will be sorted, and after the timeout occurs, the LED will be verified to be blinking red.
   b. Risks include the watchdog timer functioning properly. This will be reduced per the software tests performed in section 4.2: Software verification.
9. The system will be powered by an AC-DC power supply
   a. An AC-DC wall-wart will be purchased, eliminating the need to debug the AC-DC portion of the supply.
10. Auxiliary control circuitry will be implemented on a printed circuit board
   a. The board verification will be implemented during the final testing, after the components are placed on the board. General testing of the design functionality will provide verification for the circuit design.
   b. Risks include improper board design and component malfunction. LTspice and eagle board design tools were used to verify many of the sub-circuits, and the board was verified to be properly designed per Advanced circuit's tool.
11. The cost of all electro-mechanical parts must not exceed $50
   a. The current cost of all components is $47.75

6.0 Conclusions

The Gibson-Millwood marble sorter project was successfully completed, realizing nearly all of the requirements set forth by the client.

The mechanical design was completed and, although perhaps too many resources were used on the mechanical aspect, every finally came together and functioned in the end. The only issue with the mechanical portion was found when scaling from a few marbles to the full 25 marbles. In addition to intermittent jams of the hopper, the weight of the marbles would push on the marble being sensed, occasionally agitating it such that the sensor reading was invalid and showed that there was no marble present. The mechanical design of the diverters and servo interaction was complex, but was a unique design that yielded results.

The board was built to specifications, and *almost* everything went as planned. The switch for the PWM control did not work out, however. The small package type made it impossible to inspect solder joints, and after debugging the switch and PWM signals for many hours, the decision was made to simply switch to using one diverter at a time and to simplify the design as such. This was a great decision, which resulted in completing the entire design the very next day, a goal that was not in sight following the problems with the switch.

Due to the boards arriving late, the software was written initially all in one go. Thousands of lines of code in C++ were written with no board to test on, and so having the vast majority of the software work without too many hitches was welcomed. The most major issue with the software was certainly the PWM signal issue. As it turned out, driving a PWM on PB1 (OC1A) is not recommended. We spent hours debugging our circuits, other code, and anything we could think of, ignorantly thinking it must have been our hardware/software, when in fact we just needed to switch to PB2 instead. However in all, the software worked quite well. The use of C++ contributed to an elegant software design, with readable code, compact functions, and tireless encapsulation implemented throughout. C++ allowed the code to be very neat and tight, and the object oriented capabilities were used to their fullest to provide a robust software package.

In all, the sorter worked quite well. The sorter never made a mistake in distinguishing between a black or white marble, and each time it was sensed, the servo was able to turn the diverter successfully and knock the marble into its corresponding bin. The sorter had an average time of just over 1 second per marble, resulting in sort times of 26-28 seconds for 25 marbles. Many, many hours were

spent on the design of this project. Due to the relative complexity of designing a custom ATmega328 board instead of using the Arduino platform, and due to the mechanical complexity of the design, and more than anything due to a glitch in the PWM output on a particular pin (PB1), the project was unable to be displayed on project day. However, it was completed later on that very same day, and worked quite well, all things considered.

7.0 Project Budget and Schedule

7.1 Budget

The budget constraints for this project was to keep the cost of the electro-mechanical and electrical parts under $50. The most expensive items were the LCD module, the servos, the Atmega 328P microcontroller, and a few other integrated circuit devices. Every other item was less than a dollar USD. Tables 3,4, and 5 show a summary of purchased items, the detailed Bill of Materials can be seen in Appendix B.

Capacitors		
Designator	**Description**	**Qnty.**
C3, C4	100nF CAPACITOR0805	2
C8	100uF POL-CAPACITOR5-10.5	1
C1	10nF CAPACITOR0805	1
C9, C10, C7	10uF POL-CAPACITOR2-5	3
C5, C6	22pF CAPACITOR0805	2
C5, C6	22pF CAPACITOR0805	2

Resistors		
Designator	**Description**	**Qnty.**
R7, R21, R23, R25, R28	10k RESISTOR0805	5
R12	150 RESISTOR0805	1
R22, R27	180 RESISTOR0805	2
R10, R11, R24, R26	1k RESISTOR0805	4
R1, R2, R3, R4	220 RESISTOR0805	4
R8	240 RESISTOR0805	1
R5, R6	270 RESISTOR0805	2
R9	910 RESISTOR0805	1

Other		
Designator	**Description**	**Qnty.**
-	HS-311 Servo	2
-	LCD – QC1602A	1

Table 3: Summary Of Capacitors, Resistors, and Some Other Components

Transistors		
Designator	Description	Qnty.
T1	MMBT3904 NPN MMBT3904LT1-NPN-SOT23-BEC	1
Q1, Q2	MMBT3906 TRANSISTOR_PNPMMBT3906	2
Q3, Q4	Through-hole side-receiving IR phototransistor	2

Diodes		
Designator	Description	Qnty.
D1	1N4004	1
LED4	GREEN LEDGREEN	1
LED1, LED2, LED3	NA LED-1206 LED1206	3
LED7, LED8	NA LED3MM LED3MM	2
LED6	RED LEDGREEN	1
LED5	YELLOW LEDGREEN	1

Table 4: Summary of Transistors and Diodes that were purchased

Connectors		
Designat	Description	Qnty.
JP2	9V Power Supply Input M02PTH	1
ISP1	AVRISP-6	1
JP1	USB Connectors	1

Special ICs		
Designat	Description	Qnty.
U1	ATMEGA328P ATMEGA168	1
U2	FT232RLSSOP	1
IC1	Low drop fixed and adjustable positive voltage regulators 1 A	1
IC2	Positive VOLTAGE REGULATOR	1

Table 5: Summary of Connectors and Special ICs

7.2 Schedule

Date	Milestone	Completed
10/4/13	IR Sensors implemented to sense marble color	Y
10/11/13	Servo control implemented using PWM	Y
11/1/13	Circuit board designed in Eagle	Y
11/8/13	Software implemented for I2C LCD	Y
11/8/13	Mechanical design completed for sorting chamber	Y
11/15/13	Software implemented in C++ for Marble, Servo, Sorter, etc. structures	Y
11/22/13	Board assembly	Y
11/22/13	Board testing	Y
11/22/13	Combine PCB with breakout sensor board	Y
11/22/13	Combine mechanical and electrical systems	Y
11/29/13	Complete software implementation	Y
11/29/13	Final testing	Y
12/5/13	Project Day	Y

8.0 Revision History

(this document will be revised to include additional information describing the final design implementation and test results)

Revision Number	Date	Author	Changes
A	Nov 1	Joe Gibson And Jesse Millwood	Changed Diverter Design to be shorter to accommodate servo horn
B	Nov 8	Jesse Millwood	Altered Front Axle Mount Design
C	Nov 15	Joe Gibson	Moving From C to C++
D	Nov 16	Jesse Millwood	Finalized and started building track sections
E	Nov 18	Joe Gibson And Jesse Millwood	Finalized case and mounting of control board enclosure

Appendix A – Electrical Schematics

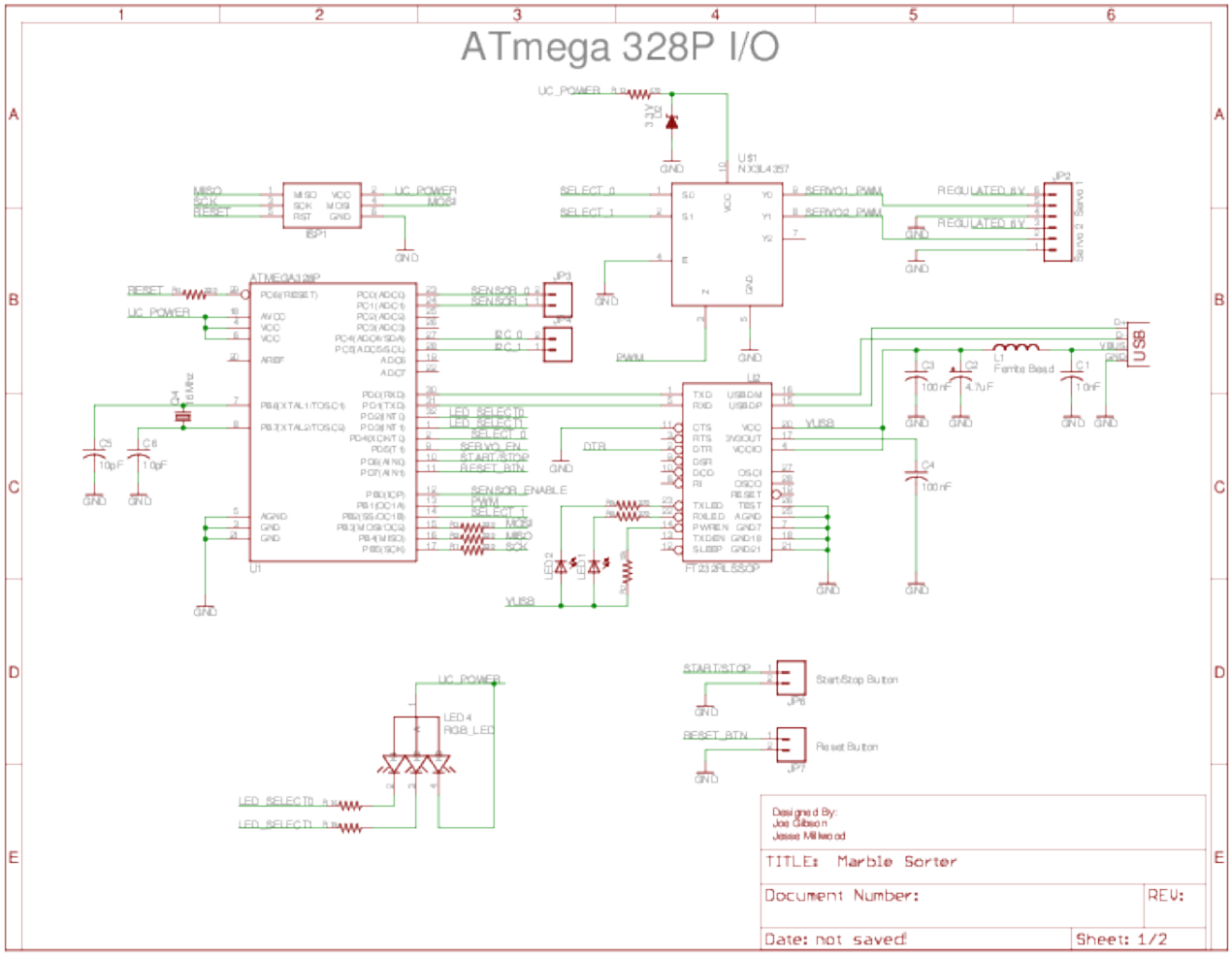
The following pages include the schematic diagrams as well as the board layout in the follwing order:

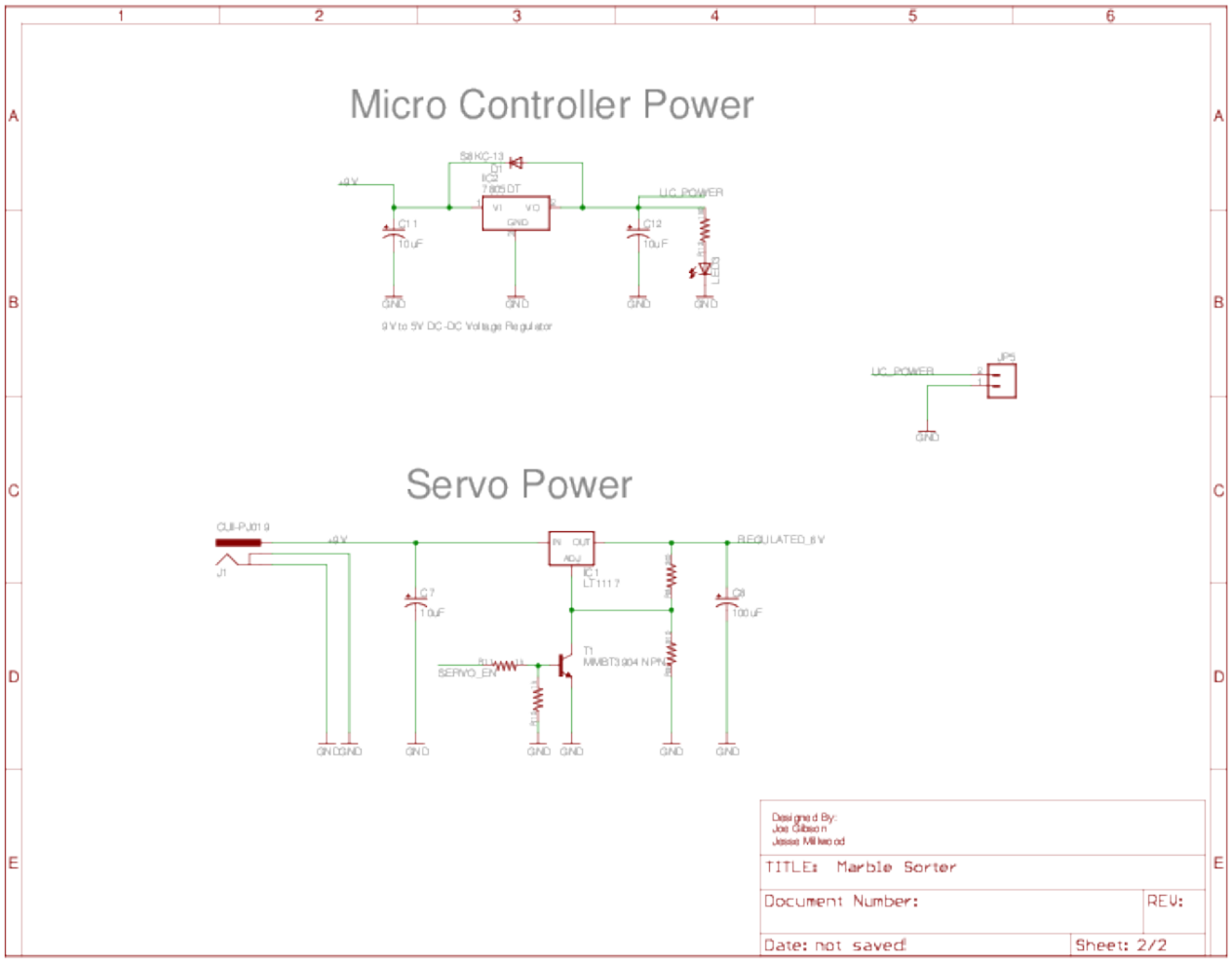
1. ATmega328P and I/O Interface

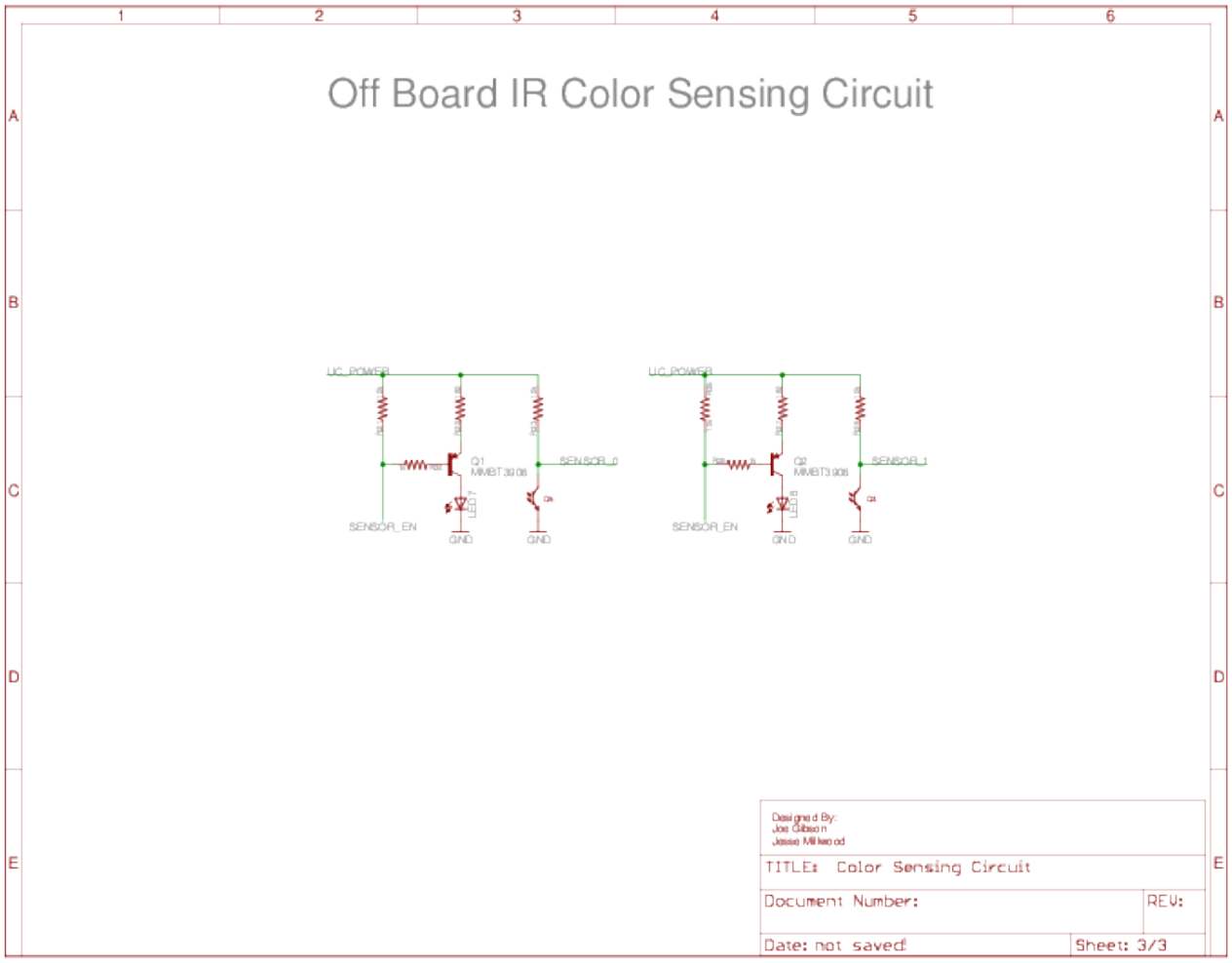
2. Microcontroller and Servo Power

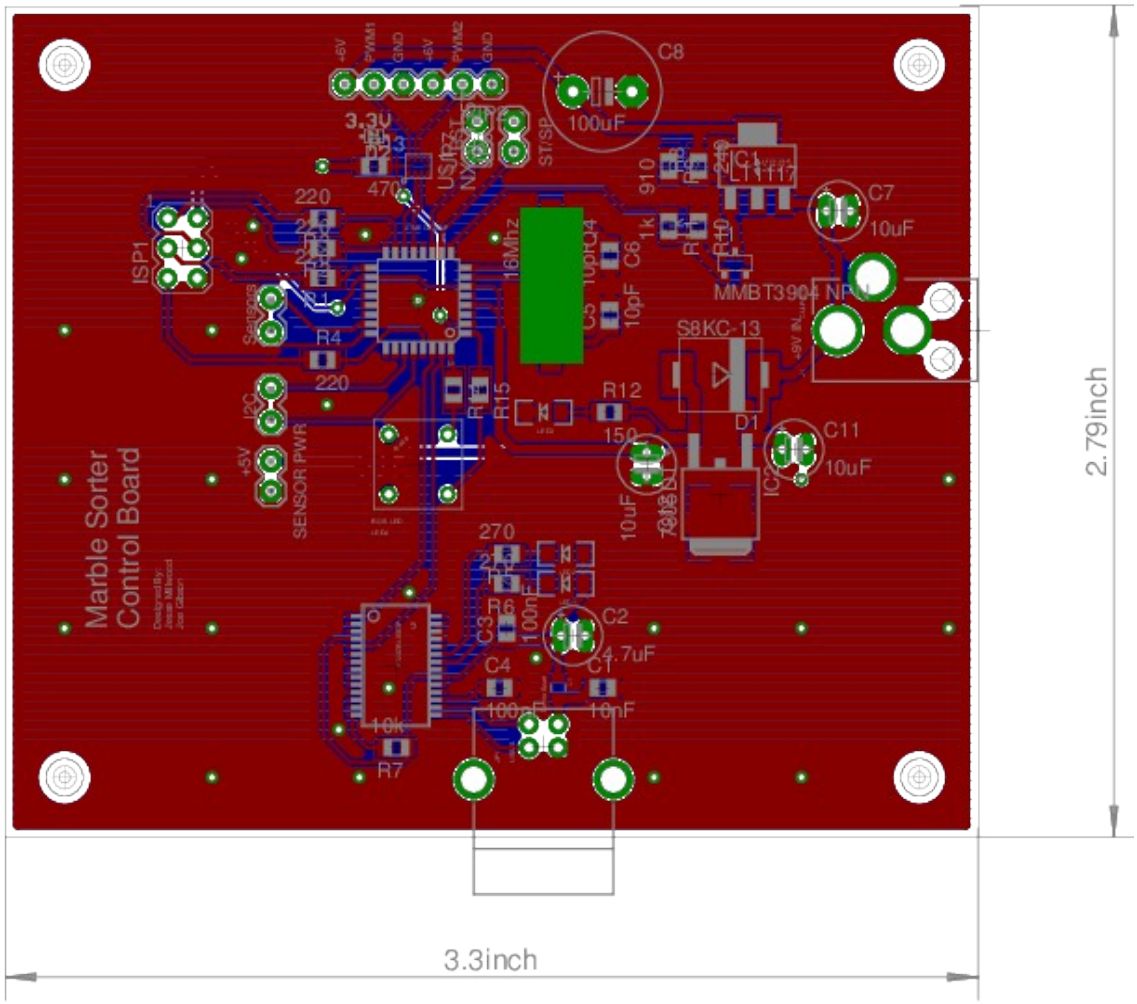
3. Off Board Color Sensing Circuit

4. Board Layout


ATmega 328P I/O
Designed By:
Joe Gibson
Jessee Millwood
TITLE: Marble Sorter
Document Number:
REV:
Date: not saved
Sheet: 1/2


Micro Controller Power
9 V to 5V DC-DC Voltage Regulator
Servo Power
Designed By:
Joe Gibson
Jessie Millwood
TITLE: Marble Sorter
Document Number:
REV:
Date: not saved
Sheet: 2/2


Off Board IR Color Sensing Circuit
UC_POWER
Q1
MMBT3906
SENSOR_0
LED 7
SENSOR_EN
GND
GND
UC_POWER
Q2
MMBT3906
SENSOR_1
LED 6
SENSOR_EN
GND
GND
Designed By:
Joe Gibson
Jessie Millwood
TITLE: Color Sensing Circuit
Document Number:
REV:
Date: not saved!
Sheet: 3/3


Marble Sorter
Control Board
2.79inch
3.3inch
C8
100uF
C7
10uF
C11
10uF
C2
4.7uF
C1
10nF
C4
100nF
C3
100nF
C6
10pF
R1
220
R4
220
R6
270
R7
10K
R12
150
R15
910
R10
1k
16Mhz
MMBT3904 NPN
S8KC-13
D1
ISP1
I2C
SENSOR PWR
+5V
3.3V
PWM1
PWM2
GND
ST/SP

Appendix B – Bill of Materials

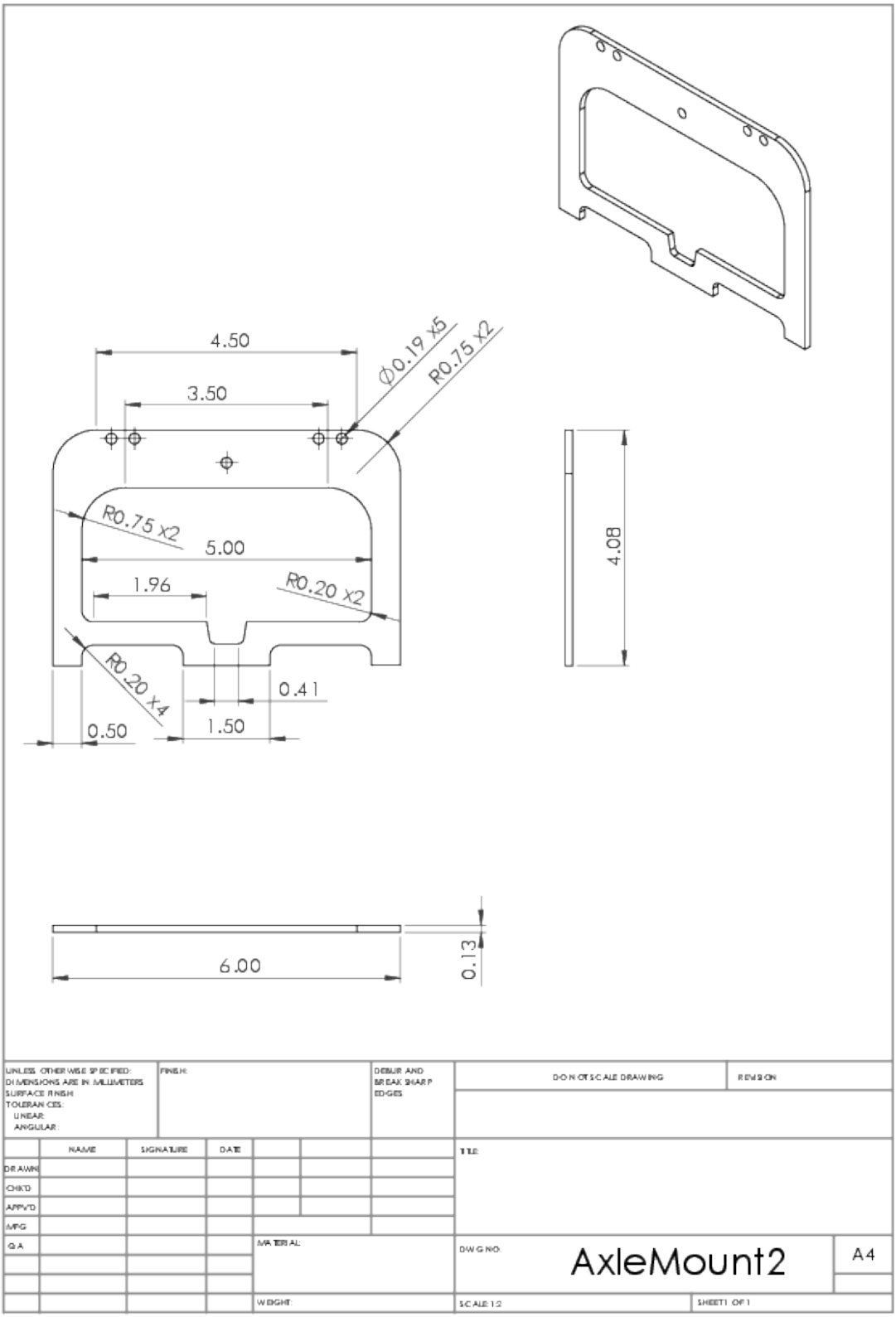
The following table includes the items purchased for this project as well as the manufacturing costs.

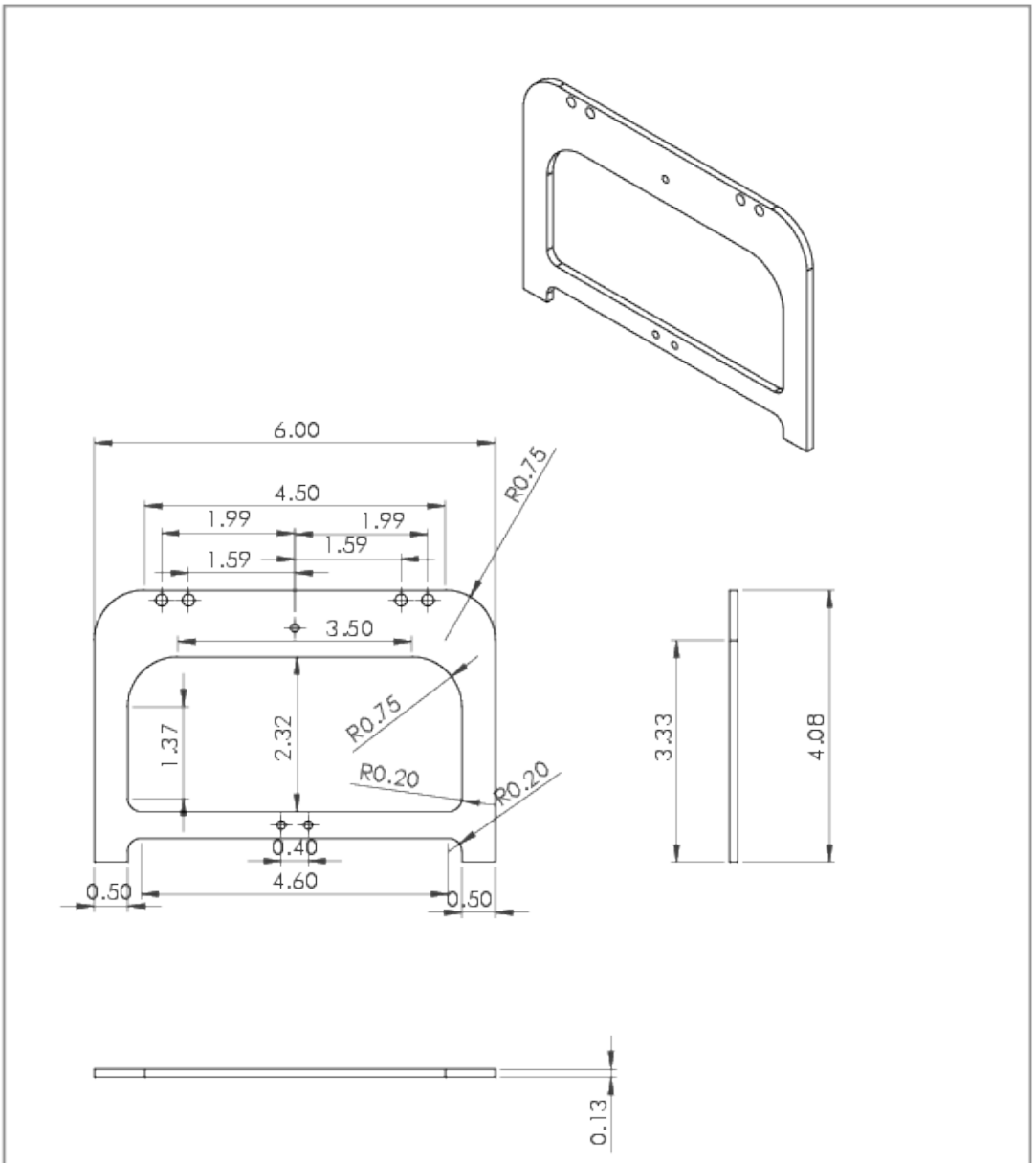
Designator	Description	Qnty.	Supplier	Supplier PN	Individual Price	Extended Price
R7, R21, R23, R25, R28	10k RESISTOR0805	5	DigiKey	311-10.0KCRCT-ND	$0.10	$0.50
R12	150 RESISTOR0805	1	DigiKey	311-150CRCT-ND	$0.10	$0.10
R22, R27	180 RESISTOR0805	2	DigiKey	311-180CRCT-ND	$0.10	$0.20
R10, R11, R24, R26	1k RESISTOR0805	4	DigiKey	311-1.00KCRCT-ND	$0.10	$0.40
R1, R2, R3, R4	220 RESISTOR0805	4	DigiKey	314-220CRCT-ND	$0.10	$0.40
R5, R6	270 RESISTOR0805	2	DigiKey	311-270CRCT-ND	$0.10	$0.20
R9	910 RESISTOR0805	1	DigiKey	311-910CRCT-ND	$0.10	$0.10
C3, C4	100nF CAPACITOR0805	2	DigiKey	399-1170-1-ND	$0.10	$0.20
C8	100uF POL-CAPACITOR5-10.5	1	DigiKey	P10323-ND	$0.54	$0.54
C1	10nF CAPACITOR0805	1	DigiKey	399-1158-1-ND	$0.10	$0.10
C9, C10, C7	10uF POL-CAPACITOR2-5	3	DigiKey	P5148-ND	$0.20	$0.60
C5, C6	22pF CAPACITOR0805	2	DigiKey			$0.00
C2	4.7uF CAPACITOR0805	1	DigiKey	P15148-ND	$0.46	$0.46
Y1	16MHz CRYSTAL10.5X4.8	1	DigiKey	631-1017-1-ND	$0.52	$0.52
D1	1N4004	1	DigiKey	S8KCDICT-ND	$0.66	$0.66
ISP1	AVR ISP HEADER	1	DigiKey	952-2121-ND	$0.21	$0.21
IC1	LT1117 LD117AS12TR	1	DigiKey	LT1117CSTMPBF-ND	$3.89	$3.89
IC2	7805DT	1	DigiKey	ND	$0.64	$0.64
U1	ATMEGA328P ATMEGA168	1	DigiKey	ATMEGA328P-15AZCT-ND	$8.45	$8.45
U2	FT232RLSSOP	1	DigiKey	768-1007-1-ND	$4.50	$4.50
U$1	IC ANALOG SWITCH SP3T 10XQFN	1	DigiKey	568-5563-1-ND	0.62	$0.62
T1	MMBT3904 NPN MMBT3904LT1-NPN-SOT23-BEC	1	DigiKey	MMBT4401LT1GOSCT-ND	$0.11	$0.11
Q1, Q2	MMBT3906 TRANSISTOR_PNPMMBT3906	2	DigiKey	MMBT4403LT1GOSCT-ND	$0.11	$0.22
Q3, Q5	NA LTR-301 LTR-301	2	DigiKey	160-1030-ND	$0.38	$0.76
LED1, LED2, LED3	NA LED-1206 LED1206	3	DigiKey	APT3216SGC	$0.20	$0.60
LED7, LED8	NA LED3MM LED3MM	2	DigiKey	160-1029-ND	$0.49	$0.98
LED6	RED LEDGREEN	1	DigiKey	APT3216SRCPRV	$0.20	$0.20
LED5	YELLOW LEDGREEN	1	DigiKey	APT3216YC	$0.20	$0.20
Off Board	I2C Expander	1	DigiKey	296-13110-1-ND	$1.91	$1.91
Off Board	LCD	1	DigiKey	QC1602A	$7.50	$7.50
Off Board	HS-311 Servo (rcplanet.com)	2	rcplanet		$6.99	$13.98
				Parts Total		$47.75
-	Board Manufacturing	1	Circuits		$35.00	$35.00
				Total		$82.75

Appendix C – Detailed Mechanical Drawings

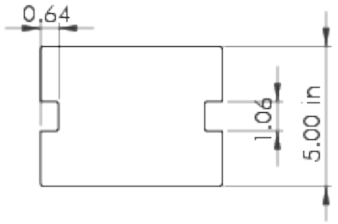
The following pages of the appendix include the 3-view drawings of the mechanical parts that were made for this project. The order of the drawings are as follows:

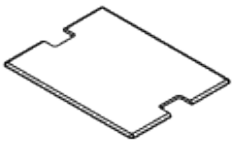
1. Axle Mount Front
2. Axle Mount Stop
3. Base
4. Diverter Model that was used for manufacturing
5. Hopper Track Section
6. Intermediate Track Section
7. Sensor Enclosure

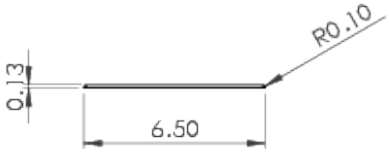

4.50
3.50
Ø0.19 x5
R0.75 x2
R0.75 x2
5.00
1.96
R0.20 x2
R0.20 x4
0.41
0.50
1.50
4.08
6.00
0.13
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBUR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
TITLE:
DWG NO.
AxleMount2
A4
SCALE: 1:2
SHEET 1 OF 1



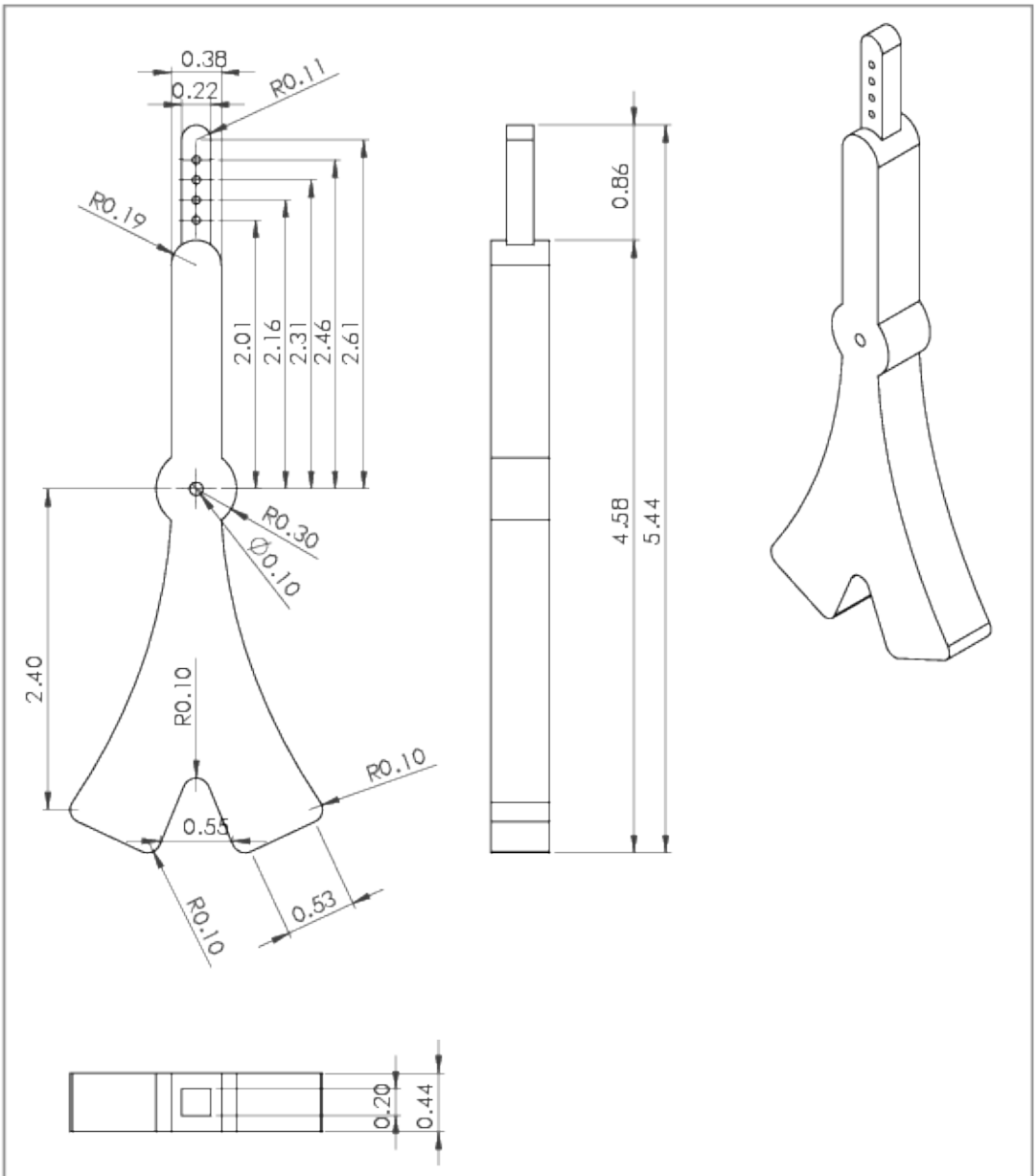
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBUR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE				TITLE:
DRAWN							
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO.	AxleMountSTOP A4
				WEIGHT:		SCALE: 1:2	SHEET 1 OF 1



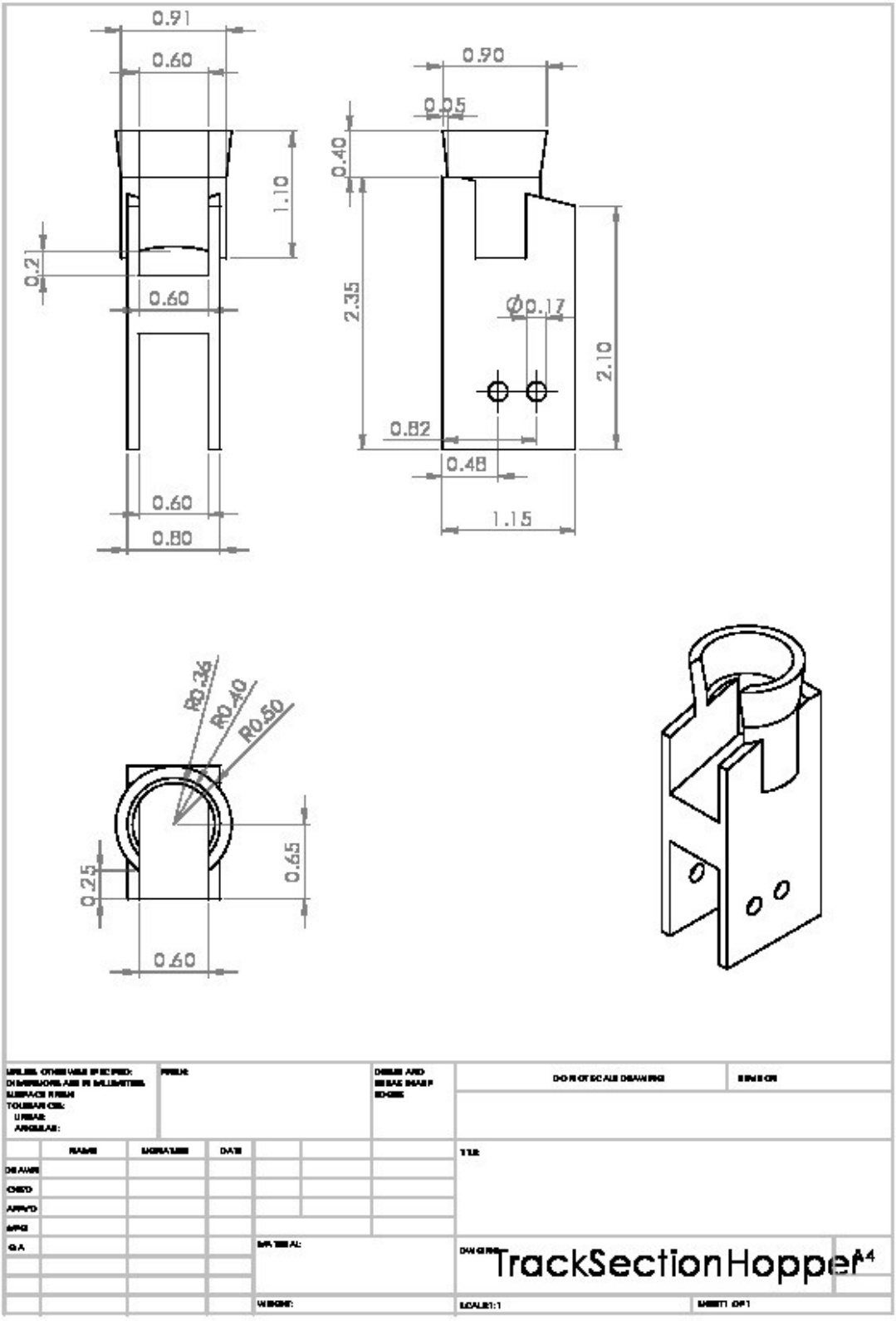


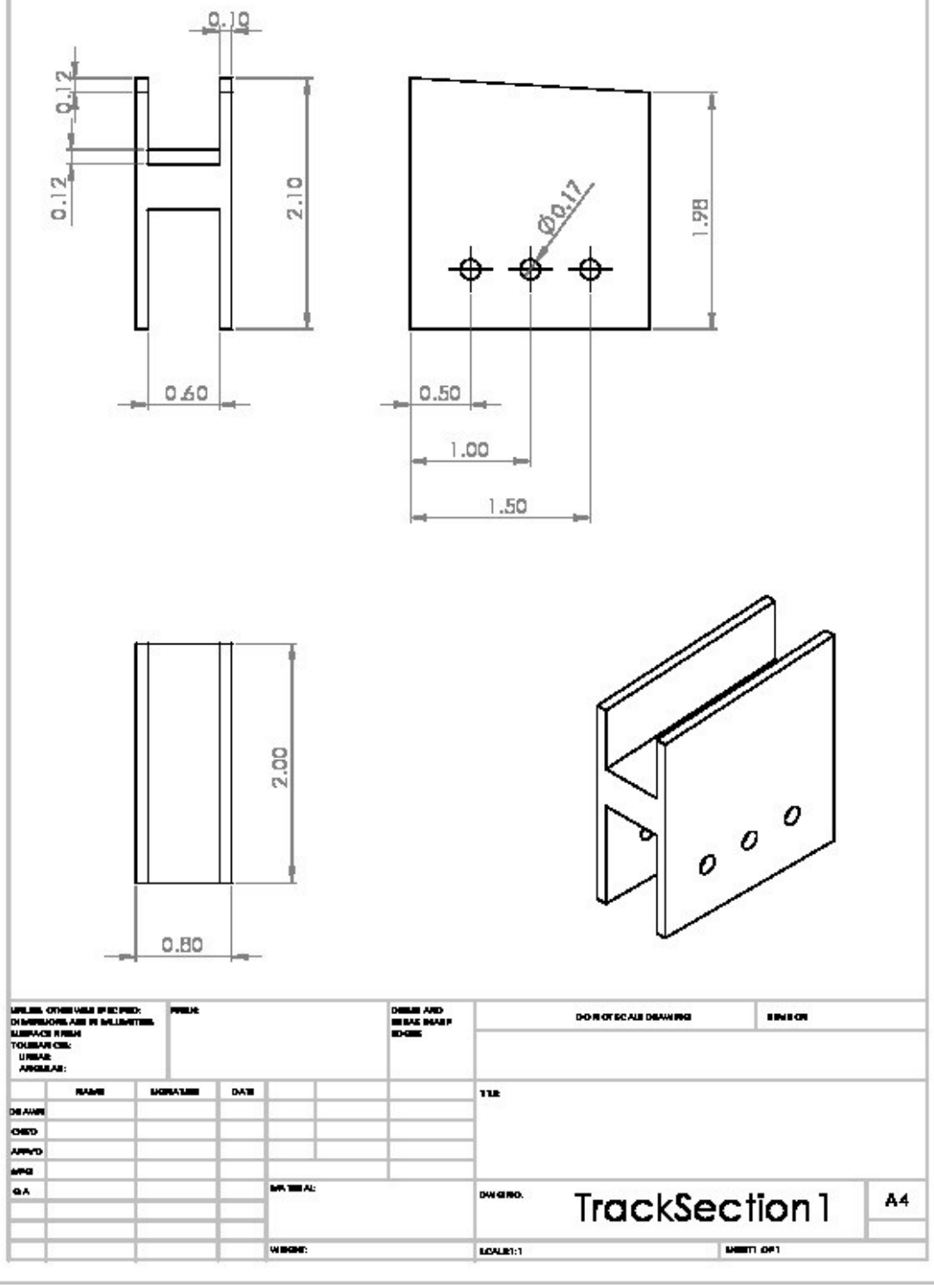


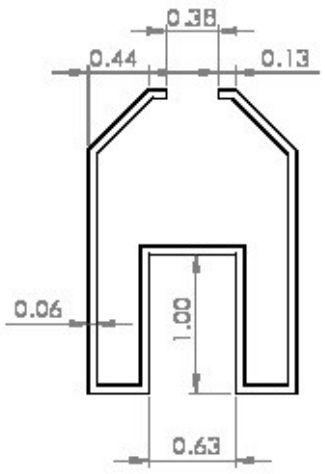
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:			DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE				TITLE:
DRAWN							
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO: Base	A4
				WEIGHT:		SCALE: 1:5	SHEET 1 OF 1

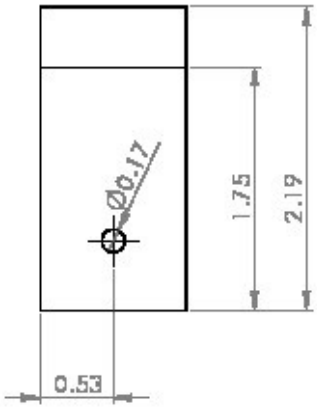


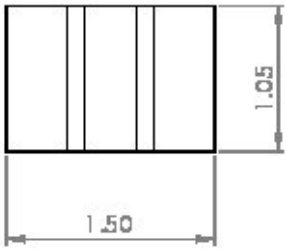
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBUR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE				TITLE:
DRAWN							
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO. DiverterFORMAN	A4
				WEIGHT:		SCALE:1:1	SHEET 1 OF 1

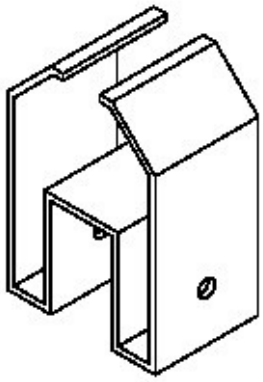

0.91
0.60
1.10
0.2
0.60
0.60
0.80
0.90
0.05
0.40
2.35
Ø0.17
2.10
0.82
0.48
1.15
R0.36
R0.40
R0.50
0.65
0.25
0.60
TrackSectionHopper


0.10
0.12
0.12
2.10
0.60
Ø0.17
1.98
0.50
1.00
1.50
2.00
0.80
TrackSection1
A4









UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING / REVISION
	NAME	SIGNATURE	DATE
DRAWN			
CHK'D			
APP'D			
MFG			
Q.A			
MATERIAL:			TITLE: Sensor Enclosure
WEIGHT:			SCALE: 1:1 / SHEET 1 OF 1 / A4

Appendix D – Source Code

The following pages contain the source code used for this project in the following order:

1. main.cpp
2. Global.h
3. Marble.h
4. Servo.h
5. Sorter.h

```
/*****************************************************************************/
/* File: main.cpp                                                            */
/* Author: Joe Gibson and Jesse Millwood                                     */
/* Date: 11/5/13                                                             */
/* Course: EGR 326                                                           */
/* Description: main.cpp implements the main file for the marble sorter      */
/*              software. This file interfaces with the Sorter, Marble,      */
/*              Servo, and other classes                                     */
/*                                                                           */
/* Grand Valley State University, 2013                                       */
/*****************************************************************************/
#define F_CPU 16000000L

#include <avr/interrupt.h>          //AVR interrupt library
#include <avr/wdt.h>                //AVR Watchdog timer library
#include <avr/eeprom.h>             //AVR EEPROM library
#include <Arduino.h>                //Arduino library
#include <LiquidCrystal_I2C.h>      //I2C LCD library
#include <string.h>                 //String library
#include "Global.h"                 //Global header file
#include "Marble.h"                 //Marble class definition
#include "Servo.h"                  //Servo class definition
#include "Sorter.h"                 //Sorter class definition

//Create the LCD object
LiquidCrystal_I2C lcd(I2C_ADDRESS, EN, RW, RS, D4, D5, D6, D7, BL, BL_POL);

//Create the sorter object
Sorter sorter;

int ResetCount = 0;
int StartStopCount = 0;

/*****************************************************************************/
/* SETUP AND LOOP                                                            */
/*****************************************************************************/
/*****************************************************************************/
/* Setup                                                                     */
/*****************************************************************************/
void setup(void)
{
    //Reset WDT right away
    wdt_reset();

    //Initialize everything
    InitPortDirections();
    InitTimers();
    InitADC();
    InitWDT();
    InitEEPROM();
    InitSorter();
    InitLCD();

    //Enable global interrupts
    sei();
}
```

```
/*****************************************************************************/
/* Main Loop                                                                 */
/*****************************************************************************/
void loop(void)
{
    static char tmp[LINE_LEN];
    static char dots = 0;
    static bool printIdleScreen = true;

    //Print the idle screen if necessary
    if(printIdleScreen)
    {
        PrintIdleScreen();
        printIdleScreen = false;
    }

    //Reset WDT
    wdt_reset();

    /********/
    /* Sort */
    /********/
    if((sorter.GetStartStopButtonAction() == Press) && (sorter.State == IdleState
        ))
    {
        sorter.ButtonActionCompleted();

        //Check if there are more marbles to be sorted
        if(sorter.MoreMarbles)
        {
            sorter.State = SortState;
            sorter.SetLEDColor(Green);

            lcd.clear();
            lcd.home();
            lcd.print("Sorting");

            while((sorter.GetStartStopButtonAction() != Press) && !sorter.WDTFlag
                )
            {
                //Check for 100ms sorter flag
                if(sorter.Flag)
                {
                    //Clear sort flag
                    sorter.Flag = false;

                    //Complete one sort cycle
                    sorter.Sort();

                    //Print dot animation
                    lcd.setCursor(7, LINE_1);

                    switch(dots++)
                    {
                        case 0:
```

```
                    lcd.print("   ");
                    break;
                case 1:
                    lcd.print(".  ");
                    break;
                case 2:
                    lcd.print(".. ");
                    break;
                case 3:
                    lcd.print("...");
                    dots = 0;
                    break;
            }

            //Update screen
            lcd.setCursor(0, LINE_3);
            sprintf(tmp, "W: %03d        B: %03d", sorter.MarbleCount.
                WhiteCount, sorter.MarbleCount.BlackCount);
            lcd.print(tmp);

            lcd.setCursor(0, LINE_4);
            sprintf(tmp, "     %02d:%02d:%d00    ", sorter.MinutesElapsed
                , sorter.SecondsElapsed, sorter.TenthsOfSecondsElapsed);
            lcd.print(tmp);
        }

        else
        {
            _delay_us(10);
        }
    }

    //Exited due to WDT flag
    if(sorter.WDTFlag)
    {
        //Clear WDTFlag
        sorter.WDTFlag = false;

        //Check number of marbles sorted
        if(sorter.MarbleCount.TotalCount >= SORT_THRESHOLD)
        {
            //Sorter was able to sort 10 marbles
            sorter.SetLEDColor(Red);
        }
        else
        {
            //Sorter was not able to sort 10 marbles
            //Flash the LED Red and wait until start/stop
            // button is pressed to acknowledge
            sorter.FlashLED = true;
            lcd.clear();
            lcd.setCursor(0, LINE_2);
            lcd.print("        ERROR       ");

            lcd.setCursor(0, LINE_3);
            lcd.print("PRESS S to Continue");
```

```
                while(sorter.GetStartStopButtonAction() != Press)
                {
                    _delay_us(10);
                    //wdt_reset();
                }

                sorter.ButtonActionCompleted();

                sorter.SetLEDColor(Off);
                sorter.FlashLED = false;
            }
        }

        //Exited due to pressing start/stop
        else
        {
            sorter.ButtonActionCompleted();
            sorter.SetLEDColor(Yellow);
        }

        //Return to idle state
        sorter.ButtonActionCompleted();
        sorter.State = IdleState;
        printIdleScreen = true;
    }

    //There are no marbles to sort
    else
    {
        lcd.setCursor(0, LINE_4);
        lcd.print("No More Marbles");

        sorter.SetLEDColor(Red);
        _delay_ms(200);
        sorter.SetLEDColor(Off);
        _delay_ms(200);
        sorter.SetLEDColor(Red);
        _delay_ms(200);
        sorter.SetLEDColor(Off);

        ClearLine(LINE_4);
    }
}

/**********************/
/* Recall Information */
/**********************/
if((sorter.GetStartStopButtonAction() == Hold) && (sorter.State == IdleState)
    )
{
    static uint8_t min = 0;
    static uint8_t sec = 0;
    static uint8_t whiteCount = 0;
    static uint8_t blackCount = 0;
    sorter.ButtonActionCompleted();
```

```
    sorter.State = RecallState;

    lcd.clear();
    lcd.home();
    lcd.print("Recall Information");

    min = eeprom_read_byte((uint8_t *)MIN_ADDR);
    sec = eeprom_read_byte((uint8_t *)SEC_ADDR);
    whiteCount = eeprom_read_byte((uint8_t *)WHITE_COUNT_ADDR);
    blackCount = eeprom_read_byte((uint8_t *)BLACK_COUNT_ADDR);

    if(min == 0xFF)
    {
        min = 0;
    }
    if(sec == 0xFF)
    {
        sec = 0;
    }
    if(whiteCount == 0xFF)
    {
        whiteCount = 0;
    }
    if(blackCount == 0xFF)
    {
        blackCount = 0;
    }

    lcd.setCursor(0, LINE_2);
    sprintf(tmp, "Time: %02d:%02d:000", min, sec);
    lcd.print(tmp);

    lcd.setCursor(0, LINE_3);
    sprintf(tmp, "White Count: %03d", whiteCount);
    lcd.print(tmp);

    lcd.setCursor(0, LINE_4);
    sprintf(tmp, "Black Count: %03d", blackCount);
    lcd.print(tmp);

    //Wait for start/stop button to be held
    while(sorter.GetStartStopButtonAction() != Hold)
    {
        _delay_us(10);
        wdt_reset();
    }

    sorter.ButtonActionCompleted();

    //Return to idle state
    sorter.State = IdleState;
    printIdleScreen = true;
}

/**********************/
/* Reset Information */
```

```
    /**********************/
    if((sorter.GetResetButtonAction() == Press) && (sorter.State == IdleState))
    {
        sorter.ButtonActionCompleted();
        sorter.State = ResetState;

        ClearLine(LINE_4);
        lcd.setCursor(0, LINE_4);
        lcd.print("Reset");

        for(int i = 0; i < 3; i++)
        {
            _delay_ms(200);
            lcd.print(".");
        }

        sorter.SecondsElapsed = 0;
        sorter.MinutesElapsed = 0;

        sorter.MarbleCount.WhiteCount = 0;
        sorter.MarbleCount.BlackCount = 0;
        sorter.MarbleCount.TotalCount = 0;
        sorter.SetLEDColor(Off);
        eeprom_update_byte((uint8_t *)MIN_ADDR, 0);
        eeprom_update_byte((uint8_t *)SEC_ADDR, 0);
        eeprom_update_byte((uint8_t *)WHITE_COUNT_ADDR, 0);
        eeprom_update_byte((uint8_t *)BLACK_COUNT_ADDR, 0);

        _delay_ms(1000);

        //Return to idle state
        sorter.State = IdleState;
        printIdleScreen = true;
    }

    /***************/
    /* TEST STATE */
    /***************/
    if((sorter.GetResetButtonAction() == Hold) && (sorter.State == IdleState))
    {
        sorter.ButtonActionCompleted();
        sorter.State = TestState;

        lcd.clear();
        lcd.home();
        lcd.print("TEST STATE");

        //Wait for reset button to be held
        while(sorter.GetResetButtonAction() != Hold)
        {
            _delay_us(10);
            wdt_reset();
        }

        sorter.ButtonActionCompleted();
```

```
        //Return to idle state
        sorter.State = IdleState;
        printIdleScreen = true;
    }
}

/*****************************************************************************/
/* INTERRUPTS                                                                */
/*****************************************************************************/
/*****************************************************************************/
/* 4s Watchdog Timer                                                         */
/*****************************************************************************/
ISR(WDT_vect)
{
    //Only set the flag if in the sort state
    if(sorter.State == SortState)
    {
        //Set the flag only if there are no more marbles
        if(!sorter.MoreMarbles)
        {
            sorter.WDTFlag = true;
        }
    }
}

/*****************************************************************************/
/* Timer 0 Output Compare A                                                  */
/*****************************************************************************/
ISR(TIMER0_COMPA_vect)
{
    static int count = 0;
    static int wdtCount = 0;
    static int timeCount = 0;
    static bool toggle = 0;

    //Increment the count
    count++;

    //Increment the WDT count
    wdtCount++;

    //Increment elapsed time count if sorting
    if(sorter.State == SortState)
    {
        timeCount++;
    }

    //2s set flag to mimic WDT
    if(wdtCount >= 200)
    {
        //Clear the WDT count
        wdtCount = 0;

        //Only set the flag if in the sort state
        if(sorter.State == SortState)
        {
```

```
            //Set the flag only if there are no more marbles
            if(!sorter.MoreMarbles)
            {
                //Set the flag
                sorter.WDTFlag = true;
            }
        }
    }

    //100 x 10ms for 1s delay
    if(count >= 100)
    {
        count = 0;
        sorter.Flag = true; //Set flag
    }

    //Keep track of seconds and minutes and flash LED if necessary
    if(timeCount >= 100)
    {
        timeCount = 0;

        //sorter.TenthsOfSecondsElapsed ++;

        sorter.SecondsElapsed++;

        /*
        if(sorter.TenthsOfSecondsElapsed == 10)
        {
            sorter.SecondsElapsed++;

            lcd.setCursor(0, LINE_2);
            lcd.print(sorter.SecondsElapsed);

            sorter.TenthsOfSecondsElapsed = 0;
        }
        */

        if(sorter.SecondsElapsed >= 60)
        {
            sorter.MinutesElapsed++;
            sorter.SecondsElapsed = 0;
        }

        toggle ^= true;

        if(sorter.FlashLED)
        {
            if(toggle)
            {
                sorter.SetLEDColor(Red);
            }
            else
            {
                sorter.SetLEDColor(Off);
            }
        }
```

```
    }
}

/*****************************************************************************/
/* Timer 2 Output Compare A                                                  */
/*****************************************************************************/
ISR(TIMER2_COMPA_vect)
{
    static int resetCount = 0;
    static int startStopCount = 0;
    static int noMoreMarblesCount = 0;

    //Check if marble present
    if(sorter.CheckForMoreMarbles() == WAR_NO_MARBLE)
    {
        noMoreMarblesCount++;
    }
    else
    {
        noMoreMarblesCount = 0;
    }

    if(noMoreMarblesCount > No_MORE_MARBLES_THRESHOLD)
    {
        sorter.MoreMarbles = false;
    }
    else
    {
        sorter.MoreMarbles = true;
    }

    //Check Reset Button (Active low)
    if(!(PIND & RESET_BTN))
    {
        resetCount++;
        sorter.ButtonActionReady = false;
    }
    else
    {
        if(resetCount >= PRESS_TIME)
        {
            sorter.ButtonActionReady = true;
        }

        resetCount = 0;
    }

    ResetCount = resetCount;

    //Check Start/Stop Button (Active low)
    if(!(PIND & START_STOP_BTN))
    {
        startStopCount++;
        sorter.ButtonActionReady = false;
    }
    else
```

```
    {
        if(startStopCount >= PRESS_TIME)
        {
            sorter.ButtonActionReady = true;
        }

        startStopCount = 0;
    }

    StartStopCount = startStopCount;

    //Handle reset button
    if(resetCount < PRESS_TIME)
    {
        //sorter.SetResetButtonAction(None);
        ;
    }
    else if((resetCount >= PRESS_TIME) && (resetCount < HOLD_TIME))
    {
        sorter.SetResetButtonAction(Press);
        sorter.SetStartStopButtonAction(None);
    }
    else
    {
        resetCount = HOLD_TIME;
        sorter.SetResetButtonAction(Hold);
        sorter.SetStartStopButtonAction(None);
    }

    //Handle start/stop button
    if(startStopCount < PRESS_TIME)
    {
        //sorter.SetStartStopButtonAction(None);
        ;
    }
    else if((startStopCount >= PRESS_TIME) && (startStopCount < HOLD_TIME))
    {
        sorter.SetStartStopButtonAction(Press);
        sorter.SetResetButtonAction(None);
    }
    else
    {
        startStopCount = HOLD_TIME;
        sorter.SetStartStopButtonAction(Hold);
        sorter.SetResetButtonAction(None);
    }
}

/*****************************************************************************/
/* GLOBAL FUNCTIONS                                                          */
/*****************************************************************************/
/*****************************************************************************/
/* Initialize Port Directions                                                */
/*****************************************************************************/
void InitPortDirections(void)
{
```

```
    //Clear all bits
    DDRB = 0x00;
    DDRC = 0x00;
    DDRD = 0x00;

    //PORTB OUTPUTS
    DDRB |= SWITCH_S0 | SWITCH_S1 | SENSOR_EN | SERVO_PWM;

    //PORTD OUTPUTS
    DDRD |= SERVO_EN | LED_RED | LED_GREEN;

    //PORTC INPUTS
    DDRC &= ~(SENSOR_0 | SENSOR_1);

    //PORTD INPUTS
    DDRD &= ~(START_STOP_BTN | RESET_BTN);

    //Enable pull up resistors
    PORTD |= (START_STOP_BTN | RESET_BTN);
}

/*************************************************************************/
/* Initialize Timers                                                     */
/*************************************************************************/
void InitTimers(void)
{
    cli();

    //Configure Timer 0 to delay 10ms
    TCCR0A = _BV(WGM01);            //CTC Mode
    TCCR0B = _BV(CS02) | _BV(CS00); //1:1024 Prescaler
    TIMSK0 = _BV(OCIE0A);           //Enable compare interrupt
    OCR0A = CYCLES_1;               //Set OCR0A to the correct number of cycles

    //Configure Timer 1 for Phase-Correct PWM mode and 20ms period
    TCCR1A = _BV(COM1B1) |  _BV(WGM10) | _BV(WGM11);    //Clear PB2 on rise, set
        on fall
    TCCR1B = _BV(CS11) | _BV(WGM13);                    //1:8 Prescaler

    OCR1A = PERIOD_CNT >> 1;                            //Set OCR1A to Period/2
    OCR1B = (PERIOD_CNT / 20) >> 1;                     //Initially set OCR1B to
        1ms on time / 2

    //Configure Timer 2 to delay 1ms
    TCCR2A = _BV(WGM21);            //CTC Mode
    TCCR2B = _BV(CS22);             //1:64 Prescaler
    TIMSK2 = _BV(OCIE2A);           //Enable compare interrupt
    OCR2A = CYCLES_2;               //Set OCR2A to the correct number of cycles

    sei();
}

/*************************************************************************/
/* Initialize ADC                                                        */
/*************************************************************************/
void InitADC(void)
```

```
{
    //Configure ADC
    ADMUX = _BV(REFS0) | _BV(ADLAR); //5V Vref, Left aligned, Channel 0
    ADCSRA = _BV(ADEN) | _BV(ADSC) | _BV(ADATE) | _BV(ADPS2) | _BV(ADPS1) | _BV
        (ADPS0);    //Enable ADC, Start Conversion,

                                                                //  Auto-Start,
                                                        1:128 Pre-scaler
    ADCSRB = 0; //Free Run Mode
}

/*************************************************************************/
/* Initialize EEPROM                                                     */
/*************************************************************************/
void InitEEPROM(void)
{
    ;
}

/*************************************************************************/
/* Initialize WDT                                                        */
/*************************************************************************/
void InitWDT(void)
{
    //Disable interrupts
    cli();

    //Reset WDT
    wdt_reset();

    //Clear WDRF in MCUSR
    MCUSR &= ~_BV(WDRF);

    //Set WDCE and WDE to start timed sequence
    WDTCSR |= _BV(WDCE) | _BV(WDE);

    //Setup WDTCSR to enable interrupt mode on WDT timeout
    // and start WDT with 4s timeout period by setting bit WDP3
    WDTCSR |= _BV(WDIE) | _BV(WDP3);

    //Set WDCE and WDE to start another timed sequence
    WDTCSR |= _BV(WDCE) | _BV(WDE);

    //Clear WDE to disable reset mode
    WDTCSR &= ~_BV(WDE);

    //Enable global interrupts
    sei();
}

void InitSorter(void)
{
    //Enable servo power and wait for transients
    Servo::Enable();
    _delay_ms(100);
```

```
    //Initialize Servos to nominal
    sorter.ServoZero.SetServo(NoMarble);
    sorter.ServoOne.SetServo(NoMarble);

    //Set sorter state to idle
    sorter.State = IdleState;
}

/*****************************************************************************/
/* Initialize LCD                                                            */
/*****************************************************************************/
void InitLCD(void)
{
    //Start the LCD as a 20x4 display
    lcd.begin(LINE_LEN, NUM_LINES);

    lcd.backlight();

    lcd.clear();
    lcd.home();
    lcd.print("Gibson-Millwood");
    _delay_ms(1000);

    lcd.setCursor(0, LINE_2);
    lcd.print("Marble Sorter");
    _delay_ms(1000);

    lcd.setCursor(0, LINE_3);
    lcd.print("V1.00");
    _delay_ms(1000);

    LoadingBar();

    lcd.clear();
    lcd.home();
}

/*****************************************************************************/
/* Loading Bar Animation                                                     */
/*****************************************************************************/
void LoadingBar(void)
{
    lcd.setCursor(0, LINE_4);

    for(int i = 0; i < LINE_LEN; i++)
    {
        lcd.write(0xFF);
        _delay_ms(100);
    }

    lcd.clear();
}

/*****************************************************************************/
/* Clear An Individual Line on the LCD                                       */
/*****************************************************************************/
```

```
void ClearLine(int line)
{
    lcd.setCursor(0, line);

    for(int i = 0; i < LINE_LEN; i++)
    {
        lcd.print(" ");
    }

    lcd.setCursor(0, line);
}

void PrintIdleScreen(void)
{
    lcd.clear();
    lcd.home();
    lcd.print("PRESS S -> Sort");

    lcd.setCursor(0, LINE_2);
    lcd.print("PRESS R -> Reset");

    lcd.setCursor(0, LINE_3);
    lcd.print("HOLD  S -> Recall");
}
```

```
/*****************************************************************************/
/* File: Global.h                                                            */
/* Author: Joe Gibson and Jesse Millwood                                     */
/* Date: 11/5/13                                                             */
/* Course: EGR 326                                                           */
/* Description: Global.h contains the definitions, structures, and           */
/*              other data types used throughout the project                 */
/*                                                                           */
/* Grand Valley State University, 2013                                       */
/*****************************************************************************/

#ifndef GLOBAL_H_
#define GLOBAL_H_

//General Definitions
#define PRESS_TIME      100             //Button press time in ms
#define HOLD_TIME       700             //Button hold time in ms
#define CYCLES_1        156             //Number of cycles at 1:1024 prescale for
    10ms delay on Timer 0
#define CYCLES_2        250             //Number of cycles at 1:64 prescale for
    1ms delay on Timer 2
#define SORT_THRESHOLD  10              //Number of marbles to be considered a
    successful sort
#define No_MORE_MARBLES_THRESHOLD 80    //Number of ms to wait when checking for
    more marbles

//EEPROM Addresses
#define MIN_ADDR            0x00    //Address for minutes
#define SEC_ADDR            0x01    //Address for seconds
#define BLACK_COUNT_ADDR    0x02    //Address for black count
#define WHITE_COUNT_ADDR    0x03    //Address for white count

//Pin Definitions
//OUTPUTS
#define LED_RED             _BV(3)              //Red LED on PD3
            (Digital Pin 3)
#define LED_GREEN           _BV(2)              //Green LED on PD2
            (Digital Pin 2)
#define LED_YELLOW          LED_RED | LED_GREEN //Yellow LED (Red and Green)


#define LCD_SDA             _BV(4)              //SDA on PC4
            (Analog Pin 4)
#define LCD_SCL             _BV(5)              //SCL on PC5
            (Analog Pin 5)

#define SERVO_EN            _BV(5)              //Servo Power Supply Enable on
    PD5     (Digital Pin 5)
#define SERVO_PWM           _BV(2)              //Servo PWM on PB2 (OC1B)
            (Digital Pin 10)
#define SWITCH_S0           _BV(4)              //Switch Select 0 on PD4
            (Digital Pin 4)
#define SWITCH_S1           _BV(2)              //Switch Select 1 on PB2
            (Digital Pin 10)

#define SENSOR_EN           _BV(0)              //Sensor Enable on PB0
```

```
            (Digital Pin 8)

//INPUTS
#define START_STOP_BTN      _BV(6)              //Start/Stop Button on PD6
            (Digital Pin 6)
#define RESET_BTN           _BV(7)              //Reset Button on PD7
            (Digital Pin 7)

#define SENSOR_0            _BV(0)              //Sensor 0 on PC0
            (Analog Pin 0)
#define SENSOR_1            _BV(1)              //Sensor 1 on PC1
            (Analog Pin 1)

//LCD Definitions
#define I2C_ADDRESS 0x27        //LCD I2C Address
#define EN          2           //Enable Pin
#define RW          1           //Read/Write Pin
#define RS          0           //Register Select Pin
#define D4          4           //Data [4..7]
#define D5          5
#define D6          6
#define D7          7
#define BL          3           //Backlight Pin
#define BL_POL      POSITIVE    //Backlight Polarity

#define LINE_LEN   20           //Length of line
#define NUM_LINES   4           //Number of lines
#define LINE_1      0           //Indexing of lines 1 to line 4
#define LINE_2      1
#define LINE_3      2
#define LINE_4      3

//Sorter Definitions
#define WHITE_THRESHOLD 8           //Threshold for a WHITE marble
#define BLACK_THRESHOLD 20          //Threshold for a BLACK marble

//Servo Definitions
#define PERIOD_CNT 40000        //Period cycle count for 20ms servo PWM period
#define SERVO_0 0               //Servo 0
#define SERVO_1 1               //Servo 1

//ADC Definitions
#define CHANNEL_0 0             //ADC Channel 0 (Sensor 0) on PC0
#define CHANNEL_1 1             //ADC Channel 1 (Sensor 1) on PC1

//Warning/Error Code Definitions
#define WAR_NO_MARBLE           -1              //No marble found. Not
    necessarily an error

#define ERR_NO_ERROR            0               //No error
#define ERR_WDT_TIMEOUT         -200            //Watchdog timer has timed out;
    at this point
                                                //  the total marble count should
                                                    be checked
#define ERR_INVALID_SERVO_ANGLE -201            //The servo angle was not between
    0 and 180 degrees
```

```
//Error Code Types
typedef int T_ErrorCode;        //Typedef for error code type

//Structures and Enumerations
typedef enum T_Color
{
    Red = LED_RED,
    Green = LED_GREEN,
    Yellow = LED_YELLOW,
    Off
}T_Color;

//Public Function Prototypes for main.cpp
void LoadingBar(void);
void ClearLine(int line);
void InitPortDirections(void);
void InitTimers(void);
void InitWDT(void);
void InitADC(void);
void InitEEPROM(void);
void InitSorter(void);
void InitLCD(void);
void PrintIdleScreen(void);

#endif /* GLOBAL_H_ */
```

```
/*****************************************************************************/
/* File: Marble.h                                                            */
/* Author: Joe Gibson and Jesse Millwood                                     */
/* Date: 11/5/13                                                             */
/* Course: EGR 326                                                           */
/* Description: Marble.h implements the Marble class, which contains         */
/*              the marble type data                                         */
/*                                                                           */
/* Grand Valley State University, 2013                                       */
/*****************************************************************************/

#ifndef MARBLE_H_
#define MARBLE_H_

#include "Global.h"

/*****************************************************************************/
/* Enumerations and Structures                                               */
/*****************************************************************************/
//Marble Type structure
typedef enum T_MarbleType
{
    Black,
    White,
    NoMarble
}T_MarbleType;

/*****************************************************************************/
/* Marble Class                                                              */
/*****************************************************************************/
class Marble
{
    /*************************************************************************/
    /* Private Members                                                       */
    /*************************************************************************/
    int Index;              //Index of marble: Marbles are indexed either 0 or 1,
        where
                            //  an index of 0 refers to the marble at the very
                                end
                            //  of the series of marbles

    T_MarbleType MarbleType;    //Type of marble: black marble, white marble, or
        no marble

    public :

    /*************************************************************************/
    /* Public Methods                                                        */
    /*************************************************************************/
    /*************************************************************************/
    /* Default Constructor                                                   */
    /*************************************************************************/
    Marble()
    {
        Index = 0;
        MarbleType = NoMarble;
```

```
    }

    /*****************************************************************************/
    /* Constructor to set Index                                                  */
    /*****************************************************************************/
    Marble(int index)
    {
        Index = index;
        MarbleType = NoMarble;
    }

    /*****************************************************************************/
    /* Default Destructor                                                        */
    /*****************************************************************************/
    ~Marble()   //Default destructor
    {
        /* */
    }

    /*****************************************************************************/
    /* Get marble type                                                           */
    /*****************************************************************************/
    T_MarbleType GetMarbleType(void)
    {
        return MarbleType;
    }

    /*****************************************************************************/
    /* Set marble type                                                           */
    /*****************************************************************************/
    void SetMarbleType(T_MarbleType marbleType)
    {
        MarbleType = marbleType;
    }

    /*****************************************************************************/
    /* Get index                                                                 */
    /*****************************************************************************/
    int GetIndex(void)
    {
        return Index;
    }

    /*****************************************************************************/
    /* Set index                                                                 */
    /*****************************************************************************/
    void SetIndex(int index)
    {
        Index = index;
    }
};

#endif /* MARBLE_H_ */
```

```
/*****************************************************************************/
/* File: Servo.h                                                             */
/* Author: Joe Gibson and Jesse Millwood                                     */
/* Date: 11/5/13                                                             */
/* Course: EGR 326                                                           */
/* Description: Servo.h implements the Servo class, which contains the       */
/*              data and methods associated with controlling servos          */
/*                                                                           */
/* Grand Valley State University, 2013                                       */
/*****************************************************************************/

#ifndef SERVO_H_
#define SERVO_H_

#include "Global.h"
#include "Marble.h"

/*****************************************************************************/
/* Servo Class                                                               */
/*****************************************************************************/
class Servo
{
    /*************************************************************************/
    /* Private Members                                                       */
    /*************************************************************************/
    int Index;

    /*************************************************************************/
    /* Private Methods                                                       */
    /*************************************************************************/
    /*************************************************************************/
    /* Set servo to an angle in degrees                                      */
    /*************************************************************************/
    T_ErrorCode SetServoAngle(double degrees)
    {
        //BYPASSING SWITCH
        /*
        //Select the correct line to switch servo
        if(this->Index == 0)
        {
            PORTD |= SWITCH_S0;
            PORTB &= ~SWITCH_S1;
        }

        if(this->Index == 1)
        {
            PORTD &= ~SWITCH_S0;
            PORTB |= SWITCH_S1;
        }
        */

        //Declare an offset variable to attempt to reach
        // max swing
        int offset = 0;

        //Check for angles outside of range
```

```
    if((degrees < 0) || degrees > 180)
    {
        //Only valid for 0 to 180 degrees
        return ERR_INVALID_SERVO_ANGLE;
    }

    //Check to see if necessary
    //Apply negative offset for 0 degrees
    if(degrees == 0)
    {
        offset = -700;
    }

    //Apply positive offset for 180 degrees
    if(degrees == 180)
    {
        offset = 550;
    }

    //Convert from  0 to 180 degrees to 1.0 to 2.0ms Ton
    OCR1B = (int)(((((degrees / 180) + 1.0) / 20.0) * PERIOD_CNT) + offset)>>
        1;

    return ERR_NO_ERROR;
}

public :

/*****************************************************************************/
/* Public Methods                                                            */
/*****************************************************************************/
/*****************************************************************************/
/* Default Constructor                                                       */
/*****************************************************************************/
Servo()
{
    this->Index = 0;
}

/*****************************************************************************/
/* Constructor with Index                                                    */
/*****************************************************************************/
Servo(int index)
{
    this->Index = index;
}

/*****************************************************************************/
/* Default Destructor                                                        */
/*****************************************************************************/
~Servo()
{

}

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

```
/* Get index                                                                  */
/****************************************************************************/
int GetIndex(void)
{
    return this->Index;
}

/****************************************************************************/
/* Set index                                                                  */
/****************************************************************************/
void SetIndex(int index)
{
    this->Index = index;
}

/****************************************************************************/
/* Enable servo power                                                         */
/****************************************************************************/
static void Enable(void)
{
    PORTD |= SERVO_EN;
}

/****************************************************************************/
/* Disable servo power                                                        */
/****************************************************************************/
static void Disable(void)
{
    PORTD &= ~SERVO_EN;
}

/****************************************************************************/
/* Set servo based on the marble type sensed                                  */
/****************************************************************************/
T_ErrorCode SetServo(T_MarbleType marbleType)
{
    if(marbleType == White)
    {
        if(this->Index == 0)
        {
            this->SetServoAngle(180);
        }
        else
        {
            this->SetServoAngle(0);
        }
    }

    else if(marbleType == Black)
    {
        if(this->Index == 0)
        {
            this->SetServoAngle(0);
        }
        else
        {
```

```
                this->SetServoAngle(180);
            }
        }

        else
        {
            //Set servo to 90 degrees
            this->SetServoAngle(90);
        }

        return ERR_NO_ERROR;
    }
};



#endif /* SERVO_H_ */
```

```
/*****************************************************************************/
/* File: Sorter.h                                                            */
/* Author: Joe Gibson and Jesse Millwood                                     */
/* Date: 11/5/13                                                             */
/* Course: EGR 326                                                           */
/* Description: Sorter.h implements the Sorter class, which contains         */
/*                                                                           */
/*                                                                           */
/* Grand Valley State University, 2013                                       */
/*****************************************************************************/

#ifndef SORTER_H_
#define SORTER_H_

#include <util/delay.h>
#include <avr/eeprom.h>
#include <string.h>
#include "Global.h"
#include "Marble.h"
#include "Servo.h"

/*****************************************************************************/
/* Enumerations and Structures                                               */
/*****************************************************************************/
//State enumeration
typedef enum T_State
{
    IdleState,
    SortState,
    RecallState,
    ResetState,
    TestState
}T_State;

//Button Action enumeration
typedef enum T_ButtonAction
{
    None = 0,
    Press = 1,
    Hold = 2
}T_ButtonAction;

//Marble Count structure
typedef struct T_MarbleCount
{
    int BlackCount;
    int WhiteCount;
    int TotalCount;

    //Constructor
    T_MarbleCount()
    {
        this->BlackCount = 0;
        this->WhiteCount = 0;
        this->TotalCount = 0;
    }
```

```
    //Destructor
    ~T_MarbleCount()
    {

    }

}T_MarbleCount;

/*****************************************************************************/
/* Sorter Class                                                              */
/*****************************************************************************/
class Sorter
{
    /*************************************************************************/
    /* Private Methods                                                       */
    /*************************************************************************/
    /*************************************************************************/
    /* Set the ADC to a channel                                              */
    /*************************************************************************/
    T_ErrorCode SelectADCChannel(int channel)
    {
        //Set corresponding channel bits
        ADMUX |= _BV(REFS0) | _BV(ADLAR) | channel;

        return ERR_NO_ERROR;
    }

    /*************************************************************************/
    /* Check the sensor on the given channel                                 */
    /*************************************************************************/
    T_ErrorCode CheckSensorOnChannel(int channel, Marble &marble)
    {
        //Select the channel
        SelectADCChannel(channel);

        //Check for White Marble
        if(ADCH <= WHITE_THRESHOLD)
        {
            //Set MarbleType
            marble.SetMarbleType(White);

            return ERR_NO_ERROR;
        }

        //Check for Black Marble
        else if((ADCH >= WHITE_THRESHOLD) && (ADCH <= BLACK_THRESHOLD))
        {
            //Set MarbleType
            marble.SetMarbleType(Black);

            return ERR_NO_ERROR;
        }

        //No Marble
        else
```

```
    {
        //Set MarbleType
        marble.SetMarbleType(NoMarble);

        return WAR_NO_MARBLE;
    }
}

/*************************************************************************/
/* Update the MarbleCount structure                                      */
/*************************************************************************/
void UpdateCount(T_MarbleType marbleType)
{
    if(marbleType == Black)
    {
        this->MarbleCount.BlackCount++;
        this->MarbleCount.TotalCount++;

        //Write to EEPROM
        eeprom_update_byte((uint8_t *)BLACK_COUNT_ADDR, (uint8_t)(this->
            MarbleCount.BlackCount));
    }

    if(marbleType == White)
    {
        this->MarbleCount.WhiteCount++;
        this->MarbleCount.TotalCount++;

        //Write to EEPROM
        eeprom_update_byte((uint8_t *)WHITE_COUNT_ADDR, (uint8_t)(this->
            MarbleCount.WhiteCount));
    }

    //Write time to EEPROM
    eeprom_update_byte((uint8_t *)MIN_ADDR, (uint8_t)(this->MinutesElapsed));
    eeprom_update_byte((uint8_t *)SEC_ADDR, (uint8_t)(this->SecondsElapsed));
}

public :

/*************************************************************************/
/* Public Members                                                        */
/*************************************************************************/
bool MoreMarbles;                       //Flag for whether there are more
    marbles to sort

bool Flag;                              //General flag

bool WDTFlag;                           //Watchdog Timer flag

bool FlashLED;                          //Flash Red LED flag

bool ButtonActionReady;                 //Flag for whether a button action is
    ready to be processed

T_ButtonAction ResetButtonActon;        //Reset button action
```

```
T_ButtonAction StartStopButtonAction;   //Start/Stop button action

T_State State;                          //Sorter state: Idle, Sort, Recall,
    Reset

T_ErrorCode Error;                      //Error code

T_MarbleCount MarbleCount;              //Count for number of black, white,
    and total marbles sorted

Marble MarbleZero;                      //Marble at position zero
Marble MarbleOne;                       //Marble at position one
Marble NullMarble;                      //Null marble

Servo ServoZero;                        //Servo at position zero
Servo ServoOne;                         //Servo at position one

int MinutesElapsed;                     //Minutes elapsed
int SecondsElapsed;                     //Seconds elapsed
int TenthsOfSecondsElapsed;             //Tenths of seconds elapsed

/*****************************************************************************/
/* Public Methods                                                            */
/*****************************************************************************/
/*****************************************************************************/
/* Default Constructor                                                       */
/*****************************************************************************/
Sorter()
{
    //Initialize sorter members
    this->MoreMarbles = true;
    this->Flag = false;
    this->WDTFlag = false;
    this->FlashLED = false;
    this->State = IdleState;
    this->Error = ERR_NO_ERROR;
    this->MarbleCount.BlackCount = 0;
    this->MarbleCount.WhiteCount = 0;
    this->MarbleCount.TotalCount = 0;
    this->MinutesElapsed = 0;
    this->SecondsElapsed = 0;
    this->TenthsOfSecondsElapsed = 0;

    this->MarbleZero.SetIndex(0);
    this->MarbleOne.SetIndex(1);
    this->NullMarble.SetIndex(-1);
    this->ServoZero.SetIndex(0);
    this->ServoOne.SetIndex(1);
    this->ServoZero.SetServo(NoMarble);
    this->ServoOne.SetServo(NoMarble);
}

/*****************************************************************************/
/* Default Destructor                                                        */
/*****************************************************************************/
~Sorter()
```

```
{
    /* */
}

/*****************************************************************************/
/* Set RGB LED color                                                         */
/*****************************************************************************/
void SetLEDColor(T_Color color)
{
    //Clear the LED color
    PORTD &= ~(LED_RED | LED_GREEN);

    //Set the LED color if necessary
    if(color != Off)
    {
        PORTD |= color;
    }
}

/*****************************************************************************/
/* Perform one sorting cycle                                                 */
/*****************************************************************************/
T_ErrorCode Sort(void)
{
    /***********************************/
    /* DEBUG: ONLY SORTING ONE MARBLE */
    /***********************************/
    //Declare error codes
    T_ErrorCode errorCodeChannelZero = ERR_NO_ERROR;
    //T_ErrorCode errorCodeChannelOne = ERR_NO_ERROR;

    //Check sensor 0
    errorCodeChannelZero = CheckSensorOnChannel(CHANNEL_0, MarbleZero);

    //Check sensor 1
    //errorCodeChannelOne = CheckSensorOnChannel(CHANNEL_1, MarbleOne);

    //Update counts
    UpdateCount(MarbleZero.GetMarbleType());
    //UpdateCount(MarbleOne.GetMarbleType());

    //IF SORTING ONE MARBLE
    //Discern if channel detected no marble
    if(errorCodeChannelZero == WAR_NO_MARBLE)
    {
        //this->MoreMarbles = false;
        return WAR_NO_MARBLE;
    }

    //Marble was detected
    //this->MoreMarbles = true;

    //Set servo to sort marble based on type
    ServoZero.SetServo(MarbleZero.GetMarbleType());

    //Delay and return to nominal
```

```
    _delay_ms(500);
    ServoZero.SetServo(NoMarble);

    //Disable servo power
    //Servo::Disable();

    //IF SORTING TWO MARBLES
    //Discern if channel one or both channels detected no marble
    /*
    if(errorCodeChannelOne == WAR_NO_MARBLE)
    {
        //One more marble to sort on channel zero
        if(errorCodeChannelZero != WAR_NO_MARBLE)
        {
            Servo::Enable();
            _delay_ms(10);

            ServoZero.SetServo(MarbleZero.GetMarbleType());

            _delay_ms(10);
            Servo::Disable();

            this->MoreMarbles = false;
            return WAR_NO_MARBLE;
        }

        //No more marbles to sort
        else
        {
            this->MoreMarbles = false;
            return WAR_NO_MARBLE;
        }

    }

    //Enable servo power supply
    Servo::Enable();
    _delay_ms(10);

    ServoZero.SetServo(MarbleZero.GetMarbleType());
    ServoOne.SetServo(MarbleOne.GetMarbleType());

    //Disable servo power supply
    _delay_ms(10);
    Servo::Disable();
    */

    return ERR_NO_ERROR;
}

/*****************************************************************************/
/* Check to see if there are any more marbles to sort                        */
/*****************************************************************************/
T_ErrorCode CheckForMoreMarbles(void)
{
    return CheckSensorOnChannel(CHANNEL_0, NullMarble);
```

```
    }

    /*****************************************************************************/
    /* A button action has been completed                                        */
    /*****************************************************************************/
    void ButtonActionCompleted(void)
    {
        this->StartStopButtonAction = None;
        this->ResetButtonActon = None;
        this->ButtonActionReady = true;
    }

    /*****************************************************************************/
    /* Set reset button action                                                   */
    /*****************************************************************************/
    void SetResetButtonAction(T_ButtonAction action)
    {
        this->ResetButtonActon = action;
    }

    /*****************************************************************************/
    /* Get reset button action                                                   */
    /*****************************************************************************/
    T_ButtonAction GetResetButtonAction(void)
    {
        if(!(this->ButtonActionReady))
        {
            return None;
        }

        return this->ResetButtonActon;
    }

    /*****************************************************************************/
    /* Set start/stop button action                                              */
    /*****************************************************************************/
    void SetStartStopButtonAction(T_ButtonAction action)
    {
        this->StartStopButtonAction = action;
    }

    /*****************************************************************************/
    /* Get start/stop button action                                              */
    /*****************************************************************************/
    T_ButtonAction GetStartStopButtonAction(void)
    {
        if(!(this->ButtonActionReady))
        {
            return None;
        }

        return this->StartStopButtonAction;
    }
};
```

```
#endif /* SORTER_H_ */
```