University of Victoria

Department of Mechanical Engineering

MECH 458/554 - Mechatronics

Design and Implementation of an Inspection System Using an ATMEL AVR Microcontroller

Final Design Report

Rushi Nathvani
V00835408

Sreejith Munthikodu
V00903589

April 16, 2018

Abstract

Modern embedded systems have become a vital part of our daily life with their applications ranging from portable devices such as digital watches and digital cameras to large and complex installations such as traffic lights, washing machines, and automotive systems. The objective of this design project is to design a high-performance inspection system, implement the proposed design using an ATMEL AVR microcontroller and demonstrate the functionality. The inspection system is required to sort given number of items based on the visual and material characteristics in a minimum amount of time. The project required to use the advanced microcontroller functionalities such as Interrupts, Timers, Pulse Width Modulation (PWM) and Analogue to Digital Converters (ADC). Also, advanced programming skills were required to implement functions such as First In First Out (FIFO) dynamic linked list using C-language. Moreover, the project demanded using various sensors such as inductance and reflectance sensors, and actuators such as a DC motor to drive the conveyor belt and a stepper motor to rotate the bin to collect the sorted items.

Acknowledgments

This highly challenging embedded system design project would not have been possible without the ample guidance of course instructor Mr. Yuanye Chen and lab instructor Mr. Patrick Chang. The concepts required to complete the project were effectively introduced in sequence in the class and the same was implemented during the weekly lab sessions. So by the end of lab 4, we could gain a lot of knowledge about the microcontrollers and their functionalities and we were in a position to start our design project with confidence.

We are highly grateful to Mr. Patrick Chang for patiently listening to all our queries and guiding us to find the right answers from the manual. He sacrificed his weekends and opened the lab on all the weekends which helped us to spend more time on our project and complete it in time. We would like to thank the University of Victoria and the Department of Mechanical Engineering for offering such a valuable course, for providing us the technical manuals and access to the Mechatronics lab and for providing other infrastructures such as the library, which helped us to successfully complete this design project. Also, we would like to extend our gratitude to all the Teaching Assistants for helping us to complete our lab works and for extending their support in evaluating assignments, supervising the exams and for evaluating the demonstration of the inspection system. Finally, we would like to thank our coursemates for being kind enough to share the limited available stations and other resources at the lab.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem Statement

The objective of this design project is to design a high-performance inspection system, implement the proposed design using an ATMEL AVR microcontroller and demonstrate its functionality. The inspection system is required to sort four types of cylindrically shaped objects which are of black plastic, a white plastic, a steel and an aluminum material. For the final performance testing, the inspection system needs to sort 48 of such items into their respective bins in less than 60 seconds. Also, there has to be a pause button which pauses the inspection system and a ramp-down button which stops the system after sorting all the items on the conveyor. The details such as a number of each item sorted and unsorted items on the belt are to be displayed using LEDs or optionally using LCD display.

1.2 Design Purpose and Overview

This design project is to sort 48 numbers of cylindrical shaped items which are black plastic, white plastic, steel and aluminum in less than 60 seconds. Additionally, a pause button and a system ramp-down button is required to interrupt the system. Microcontroller AT90USB 1287 is used as the heart of the sorting system. A conveyor belt driven by a DC motor is used to carry the items to be sorted through various sensor stations. The sensor stations are used to identify the correct type of the item. Finally, a stepper motor driven bin is used to collect the sorted items while it falls off the conveyor belt. The various components used in the design is described below.

1.2.1 AT90USB 1287

The high-performance, low-power Microchip 8-bit AVR microcontroller combines 128KB ISP flash memory with read-while-write capabilities, 8KB SRAM, 48 general purpose I/O lines, 32 general purpose working registers, real-time counter, four flexible timer/counters with compare modes and PWM, USB 2.0 low-speed and full-speed On-The-Go (OTG) host/device, an 8-channel 10-bit A/D converter, JTAG (IEEE 1149.1 compliant) interface for on-chip debugging, and six software selectable power saving modes.

1.2.2 Conveyor System

A conveyor belt is used to transport the items that are to be sorted. It is driven by a DC motor and the speed of the conveyor is controlled by the PWM signal. Guides are provided at the loading point of the conveyor

which aligns the loaded items on the belt and ensures that there is sufficient gap between adjacent items for the sensors to read correctly. The belt can be started or stopped as required but the speed of the belt is maintained constant during the operation.

1.2.3 Inspection Stations

As the items travel on the conveyor belt, they pass through various sensor stations which determine the material and visual characteristics of the items. Three sensors are used in this project to determine the type of the items.

- Pre-reflectance Optical Sensor:

  It is a digital sensor which is active high. It is used to identify if an item is in the range of the reflectance sensor. Only when an object is between the pre-reflectance optical sensor and the reflectance sensor, the analog signal is measured and converted into a digital value.

- Reflectance Sensor:

  It is an analog sensor which measures the reflectance of the item in its range. Baes on the visual and material characteristics, different types of items have their unique range of reflectance value which can be used to identify the type of the item.

1.2.4 Sorting System

There is an exit optical sensor placed near the end of the conveyor belt system. If an item is detected by the exit optical sensor, which is an active low digital sensor, it indicates that the item has reached the end of travel. There is a rotating bin below the end of travel which can collect the falling items. If the item at the end of travel doesn't match the respective tray bin, the belt can be stopped and the tray can be rotated to the correct bin before starting the belt again and allowing the item to fall down. The location of the bin is identified by using a Hall-Effect sensor, which sets the bin to its home position before the start of the inspection. The positions of other bins on the tray are defined relative to the home position. The two sensors used in the sorting system can be summarized as below:

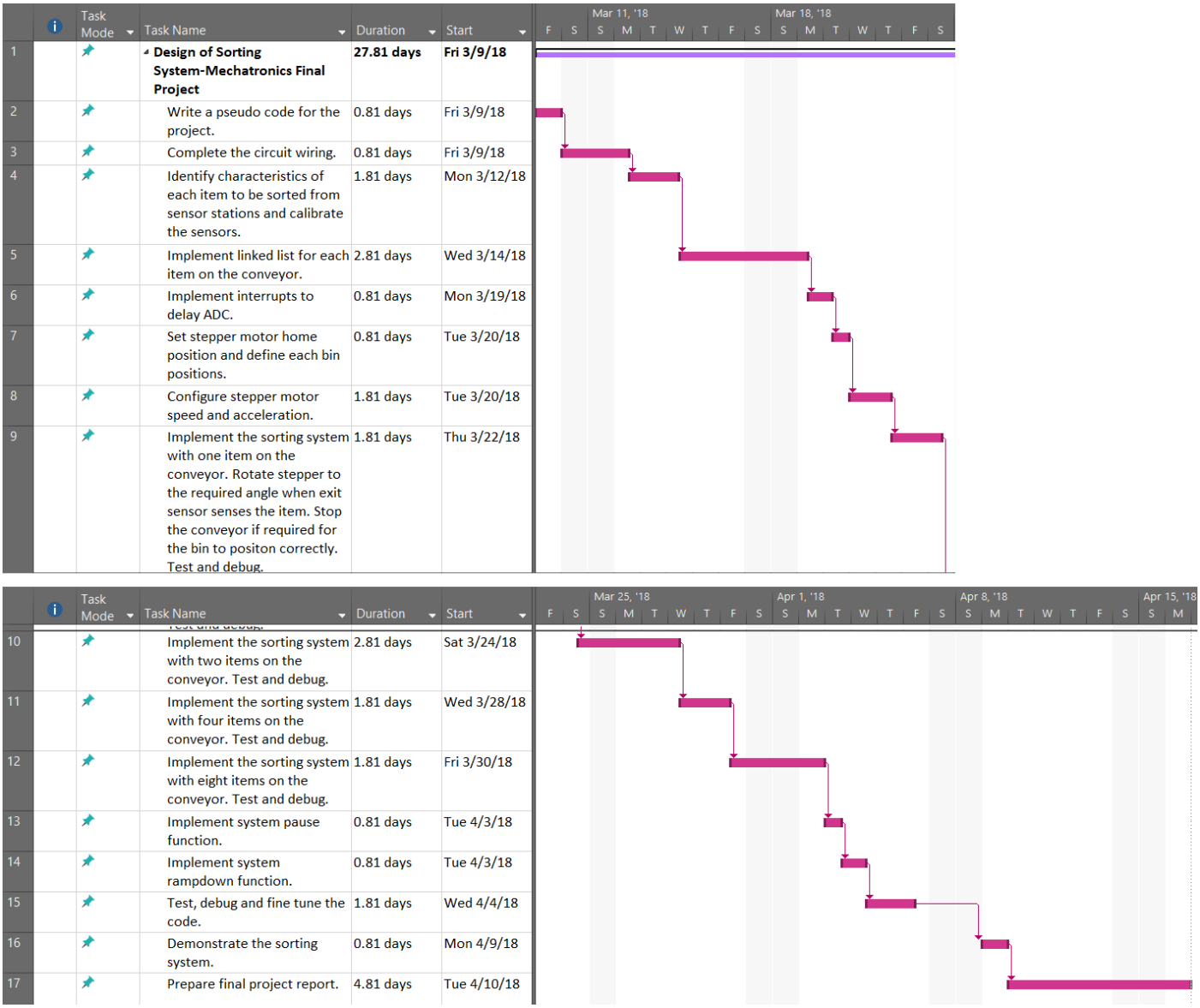
- Exit Optical Sensor

  It is an active low digital sensor which indicates that the item has reached the end of travel.

- Hall-Effect Sensor

  It is an active low digital sensor that sets the sorting tray to its home position.

2 Project Timeline



	Task Mode	Task Name	Duration	Start
1		**Design of Sorting System-Mechatronics Final Project**	**27.81 days**	**Fri 3/9/18**
2		Write a pseudo code for the project.	0.81 days	Fri 3/9/18
3		Complete the circuit wiring.	0.81 days	Fri 3/9/18
4		Identify characteristics of each item to be sorted from sensor stations and calibrate the sensors.	1.81 days	Mon 3/12/18
5		Implement linked list for each item on the conveyor.	2.81 days	Wed 3/14/18
6		Implement interrupts to delay ADC.	0.81 days	Mon 3/19/18
7		Set stepper motor home position and define each bin positions.	0.81 days	Tue 3/20/18
8		Configure stepper motor speed and acceleration.	1.81 days	Tue 3/20/18
9		Implement the sorting system with one item on the conveyor. Rotate stepper to the required angle when exit sensor senses the item. Stop the conveyor if required for the bin to positon correctly. Test and debug.	1.81 days	Thu 3/22/18
10		Implement the sorting system with two items on the conveyor. Test and debug.	2.81 days	Sat 3/24/18
11		Implement the sorting system with four items on the conveyor. Test and debug.	1.81 days	Wed 3/28/18
12		Implement the sorting system with eight items on the conveyor. Test and debug.	1.81 days	Fri 3/30/18
13		Implement system pause function.	0.81 days	Tue 4/3/18
14		Implement system rampdown function.	0.81 days	Tue 4/3/18
15		Test, debug and fine tune the code.	1.81 days	Wed 4/4/18
16		Demonstrate the sorting system.	0.81 days	Mon 4/9/18
17		Prepare final project report.	4.81 days	Tue 4/10/18

Figure 1: Project timeline

3 Methods and Design Approach

The design demands a careful coordination of the DC motor, different sensor stations and the stepper motor for correctly sorting the items on the conveyor. The items are loaded onto the conveyor belt which takes them through the sensor stations. The first sensor station detects the type of the item and the exit sensor station make sure that the tray is positioned correctly before the item is allowed to fall down into the bin.

The conveyor belt is driven by a DC motor using PWM signal. The speed of the DC motor is controlled by the width of the PWM signal. The speed is optimized for minimum traveling time with the time available for the ADC as a constraint. Too fast belt speed resulted in an error in the ADC values and hence resulted in

identifying incorrect item type. Also, at very high speed, when the belt is stopped at the exit sensor, due to inertia item may slip and fall into the incorrect bin.

The reflectance sensor is primarily used to identify the type of the item. It is an analog sensor and hence Analogue to Digital Conversions was required. A 10 bit ADC resolution is used in this project. Since the ADC consumes considerable process cycles, the pre-reflectance optical sensor is used to ensure that ADC is started only when an item is present in the range of the reflectance sensor. As the items move through the belt and reach the first sensor station, which is the pre-reflectance optical sensor and the reflectance sensor, ADC is started and the minimum value of the ADC result is stored. As the item leaves the first sensor station and travels on its way to the exit sensor, the MCU compares the minimum ADC result value with that of the range of each item to be sorted and identifies the object type. A dynamic FIFO linked list is created and the object type is enqueued.

When the object reaches the exit optical sensor station, the first item in the linked list is dequeued and the object type is read. The object type is counted using a different variable to update the total number of each item sorted and to identify the number of items remaining on the belt. The object type of the dequeued item is compared with the current tray position. If the tray position matches to that of the object type the item is allowed to fall into the correct bin without stopping the conveyor belt. If the current bin does not match to the object type, the belt is stopped and a command is given to the stepper motor for rotating the tray to the correct bin. Once the correct bin is positioned, the belt is started and the object is allowed to fall into the correct bin.

The Hall-Effect sensor sets the home position of the bin which is black. Other bin positions are defined relative to the home position and the stepper is turned 90 degrees clockwise, 90 degrees counterclockwise or 180 degrees as required to set the bin to the correct position before the object is dropped.

The design is mainly interrupt driven. A large chunk of the codes is executed inside the interrupts. First interrupt is triggered when the item is detected by the pre-reflectance optical sensor. The second interrupt is triggered when an ADC is completed. A third interrupt is triggered when the object is detected by the exit optical sensor. Fourth and the fifth interrupt are used for the system pause button and system ramp-down button respectively.

For the system pause, a push button triggered an interrupt on rising edge is used. When the pushbutton is pressed and released, the interrupt stops the conveyor belt immediately and displays the information such as a number of each object type sorted and the total number of items remaining on the belt. When the button is pressed and released again, the belt resumes and the system continues to sort items.

Another pushbutton is used for the system ramp-down function. When the interrupt is triggered by the

pushbutton, the systems starts counting and after a set delay the belt is stopped and the sorted item information is displayed on the LCD screen. The delay in stopping the conveyor belt ensures that all items remaining on the belt while the system ramp-down is initiated, are sorted before shutting down the system.

4 System Technical Details

4.1 Block Diagram

TODO

4.2 Circuit Diagram

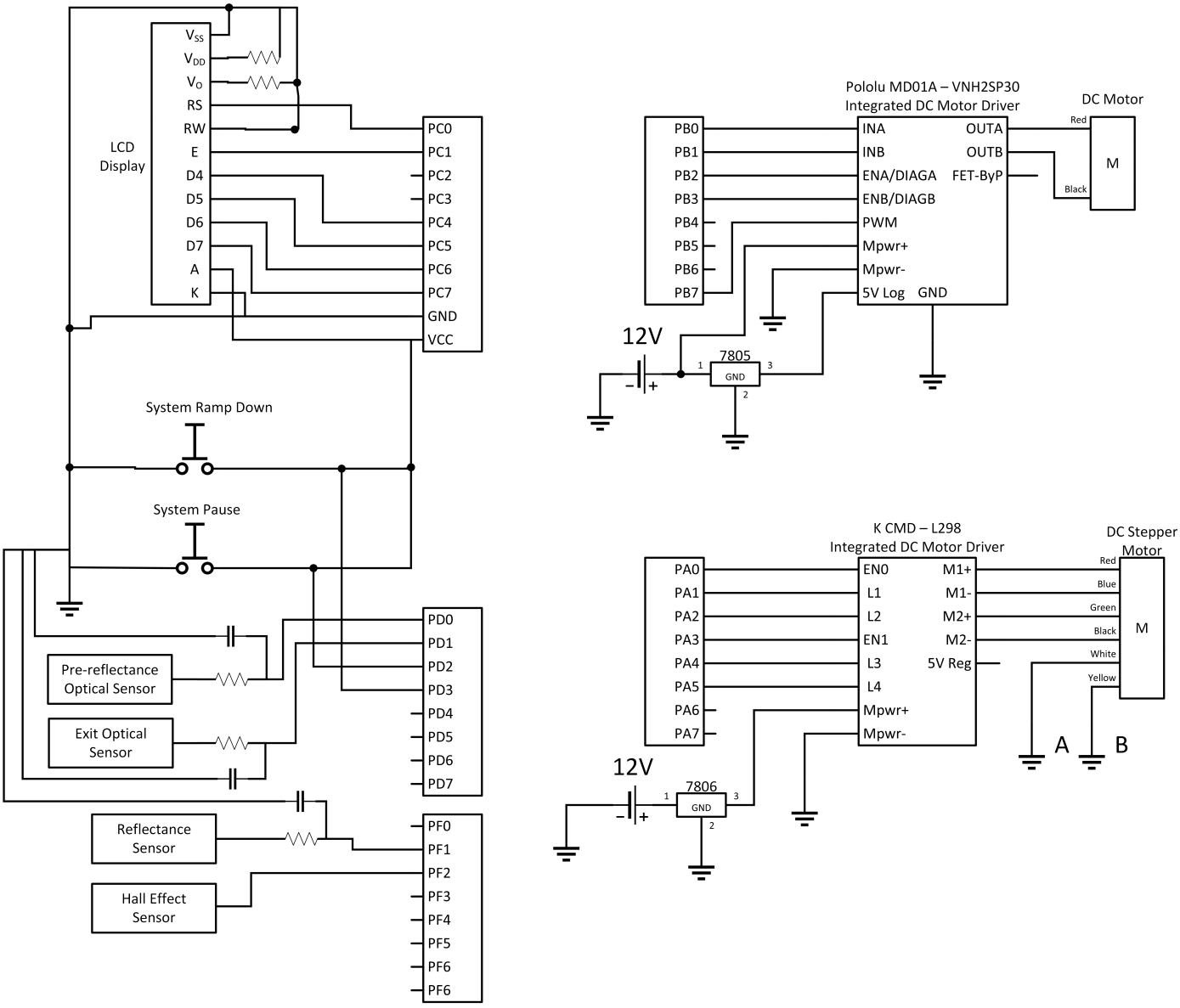


Figure 2: Circuit diagram of complete system

4.3 System Algorithm/Flowchart

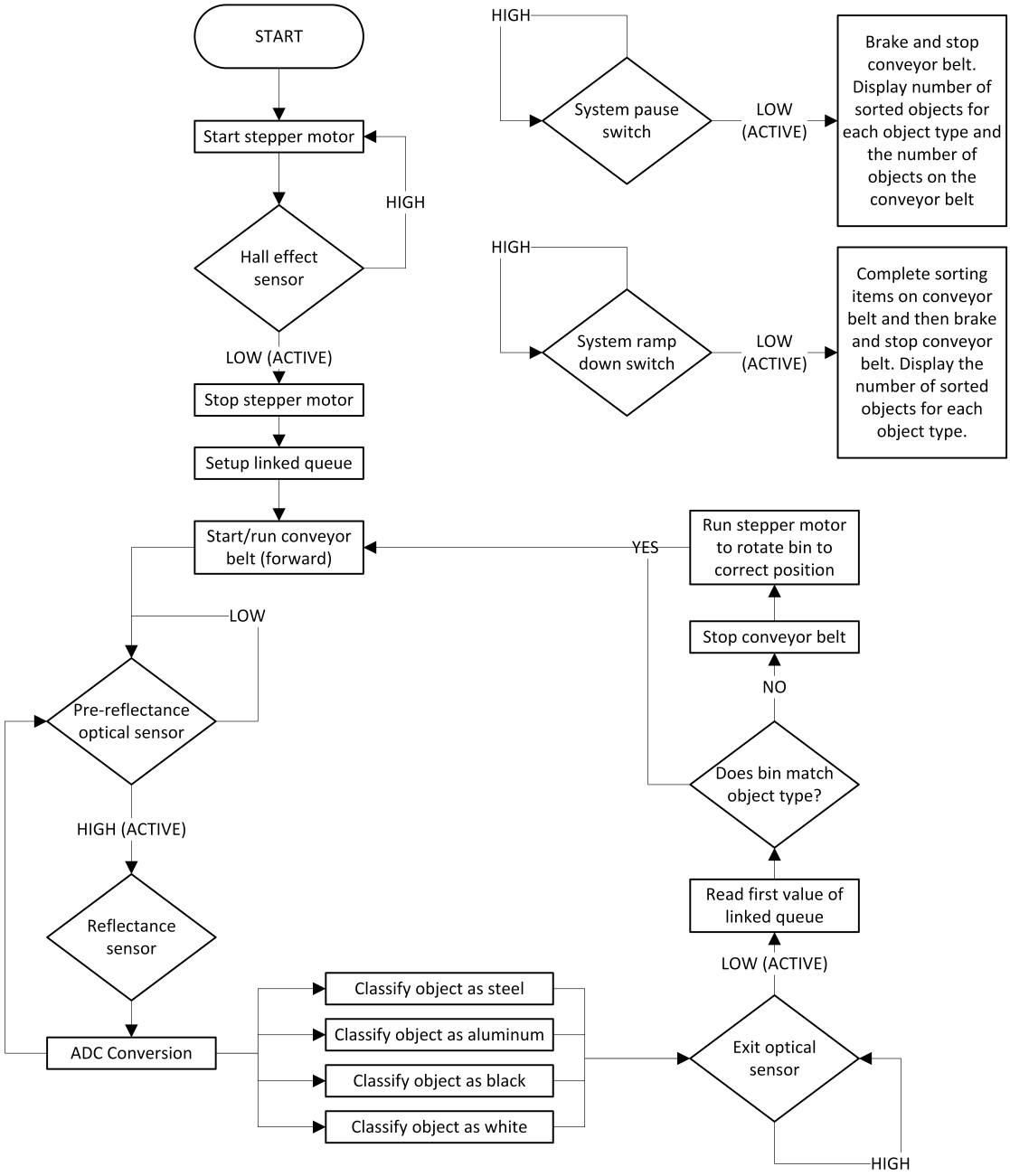


Figure 3: System algorithm/flowchart

4.4 Technical Description

The technical description of the various components and functionalities used in this design are described below:

- DC Motor

  TODO

- Pulse Width Modulation (PWM)

  TODO

- Timer Delay

  TODO

- Stepper Motor

  TODO

- Analog to Digital Conversion (ADC)

  TODO

- Pre-Reflectance Optical Sensor

  TODO

- Reflectance Sensor

  TODO

- Exit Optical Sensor

  TODO

- Hall-Effect Sensor

  TODO

- System Pause

  TODO

- System Ramp-Down

  TODO

5 System Performance Specifications

The major system performance specifications of the inspection system designed in this project are as follows:

- The final demonstration successfully proved that all the 48 items can be sorted in 45 seconds without any errors.
- Up to eight objects can be sorted at a time. The system will not give accurate results if both the exit sensor and the pre-reflectance optical sensor detects items in their range at the same time. Hence sufficient gap must be maintained before loading another batch of eight items onto the conveyor.
- The conveyor speed cannot be increased beyond the set speed. If a higher speed is used the ADC results are inaccurate sometimes and also there is a chance of the item slipping due to inertia when the belt is

stopped at the exit sensor. This may lead to the object falling into the wrong bin even if it is identified correctly by the MCU.

- The stepper uses a very slow acceleration. A faster rate of acceleration was causing the item falling on the edge of the bin. However, it was identified during the final demonstration that the tray was slightly misaligned when the bin is set to the home position. Manually correcting the position solved the issue. However, had this been detected earlier, a faster stepper speed could have been used which could have reduced the time required to sort 48 pieces to possibly less than 40 seconds.
- The inspection system is to be calibrated each time before startup as the ADC can give different results depending on the time of the day and ambient temperature. It will not cause any issue in identifying between steel, aluminum, and plastic. But, it may give an error in differentiating between black and white item.

6 Testing and Calibration Procedures

Extensive testing and calibration were done on the system prior to the final demonstration to make sure that the system sorts all 48 items with zero errors in the minimum possible time. Calibration of the reflectance proved to be the most critical task in the set up of the system as an error would result in classifying the incorrect object type. We used LCD display for the final demonstration to display the details of items sorted. However, for the calibration of the analog sensor, we used 6 LEDs on PORTC and the two inbuilt LEDs on PORTD as this was only one-time activity before start-up of the system. A separate code is maintained for calibration. The 10-bit ADC values are stored in an excel file for each item type and after multiple trails, the range of ADC value corresponding to each object type is identified and updated on the main code. The ADC range of steel, aluminum, and plastics was sufficiently apart and hence the system never gave any errors in differentiating between them. However, the range of black plastic and white plastic are very close and hence frequent calibrations were required to make sure that the system identifies item types correctly.

Due to noise created by the DC motor and stepper motor, the ADC and other sensors were sometimes giving false errors and sometimes triggered false interrupts. When discussed with Mr. Patrick Chang, the lab instructor, bout these issues, he explained what noise and other interferences can disrupt the system performance and he also explained to us how to remove such noises. We then used filters to filter out the noises from the sensors which then started giving correct readings without any random errors.

The optimization of the stepper and DC motor speed was also a challenging task. We tried various acceleration values for the stepper. However, at high speeds, the stepper was so fast that the items were falling on the

edges. However, later it was found that the tray home position was slightly misaligned. This problem was identified and corrected only during the demonstration and hence we could not use a fast stepper acceleration which compromised on the time required to sort the pieces. Different conveyor speed was tried by varying the PWM signal. Finally, an optimum speed of the belt was identified. Beyond this speed, items were slipping off the belt at the exit sensor when the belt was stopped and also ADC was giving less accurate readings due to less available time for analog to digital conversions.

7 Limitations and System Tradeoffs

The major limitations and tradeoffs of the design are summarized below:

- The major limitation of the system is that it cannot sort items continuously. Items need to be loaded in batches of 8 for the inspection system to work accurately. This is due to a limitation in the code which is unable to handle when two interrupts were triggered at the same- one by the exit optical sensor to deque the linked list and one by the pre-reflectance optical sensor to initiate a new link in the FIFO linked list. This significantly increases the time required to sort the items.
- The inspection system uses only reflectance sensor for differentiating different items. Though it is sufficient for identifying object type between steel, aluminum and plastic materials, it is not always very accurate in differentiating between white plastic and black plastic items as their reflectance values are much closer. Hence frequent calibrations were required for the system to work 100% accurately.
- Very high DC motor drive speed could not be used due to the less available time for the ADC and also due to the items slipping off the belt when stopped at the exit sensor. This compromises on the time required for sorting the items as the travel time is higher.
- The acceleration of the stepper is not very impressive. A higher acceleration and hence speed of stepper turns were causing the items to fall on the edges of the bin. This could have been avoided by carefully calibrating the tray to its home position.

8 Novel System Additions

8.1 LCD Display

TODO

9 Experience and Recommendations

10 Conclusion

The final design of the inspection system successfully met all the requirements set for the project. All the 48 pieces were sorted in 45 seconds without any errors. System pause function and system ramp-down functions were successfully demonstrated. Also, all the details of items sorted and items remaining on the belt were displayed correctly on the LCD display.

The design of this embedded system is very challenging and at the same time highly rewarding. The project demanded the use of advanced programming skills and a very good knowledge of microcontrollers. The whole course was so well designed that every day we learned something new which helped us to complete the project.

The confidence that the successful demonstration of this project gave us is enormous. We got a chance to learn about the microcontroller architecture, Analogue to Digital Conversion, Pulse Width Modulation, Interrupts, Timers, implementation of a linked list in C using pointers, various sensors, and actuators. We are confident that we have learned enough from this course on how to interface between sensors, actuators, and MCU and on how to configure MCU register for using the various functionalities, to take up more challenging tasks using microcontrollers. The most impressive aspect of this project was learning tonnes of new things by doing something very interesting and fun.

11 References

1. User manual for ATMEL 8-bit AVR Microcontrollers with 64/ 128K Bytes of ISP Flash and USB controller.
2. AT90USBKey Hardware user guide.

Appendix A Summary of Contributions

Task	Completed by
Stepper motor acceleration & deceleration	Sreejith
LCD display	Rushi
Testing & debugging	Rushi & Sreejith
Project timeline	Sreejith
Block diagram, circuit diagram, & system algorithm/flowchart	Rushi

Table 1: Summary of contributions

Appendix B Source Code

```
#include <avr/interrupt.h>
#include <avr/io.h>

// Global Variables
volatile char STATE;
volatile uint16_t ADC_result;
volatile unsigned int ADC_result_flag;
volatile unsigned int PAUSE;
volatile unsigned int STATION;

volatile uint16_t reflectanceTemp;
volatile uint16_t minADC;

//volatile unsigned int i =0;
volatile unsigned int objectCount;
volatile unsigned int countAlum;
volatile unsigned int countSteel;
volatile unsigned int countWhite;
volatile unsigned int countBlack;
```

Listing 1: main.h

```
#include "main.h"
#include "LinkedQueue.h"
#include "mTimer.h"
#include "init.h"
#include "motors.h"
#include "lcd.h"

int main(){

    STATE = 0;
    PAUSE = 0;
    STATION = 8;
    position = 4;
    mindelay =6;
    maxdelay =27;

    objectCount = 0;
    countAlum = 0;
    countSteel = 0;
    countWhite = 0;
    countBlack = 0;

    cli();  // Disables all interrupts

    DDRA = 0xff; //Set PORTA to output for stepper motor
    DDRB = 0xff; //Set PORTB to output for DC motor
    DDRC = 0xff; //Set PORTC to output for LCD
    DDRD = 0b11110000; //Set PORTD to input for 4 sensors and 2 push buttons (default)

    rotate_cw(101);
    initStepperMotor(); // initialize stepper motor to home position
    currentBin = 3;
    initPWM(); // initialize PWM
    initInterrupts(); // initialize interrupts
    startDCMotor(); // start DC motor
    initADC(); // initialize ADC
    initLinkedQueue(); // initialize linked queue
    configSensors(STATION); // set sensor thresholds based on station


    // Enable all interrupts
```

```
42      sei();  // Note this sets the Global Enable for all interrupts
43
44      goto POLLING_STAGE;
45
46      // POLLING STATE
47      POLLING_STAGE:
48          switch(STATE){
49              case (0) :
50                  goto POLLING_STAGE;
51                  break;  //not needed but syntax is correct
52              case (1) :
53                  goto PAUSE_STAGE;
54                  break;
55              case (2) :
56                  goto RAMPDOWN_STAGE;
57                  break;
58              case (5) :
59                  goto END;
60              default :
61                  goto POLLING_STAGE;
62          }//switch STATE
63
64      // PAUSE STATE
65      PAUSE_STAGE:
66          //Reset the state variable
67          STATE = 0;
68          goto POLLING_STAGE;
69
70      // RAMPDOWN STATE
71      RAMPDOWN_STAGE:
72          mTimer(6000);
73          PORTB = 0x00;
74          cli();
75          printLCD();
76
77          //Reset the state variable
78          STATE = 0;
79          goto POLLING_STAGE;
80
81      END:
82          // The closing STATE ... how would you get here? ANS: When ramp down button is pressed?
83          PORTA = 0xF0;   // Indicates this state is active
84          // Stop everything here...'MAKE SAFE'
85      return(0);
86  }
87
88  // INT0 is triggered by pre-reflectance optical sensor
89  ISR(INT0_vect){
90      ADCSRA |= _BV(ADSC); // initialize the ADC, start one conversion
91      minADC = 0xffff;
92      objectCount += 1;
93  }
94
95  // triggered when each ADC conversion is complete
96  ISR(ADC_vect){
97      ADC_result = ADCL | ADCH<<8;
98      if(ADC_result < minADC){
99          minADC = ADC_result;
100     }
101
102     if((PIND & 0b00000001) == 1){
103         ADCSRA |= _BV(ADSC); // initialize the ADC, start one conversion
104     } else {
105         initLink(&newLink);
106         newLink->o.reflectance = minADC;
107         enqueue(&head, &tail, &newLink);
```

```
108         if((newLink->o.reflectance >= minReflectanceAluminum) && (newLink->o.reflectance <
                maxReflectanceAluminum)){
109             newLink->o.objectType = 0;
110         } else if ((newLink->o.reflectance >= minReflectanceSteel) && (newLink->o.reflectance <
                maxReflectanceSteel)){
111             newLink->o.objectType = 1;
112         } else if ((newLink->o.reflectance >= minReflectanceWhite) && (newLink->o.reflectance <
                maxReflectanceWhite)){
113             newLink->o.objectType = 2;
114         } else if ((newLink->o.reflectance >= minReflectanceBlack) && (newLink->o.reflectance <
                maxReflectanceBlack)){
115             newLink->o.objectType = 3;
116         } else {
117             while(1){
118                 stopDCMotor();
119             }
120         }
121         STATE = 0;
122     }
123 }
124
125 // INT1 triggered by exit optical sensor
126 ISR(INT1_vect){
127     //mTimer(30);
128     dequeue(&head, &tail, &rtnLink);
129     if(rtnLink->o.objectType == 0){ //  PORTC = 1;
130         if(currentBin == 0){
131             // do nothing
132         } else if(currentBin == 1){
133             stopDCMotor();
134             rotate_cw(100);
135         } else if(currentBin == 2){
136             stopDCMotor();
137             rotate_ccw(50);
138         } else if(currentBin == 3){
139             stopDCMotor();
140             rotate_cw(50);
141         }
142         currentBin = 0;
143     } else if(rtnLink->o.objectType == 1){ //   PORTC = 2;
144         if(currentBin == 0){
145             stopDCMotor();
146             rotate_cw(100);
147         } else if(currentBin == 1){
148             // do nothing
149         } else if(currentBin == 2){
150             stopDCMotor();
151             rotate_cw(50);
152         } else if(currentBin == 3){
153             stopDCMotor();
154             rotate_ccw(50);
155         }
156         currentBin = 1;
157     } else if(rtnLink->o.objectType == 2){ //   PORTC = 3;
158         if(currentBin == 0){
159             stopDCMotor();
160             rotate_cw(50);
161         } else if(currentBin == 1){
162             stopDCMotor();
163             rotate_ccw(50);
164         } else if(currentBin == 2){
165             // do nothing
166         } else if(currentBin == 3){
167             stopDCMotor();
168             rotate_cw(100);
169         }
```

```
170         currentBin = 2;
171     } else if(rtnLink->o.objectType == 3){  //      PORTC = 4;
172         if(currentBin == 0){
173             stopDCMotor();
174             rotate_ccw(50);
175         } else if(currentBin == 1){
176             stopDCMotor();
177             rotate_cw(50);
178         } else if(currentBin == 2){
179             stopDCMotor();
180             rotate_cw(100);
181         } else if(currentBin == 3){
182             // do nothing;
183         }
184         currentBin = 3;
185     }
186 
187     startDCMotor();
188     mTimer(20);
189 
190     if(rtnLink->o.objectType == 0){ //  PORTC = 1;
191         countAlum++;
192     } else if(rtnLink->o.objectType == 1){ //   PORTC = 2;
193         countSteel++;
194     } else if(rtnLink->o.objectType == 2){ //   PORTC = 3;
195         countWhite++;
196     } else if(rtnLink->o.objectType == 3){  //      PORTC = 4;
197         countBlack++;
198     }
199 
200     objectCount -= 1;
201     free(rtnLink);
202     STATE = 0;
203 }
204 
205 // INT3 triggered by system pause button
206 ISR(INT2_vect){
207     mTimer(30); /* Wait 20ms */
208     while((PIND & 0b00000100) == 0);    /* Wait till button is released */
209     mTimer(30); /* Wait 20ms */
210     if(PAUSE == 0){
211         stopDCMotor();
212         PAUSE = 1;
213         STATE = 1;
214         printLCD();
215     } else {
216         startDCMotor();
217         PAUSE = 0;
218         STATE = 0;
219     }
220 }
221 
222 // INT4 triggered by system ramp-down button
223 ISR(INT3_vect){
224     mTimer(30); /* Wait 20ms */
225     while((PIND & 0b00001000) == 0);    /* Wait till button is released */
226     mTimer(30); /* Wait 20ms */
227 
228     //rampDown = 1;
229     STATE = 2;
230 }
231 
232 ISR(BADISR_vect){
233     while(1){
234         Lcd4_Init();
235         Lcd4_Clear();
```

```
236         Lcd4_Set_Cursor(1,0);
237         Lcd4_Write_String("********************" );
238         Lcd4_Set_Cursor(2,0);
239         Lcd4_Write_String("**     ERROR!     **" );
240         Lcd4_Set_Cursor(3,0);
241         Lcd4_Write_String("**                **" );
242         Lcd4_Set_Cursor(4,0);
243         Lcd4_Write_String("********************" );
244     }
245 }
```

Listing 2: main.c

```
1  #include <stdlib.h>
2  #include <avr/io.h>
3
4  void initPWM();
5  void initADC();
6  void initInterrupts();
7  void initLinkedQueue();
8  void configSensors(int station);
9
10 volatile uint16_t minReflectanceAluminum;
11 volatile uint16_t maxReflectanceAluminum;
12 volatile uint16_t minReflectanceSteel;
13 volatile uint16_t maxReflectanceSteel;
14 volatile uint16_t minReflectanceWhite;
15 volatile uint16_t maxReflectanceWhite;
16 volatile uint16_t minReflectanceBlack;
17 volatile uint16_t maxReflectanceBlack;
```

Listing 3: init.h

```
1  #include "init.h"
2
3  void initPWM(){
4      /* Generating PWM */
5       TCCR0A|=1<<WGM00|1<<WGM01;      /*Set timer to fast PWM mode, update OCRA at TOP=0xFF*/
6       //TIMSK0|=0<<OCIE0A;            /*Interrupt is disabled for TCNT0 on A Compare */
7       //TIFR0|=1<<OCF0A;              /*Reset TIFR0 and initiate counter */
8       TCCR0A|=1<<COM0A1;              /*Set output mode to non-inverted, OC0A clear on compare, set
            at TOP */
9       TCCR0B|=1<<CS01;                /*Set Timer pre-scale to 8    */
10      OCR0A=90;                       /*Duty Cycle out of 255 */
11      DDRB|=1<<DDB7;                  /*Set PORTB PIN7 as output to enable OC0A to control PIN
            output */
12 }
13
14 void initADC(){
15     // by default, the ADC input (analog input is set to be ADC0 / PORTF0
16     ADCSRA |= _BV(ADEN); // enable ADC
17     ADCSRA |= _BV(ADIE); // enable interrupt of ADC
18     ADCSRA |= _BV(ADPS2); // set ADC prescaler to 128
19     ADMUX |= _BV(REFS0); // set reference voltage to VCC
20     ADMUX |= _BV(MUX0); // use ADC channel 1
21     // ADLAR = 0 (default)
22 }
23
24 void initInterrupts(){
25     // config the external interrupt ======================================
26     EIMSK|= _BV(INT0); // enable INT0 for pre-reflectance optical sensor
27     EIMSK|= _BV(INT1); // enable INT1 for exit optical sensor
28     EIMSK|= _BV(INT2); // enable INT2 for system pause PB
29     EIMSK|= _BV(INT3); // enable INT3 for system ramp-down PB
30
31     EICRA |= _BV(ISC01) | _BV(ISC00); // rising edge INT0 for pre-reflectance optical sensor
```

```
    EICRA |= _BV(ISC11); // falling edge INT1 for exit optical sensor
    EICRA |= _BV(ISC21); // falling edge INT2 for system pause PB
    EICRA |= _BV(ISC31); // falling edge INT3 for system ramp-down PB
}

void configSensors(int station){
    if(station == 3){
        minReflectanceAluminum = 0;
        maxReflectanceAluminum = 400;
        minReflectanceSteel = 401;
        maxReflectanceSteel = 775;
        minReflectanceWhite = 776;
        maxReflectanceWhite = 932;
        minReflectanceBlack = 933;
        maxReflectanceBlack = 0xffff;
    } else if(station == 8){
        minReflectanceAluminum = 0;
        maxReflectanceAluminum = 350;
        minReflectanceSteel = 351;
        maxReflectanceSteel = 750;
        minReflectanceWhite = 751;
        maxReflectanceWhite = 936;
        minReflectanceBlack = 937;
        maxReflectanceBlack = 0xfff;
    } else if(station == 11){
        minReflectanceAluminum = 130;
        maxReflectanceAluminum = 210;
        minReflectanceSteel = 520;
        maxReflectanceSteel = 640;
        minReflectanceWhite = 900;
        maxReflectanceWhite = 935;
        minReflectanceBlack = 936;
        maxReflectanceBlack = 980;
    }
}
```

Listing 4: init.c

```
#include <stdlib.h>
#include <avr/io.h>

volatile unsigned int position;
volatile unsigned int currentBin;
volatile unsigned int maxdelay;
volatile unsigned int mindelay;

void startDCMotor();
void stopDCMotor();
void initStepperMotor();
int rotate_cw(int steps);
int rotate_ccw(int steps);
```

Listing 5: motors.h

```
#include "motors.h"
#include "mTimer.h"
#include "LinkedQueue.h"

//function to start DC motor
void startDCMotor(){
    PORTB = 0b00000010; // start the conveyor
}

//function to stop DC motor
void stopDCMotor(){
    PORTB = 0b00000000; // stop the conveyor
```

```
}

//function to initialize bin to home position
void initStepperMotor(){
    while((PINF & 0b00000100) == 0b00000100){
        rotate_cw(2);
    }
}

//function for clockwise rotation
int rotate_cw(int steps){
    int count=0;
    int delay;
    int acceleration=1;
    int delta;
    delta=maxdelay-mindelay;
    delay=maxdelay;

    while(steps>count){
        if(position==4){
            PORTA= 0b00011011; /*clockwise rotation through 0.9 degrees, M2, L4 energized */
            mTimer(delay);
            count++;
            if ((delay>mindelay)&(acceleration==1)) /* check if in acceleration mode */
            {
                delay=delay-1;                      /* decrease delay by 5 milliseconds or
                    accelerate */
            }
            else if((steps-count)<delta)                /* check if in deceleration mode */
            {
                delay=delay+1;                      /* increase delay by 5 milliseconds or
                    accelerate */
                acceleration=0;                     /* set to deceleration mode */
            }
            position = 1;
        } else if (position==1){
            PORTA = 0b00011101; /* half step clockwise rotation through 0.9 degrees, M1, L2
                energized */
            mTimer(delay);
            count++;
            if ((delay>mindelay)&(acceleration==1)) /* check if in acceleration mode */
            {
                delay=delay-1;                      /* decrease delay by 5 milliseconds or
                    accelerate */
            }
            else if((steps-count)<delta)                /* check if in deceleration mode */
            {
                delay=delay+1;                      /* increase delay by 5 milliseconds or
                    accelerate */
                acceleration=0;                     /* set to deceleration mode */
            }
            position = 2;
        } else if (position==2){
            PORTA=0b00101101; /*clockwise rotation through 0.9 degrees, M1, L2 energized */
            mTimer(delay);
            count++;
            if ((delay>mindelay)&(acceleration==1)) /* check if in acceleration mode */
            {
                delay=delay-1;                      /* decrease delay by 5 milliseconds or
                    accelerate */
            }
            else if((steps-count)<delta)                /* check if in deceleration mode */
            {
                delay=delay+1;                      /* increase delay by 5 milliseconds or
                    accelerate */
                acceleration=0;                     /* set to deceleration mode */
```

```
72              }
73              position = 3;
74          } else if (position==3){
75              PORTA =0b00101011; /* half step clockwise rotation through 0.9 degrees, M2, L3
                    energized */
76              mTimer(delay);
77              count++;
78              if ((delay>mindelay)&(acceleration==1)) /* check if in acceleration mode */
79              {
80                  delay=delay-1;                      /* decrease delay by 5 milliseconds or
                        accelerate */
81              }
82              else if((steps-count)<delta)                /* check if in deceleration mode */
83              {
84                  delay=delay+1;                      /* increase delay by 5 milliseconds or
                        accelerate */
85                  acceleration=0;                     /* set to deceleration mode */
86              }
87              position = 4;
88          }// end if
89      } /* end while */
90      return(0);
91  } /* end rotate_ccw */
92
93  //function for counter clockwise rotation
94  int rotate_ccw(int steps){
95      int count=0;
96      int delay;
97      int acceleration=1;
98      int delta;
99      delta=maxdelay-mindelay;
100     delay=maxdelay;
101     while(steps>count){
102         if(position==4){
103             PORTA= 0b00101011; /*clockwise rotation through 0.9 degrees, M2, L4 energized */
104             mTimer(delay);
105             count++;
106             if ((delay>mindelay)&(acceleration==1)) /* check if in acceleration mode */
107             {
108                 delay=delay-1;                      /* decrease delay by 5 milliseconds or
                        accelerate */
109             }
110             else if((steps-count)<delta)                /* check if in deceleration mode */
111             {
112                 delay=delay+1;                      /* increase delay by 5 milliseconds or
                        accelerate */
113                 acceleration=0;                     /* set to deceleration mode */
114             }
115             position = 1;
116         } else if (position==1){
117             PORTA = 0b00101101; /* half step clockwise rotation through 0.9 degrees, M1, L2
                    energized */
118             mTimer(delay);
119             count++;
120             if ((delay>mindelay)&(acceleration==1)) /* check if in acceleration mode */
121             {
122                 delay=delay-1;                      /* decrease delay by 5 milliseconds or
                        accelerate */
123             }
124             else if((steps-count)<delta)                /* check if in deceleration mode */
125             {
126                 delay=delay+1;                      /* increase delay by 5 milliseconds or
                        accelerate */
127                 acceleration=0;                     /* set to deceleration mode */
128             }
129             position = 2;
```

```
130         } else if (position==2){
131             PORTA=0b00011101; /*clockwise rotation through 0.9 degrees, M1, L2 energized */
132             mTimer(delay);
133             count++;
134             if ((delay>mindelay)&(acceleration==1)) /* check if in acceleration mode */
135             {
136                 delay=delay-1;                      /* decrease delay by 5 milliseconds or
                        accelerate */
137             }
138             else if((steps-count)<delta)                /* check if in deceleration mode */
139             {
140                 delay=delay+1;                      /* increase delay by 5 milliseconds or
                        accelerate */
141                 acceleration=0;                     /* set to deceleration mode */
142             }
143             position = 3;
144         } else if (position==3){
145             PORTA =0b00011011; /* half step clockwise rotation through 0.9 degrees, M2, L3
                    energized */
146             mTimer(delay);
147             count++;
148             if ((delay>mindelay)&(acceleration==1)) /* check if in acceleration mode */
149             {
150                 delay=delay-1;                      /* decrease delay by 5 milliseconds or
                        accelerate */
151             }
152             else if((steps-count)<delta)                /* check if in deceleration mode */
153             {
154                 delay=delay+1;                      /* increase delay by 5 milliseconds or
                        accelerate */
155                 acceleration=0;                     /* set to deceleration mode */
156             }
157             position = 4;
158         }
159     } /* end while */
160     return(0);
161 } /* end rotate_ccw */
```

Listing 6: motors.c

```
1 #include <stdlib.h>
2 #include <avr/io.h>
3
4 void mTimer(int count);
```

Listing 7: mTimer.h

```
1  #include "mTimer.h"
2
3  //############################################################################
4  //function for timer in milliseconds
5  void mTimer(int count){
6      TCCR1B|=(1<<CS10)|(1<<WGM12); //set timer control prescaling to none and mode
7      //to Clear Time on Compare, CTC.
8      OCR1A=1000; // Output compare register is set to 1000D.
9      TCNT1=0; // Initialize counter
10     //TIMSK1|= (0<<1);//Timer counter output compare A match is disabled.
11     TIFR1|=(1<<OCF1A); //Timer counter interrupt flag is cleared and timer is started.
12
13     int i=0; // for counting the loop
14     while(i<count){
15         if((TIFR1&(0b00000010))==0b00000010){
16             TIFR1|=(1<<OCF1A); //Timer counter interrupt flag is cleared.
17             i++;
18         }//if
19     }//while
```

```
}//mTimer
```

Listing 8: mTimer.c

```
#include <stdlib.h>
#include <avr/io.h>

/* Type definitions */
typedef struct {
    int objectID;   /* unique ID of object starting from 0 */
    int magnetic;   /* 1: object is magnetic (i.e. steel or aluminum), 0: object is not magnetic
        (i.e. black or white) */
    uint16_t reflectance;   /* reflectance sensor value */
    int objectType; /* type of object, 0: aluminum, 1: steel, 2: white, 3: black) */
} object;

typedef struct link{
    object      o;
    struct link *next;
} link;

void    initLink    (link **newLink);
void    setup       (link **h, link **t);
void    clearQueue  (link **h, link **t);
void    enqueue     (link **h, link **t, link **nL);
void    dequeue     (link **h, link **t, link **deQueuedLink);
object firstValue   (link **h);
char    isEmpty     (link **h);
int     size        (link **h, link **t);

struct link *head;          /* The ptr to the head of the queue */
struct link *tail;          /* The ptr to the tail of the queue */
struct link *newLink;       /* A ptr to a link aggregate data type (struct) */
struct link *rtnLink;       /* same as the above */
```

Listing 9: LinkedQueue.h

```
#include "LinkedQueue.h"

void initLinkedQueue(){
    rtnLink = NULL;
    newLink = NULL;
    setup(&head, &tail);
}

/*****************************************************************************
* DESC: initializes the linked queue to 'NULL' status
* INPUT: the head and tail pointers by reference
*/
void setup(link **h,link **t){
    *h = NULL;      /* Point the head to NOTHING (NULL) */
    *t = NULL;      /* Point the tail to NOTHING (NULL) */
    return;
}/*setup*/

/*****************************************************************************
* DESC: This initializes a link and returns the pointer to the new link or NULL if error
* INPUT: the head and tail pointers by reference
*/
void initLink(link **newLink){
    //link *l;
    *newLink = malloc(sizeof(link));
    (*newLink)->next = NULL;
    return;
}/*initLink*/

```

```
/*****************************************************************************
*  DESC: Accepts as input a new link by reference, and assigns the head and tail
*  of the queue accordingly
*  INPUT: the head and tail pointers, and a pointer to the new link that was created
*/
/* will put an item at the tail of the queue */
void enqueue(link **h, link **t, link **nL){

    if (*t != NULL){
        /* Not an empty queue */
        (*t)->next = *nL;
        *t = *nL; //(*t)->next;
    }/*if*/
    else{
        /* It's an empty Queue */
        //(*h)->next = *nL;
        //should be this
        *h = *nL;
        *t = *nL;
    }/* else */
    return;
}/*enqueue*/

/*****************************************************************************
* DESC : Removes the link from the head of the list and assigns it to deQueuedLink
* INPUT: The head and tail pointers, and a ptr 'deQueuedLink'
*        which the removed link will be assigned to
*/
/* This will remove the link and element within the link from the head of the queue */
void dequeue(link **h, link **t, link **deQueuedLink){
    /* ENTER YOUR CODE HERE */
    *deQueuedLink = *h; // Will set to NULL if Head points to NULL
    /* Ensure it is not an empty queue */
    if (*h != NULL){
        *h = (*h)->next;
    }/*if*/

    if (*h == NULL){
        *t = NULL;
    }
    return;
}/*dequeue*/

/*****************************************************************************
* DESC: Peeks at the first element in the list
* INPUT: The head pointer
* RETURNS: The element contained within the queue
*/
/* This simply allows you to peek at the head element of the queue and returns a NULL pointer if
    empty */
object firstValue(link **h){
    return((*h)->o);
}/*firstValue*/

/*****************************************************************************
* DESC: deallocates (frees) all the memory consumed by the Queue
* INPUT: the pointers to the head and the tail
*/
/* This clears the queue */
void clearQueue(link **h, link **t){
    link *temp;
    while (*h != NULL){
        temp = *h;
        *h=(*h)->next;
        free(temp);
    }/*while*/
```

```
95      /* Last but not least set the tail to NULL */
96      *t = NULL;
97      return;
98  }/*clearQueue*/
99
100 /*****************************************************************************
101 * DESC: Checks to see whether the queue is empty or not
102 * INPUT: The head pointer
103 * RETURNS: 1:if the queue is empty, and 0:if the queue is NOT empty
104 */
105 /* Check to see if the queue is empty */
106 char isEmpty(link **h){
107     /* ENTER YOUR CODE HERE */
108     return(*h == NULL);
109 }/*isEmpty*/
110
111 /*****************************************************************************
112 * DESC: Obtains the number of links in the queue
113 * INPUT: The head and tail pointer
114 * RETURNS: An integer with the number of links in the queue
115 */
116 /* returns the size of the queue*/
117 int size(link **h, link **t){
118     link    *temp;          /* will store the link while traversing the queue */
119     int     numObjects;
120     numObjects = 0;
121     temp = *h;          /* point to the first item in the list */
122     while(temp != NULL){
123         numObjects++;
124         temp = temp->next;
125     }/*while*/
126     return(numObjects);
127 }/*size*/
```

Listing 10: LinkedQueue.c

```
1  //LCD Functions Developed by electroSome
2  #define eS_PORTA0 0
3  #define eS_PORTA1 1
4  #define eS_PORTA2 2
5  #define eS_PORTA3 3
6  #define eS_PORTA4 4
7  #define eS_PORTA5 5
8  #define eS_PORTA6 6
9  #define eS_PORTA7 7
10 #define eS_PORTB0 10
11 #define eS_PORTB1 11
12 #define eS_PORTB2 12
13 #define eS_PORTB3 13
14 #define eS_PORTB4 14
15 #define eS_PORTB5 15
16 #define eS_PORTB6 16
17 #define eS_PORTB7 17
18 #define eS_PORTC0 20
19 #define eS_PORTC1 21
20 #define eS_PORTC2 22
21 #define eS_PORTC3 23
22 #define eS_PORTC4 24
23 #define eS_PORTC5 25
24 #define eS_PORTC6 26
25 #define eS_PORTC7 27
26 #define eS_PORTD0 30
27 #define eS_PORTD1 31
28 #define eS_PORTD2 32
29 #define eS_PORTD3 33
30 #define eS_PORTD4 34
31 #define eS_PORTD5 35
```

```
#define eS_PORTD6 36
#define eS_PORTD7 37

#ifndef D0
#define D0 eS_PORTA0
#define D1 eS_PORTA1
#define D2 eS_PORTA2
#define D3 eS_PORTA3
#endif

#include<util/delay.h>
#include "main.h"

void print_LCD();
```

Listing 11: lcd.h

```
#define D4 eS_PORTC4
#define D5 eS_PORTC5
#define D6 eS_PORTC6
#define D7 eS_PORTC7
#define RS eS_PORTC0
#define EN eS_PORTC1

#include "lcd.h"

void printLCD(){
    Lcd4_Init();
    Lcd4_Clear();

    Lcd4_Set_Cursor(1,0);
    Lcd4_Write_String("BLK =    WHT=  " );

    switch (countBlack){
        case 0:
        Lcd4_Set_Cursor(1,6);
        Lcd4_Write_Char(0x30);
        break;

        case 1:
        Lcd4_Set_Cursor(1,6);
        Lcd4_Write_Char(0x31);
        break;

        case 2:
        Lcd4_Set_Cursor(1,6);
        Lcd4_Write_Char(0x32);
        break;

        case 3:
        Lcd4_Set_Cursor(1,6);
        Lcd4_Write_Char(0x33);
        break;

        case 4:
        Lcd4_Set_Cursor(1,6);
        Lcd4_Write_Char(0x34);
        break;

        case 5:
        Lcd4_Set_Cursor(1,6);
        Lcd4_Write_Char(0x35);
        break;

        case 6:
        Lcd4_Set_Cursor(1,6);
        Lcd4_Write_Char(0x36);
```

```
51          break;
52
53          case 7:
54          Lcd4_Set_Cursor(1,6);
55          Lcd4_Write_Char(0x37);
56          break;
57
58          case 8:
59          Lcd4_Set_Cursor(1,6);
60          Lcd4_Write_Char(0x38);
61          break;
62
63          case 9:
64          Lcd4_Set_Cursor(1,6);
65          Lcd4_Write_Char(0x39);
66          break;
67
68          case 10:
69          Lcd4_Set_Cursor(1,6);
70          Lcd4_Write_Char(0x31);
71          Lcd4_Set_Cursor(1,7);
72          Lcd4_Write_Char(0x30);
73          break;
74
75          case 11:
76          Lcd4_Set_Cursor(1,6);
77          Lcd4_Write_Char(0x31);
78          Lcd4_Set_Cursor(1,7);
79          Lcd4_Write_Char(0x31);
80          break;
81
82          case 12:
83          Lcd4_Set_Cursor(1,6);
84          Lcd4_Write_Char(0x31);
85          Lcd4_Set_Cursor(1,7);
86          Lcd4_Write_Char(0x32);
87          break;
88
89          default:
90          Lcd4_Set_Cursor(1,6);
91          Lcd4_Write_Char(0x30);
92      }
93
94      switch (countWhite){
95          case 0:
96          Lcd4_Set_Cursor(1,14);
97          Lcd4_Write_Char(0x30);
98          break;
99
100         case 1:
101         Lcd4_Set_Cursor(1,14);
102         Lcd4_Write_Char(0x31);
103         break;
104
105         case 2:
106         Lcd4_Set_Cursor(1,14);
107         Lcd4_Write_Char(0x32);
108         break;
109
110         case 3:
111         Lcd4_Set_Cursor(1,14);
112         Lcd4_Write_Char(0x33);
113         break;
114
115         case 4:
116         Lcd4_Set_Cursor(1,14);
```

```
117         Lcd4_Write_Char(0x34);
118         break;
119 
120         case 5:
121         Lcd4_Set_Cursor(1,14);
122         Lcd4_Write_Char(0x35);
123         break;
124 
125         case 6:
126         Lcd4_Set_Cursor(1,14);
127         Lcd4_Write_Char(0x36);
128         break;
129 
130         case 7:
131         Lcd4_Set_Cursor(1,14);
132         Lcd4_Write_Char(0x37);
133         break;
134 
135         case 8:
136         Lcd4_Set_Cursor(1,14);
137         Lcd4_Write_Char(0x38);
138         break;
139 
140         case 9:
141         Lcd4_Set_Cursor(1,14);
142         Lcd4_Write_Char(0x39);
143         break;
144 
145         case 10:
146         Lcd4_Set_Cursor(1,14);
147         Lcd4_Write_Char(0x31);
148         Lcd4_Set_Cursor(1,15);
149         Lcd4_Write_Char(0x30);
150         break;
151 
152         case 11:
153         Lcd4_Set_Cursor(1,14);
154         Lcd4_Write_Char(0x31);
155         Lcd4_Set_Cursor(1,15);
156         Lcd4_Write_Char(0x31);
157         break;
158 
159         case 12:
160         Lcd4_Set_Cursor(1,14);
161         Lcd4_Write_Char(0x31);
162         Lcd4_Set_Cursor(1,15);
163         Lcd4_Write_Char(0x32);
164         break;
165 
166         default:
167         Lcd4_Set_Cursor(1,14);
168         Lcd4_Write_Char(0x30);
169     }
170 
171     //ALUMINUM, STEEL AND UNSORTED
172     Lcd4_Set_Cursor(2,0);
173     Lcd4_Write_String("ALUM=    STL=       UNSORTED = " );
174 
175     switch (countAlum){
176         case 0:
177         Lcd4_Set_Cursor(2,6);
178         Lcd4_Write_Char(0x30);
179         break;
180 
181         case 1:
182         Lcd4_Set_Cursor(2,6);
```

```
183         Lcd4_Write_Char(0x31);
184         break;
185
186         case 2:
187         Lcd4_Set_Cursor(2,6);
188         Lcd4_Write_Char(0x32);
189         break;
190
191         case 3:
192         Lcd4_Set_Cursor(2,6);
193         Lcd4_Write_Char(0x33);
194         break;
195
196         case 4:
197         Lcd4_Set_Cursor(2,6);
198         Lcd4_Write_Char(0x34);
199         break;
200
201         case 5:
202         Lcd4_Set_Cursor(2,6);
203         Lcd4_Write_Char(0x35);
204         break;
205
206         case 6:
207         Lcd4_Set_Cursor(2,6);
208         Lcd4_Write_Char(0x36);
209         break;
210
211         case 7:
212         Lcd4_Set_Cursor(2,6);
213         Lcd4_Write_Char(0x37);
214         break;
215
216         case 8:
217         Lcd4_Set_Cursor(2,6);
218         Lcd4_Write_Char(0x38);
219         break;
220
221         case 9:
222         Lcd4_Set_Cursor(2,6);
223         Lcd4_Write_Char(0x39);
224         break;
225
226         case 10:
227         Lcd4_Set_Cursor(2,6);
228         Lcd4_Write_Char(0x31);
229         Lcd4_Set_Cursor(2,7);
230         Lcd4_Write_Char(0x30);
231         break;
232
233         case 11:
234         Lcd4_Set_Cursor(2,6);
235         Lcd4_Write_Char(0x31);
236         Lcd4_Set_Cursor(2,7);
237         Lcd4_Write_Char(0x31);
238         break;
239
240         case 12:
241         Lcd4_Set_Cursor(2,6);
242         Lcd4_Write_Char(0x31);
243         Lcd4_Set_Cursor(2,7);
244         Lcd4_Write_Char(0x32);
245         break;
246
247         default:
248         Lcd4_Set_Cursor(2,6);
```

```
249         Lcd4_Write_Char(0x30);
250     }
251 
252     switch (countSteel){
253         case 0:
254         Lcd4_Set_Cursor(2,14);
255         Lcd4_Write_Char(0x30);
256         break;
257 
258         case 1:
259         Lcd4_Set_Cursor(2,14);
260         Lcd4_Write_Char(0x31);
261         break;
262 
263         case 2:
264         Lcd4_Set_Cursor(2,14);
265         Lcd4_Write_Char(0x32);
266         break;
267 
268         case 3:
269         Lcd4_Set_Cursor(2,14);
270         Lcd4_Write_Char(0x33);
271         break;
272 
273         case 4:
274         Lcd4_Set_Cursor(2,14);
275         Lcd4_Write_Char(0x34);
276         break;
277 
278         case 5:
279         Lcd4_Set_Cursor(2,14);
280         Lcd4_Write_Char(0x35);
281         break;
282 
283         case 6:
284         Lcd4_Set_Cursor(2,14);
285         Lcd4_Write_Char(0x36);
286         break;
287 
288         case 7:
289         Lcd4_Set_Cursor(2,14);
290         Lcd4_Write_Char(0x37);
291         break;
292 
293         case 8:
294         Lcd4_Set_Cursor(2,14);
295         Lcd4_Write_Char(0x38);
296         break;
297 
298         case 9:
299         Lcd4_Set_Cursor(2,14);
300         Lcd4_Write_Char(0x39);
301         break;
302 
303         case 10:
304         Lcd4_Set_Cursor(2,14);
305         Lcd4_Write_Char(0x31);
306         Lcd4_Set_Cursor(2,15);
307         Lcd4_Write_Char(0x30);
308         break;
309 
310         case 11:
311         Lcd4_Set_Cursor(2,14);
312         Lcd4_Write_Char(0x31);
313         Lcd4_Set_Cursor(2,15);
314         Lcd4_Write_Char(0x31);
```

```
315         break;
316
317         case 12:
318         Lcd4_Set_Cursor(2,14);
319         Lcd4_Write_Char(0x31);
320         Lcd4_Set_Cursor(2,15);
321         Lcd4_Write_Char(0x32);
322         break;
323
324         default:
325         Lcd4_Set_Cursor(2,14);
326         Lcd4_Write_Char(0x30);
327     }
328
329     switch (objectCount){
330         case 0:
331         Lcd4_Set_Cursor(2,31);
332         Lcd4_Write_Char(0x30);
333         break;
334
335         case 1:
336         Lcd4_Set_Cursor(2,31);
337         Lcd4_Write_Char(0x31);
338         break;
339
340         case 2:
341         Lcd4_Set_Cursor(2,31);
342         Lcd4_Write_Char(0x32);
343         break;
344
345         case 3:
346         Lcd4_Set_Cursor(2,31);
347         Lcd4_Write_Char(0x33);
348         break;
349
350         case 4:
351         Lcd4_Set_Cursor(2,31);
352         Lcd4_Write_Char(0x34);
353         break;
354
355         case 5:
356         Lcd4_Set_Cursor(2,31);
357         Lcd4_Write_Char(0x35);
358         break;
359
360         case 6:
361         Lcd4_Set_Cursor(2,31);
362         Lcd4_Write_Char(0x36);
363         break;
364
365         case 7:
366         Lcd4_Set_Cursor(2,31);
367         Lcd4_Write_Char(0x37);
368         break;
369
370         case 8:
371         Lcd4_Set_Cursor(2,31);
372         Lcd4_Write_Char(0x38);
373         break;
374
375         case 9:
376         Lcd4_Set_Cursor(2,31);
377         Lcd4_Write_Char(0x39);
378         break;
379
380         case 10:
```

```
381         Lcd4_Set_Cursor(2,31);
382         Lcd4_Write_Char(0x31);
383         Lcd4_Set_Cursor(2,32);
384         Lcd4_Write_Char(0x30);
385         break;
386 
387         case 11:
388         Lcd4_Set_Cursor(2,31);
389         Lcd4_Write_Char(0x31);
390         Lcd4_Set_Cursor(2,32);
391         Lcd4_Write_Char(0x31);
392         break;
393 
394         case 12:
395         Lcd4_Set_Cursor(2,31);
396         Lcd4_Write_Char(0x31);
397         Lcd4_Set_Cursor(2,32);
398         Lcd4_Write_Char(0x32);
399         break;
400 
401         default:
402         Lcd4_Set_Cursor(2,31);
403         Lcd4_Write_Char(0x30);
404     }
405 }
406 
407 void pinChange(int a, int b)
408 {
409     if(b == 0)
410     {
411         if(a == eS_PORTA0)
412         PORTA &= ~(1<<PA0);
413         else if(a == eS_PORTA1)
414         PORTA &= ~(1<<PA1);
415         else if(a == eS_PORTA2)
416         PORTA &= ~(1<<PA2);
417         else if(a == eS_PORTA3)
418         PORTA &= ~(1<<PA3);
419         else if(a == eS_PORTA4)
420         PORTA &= ~(1<<PA4);
421         else if(a == eS_PORTA5)
422         PORTA &= ~(1<<PA5);
423         else if(a == eS_PORTA6)
424         PORTA &= ~(1<<PA6);
425         else if(a == eS_PORTA7)
426         PORTA &= ~(1<<PA7);
427         else if(a == eS_PORTB0)
428         PORTB &= ~(1<<PB0);
429         else if(a == eS_PORTB1)
430         PORTB &= ~(1<<PB1);
431         else if(a == eS_PORTB2)
432         PORTB &= ~(1<<PB2);
433         else if(a == eS_PORTB3)
434         PORTB &= ~(1<<PB3);
435         else if(a == eS_PORTB4)
436         PORTB &= ~(1<<PB4);
437         else if(a == eS_PORTB5)
438         PORTB &= ~(1<<PB5);
439         else if(a == eS_PORTB6)
440         PORTB &= ~(1<<PB6);
441         else if(a == eS_PORTB7)
442         PORTB &= ~(1<<PB7);
443         else if(a == eS_PORTC0)
444         PORTC &= ~(1<<PC0);
445         else if(a == eS_PORTC1)
446         PORTC &= ~(1<<PC1);
```

```
447         else if(a == eS_PORTC2)
448         PORTC &= ~(1<<PC2);
449         else if(a == eS_PORTC3)
450         PORTC &= ~(1<<PC3);
451         else if(a == eS_PORTC4)
452         PORTC &= ~(1<<PC4);
453         else if(a == eS_PORTC5)
454         PORTC &= ~(1<<PC5);
455         else if(a == eS_PORTC6)
456         PORTC &= ~(1<<PC6);
457         else if(a == eS_PORTC7)
458         PORTC &= ~(1<<PC7);
459         else if(a == eS_PORTD0)
460         PORTD &= ~(1<<PD0);
461         else if(a == eS_PORTD1)
462         PORTD &= ~(1<<PD1);
463         else if(a == eS_PORTD2)
464         PORTD &= ~(1<<PD2);
465         else if(a == eS_PORTD3)
466         PORTD &= ~(1<<PD3);
467         else if(a == eS_PORTD4)
468         PORTD &= ~(1<<PD4);
469         else if(a == eS_PORTD5)
470         PORTD &= ~(1<<PD5);
471         else if(a == eS_PORTD6)
472         PORTD &= ~(1<<PD6);
473         else if(a == eS_PORTD7)
474         PORTD &= ~(1<<PD7);
475     } else {
476         if(a == eS_PORTA0)
477         PORTA |= (1<<PA0);
478         else if(a == eS_PORTA1)
479         PORTA |= (1<<PA1);
480         else if(a == eS_PORTA2)
481         PORTA |= (1<<PA2);
482         else if(a == eS_PORTA3)
483         PORTA |= (1<<PA3);
484         else if(a == eS_PORTA4)
485         PORTA |= (1<<PA4);
486         else if(a == eS_PORTA5)
487         PORTA |= (1<<PA5);
488         else if(a == eS_PORTA6)
489         PORTA |= (1<<PA6);
490         else if(a == eS_PORTA7)
491         PORTA |= (1<<PA7);
492         else if(a == eS_PORTB0)
493         PORTB |= (1<<PB0);
494         else if(a == eS_PORTB1)
495         PORTB |= (1<<PB1);
496         else if(a == eS_PORTB2)
497         PORTB |= (1<<PB2);
498         else if(a == eS_PORTB3)
499         PORTB |= (1<<PB3);
500         else if(a == eS_PORTB4)
501         PORTB |= (1<<PB4);
502         else if(a == eS_PORTB5)
503         PORTB |= (1<<PB5);
504         else if(a == eS_PORTB6)
505         PORTB |= (1<<PB6);
506         else if(a == eS_PORTB7)
507         PORTB |= (1<<PB7);
508         else if(a == eS_PORTC0)
509         PORTC |= (1<<PC0);
510         else if(a == eS_PORTC1)
511         PORTC |= (1<<PC1);
512         else if(a == eS_PORTC2)
```

```
513         PORTC |= (1<<PC2);
514         else if(a == eS_PORTC3)
515         PORTC |= (1<<PC3);
516         else if(a == eS_PORTC4)
517         PORTC |= (1<<PC4);
518         else if(a == eS_PORTC5)
519         PORTC |= (1<<PC5);
520         else if(a == eS_PORTC6)
521         PORTC |= (1<<PC6);
522         else if(a == eS_PORTC7)
523         PORTC |= (1<<PC7);
524         else if(a == eS_PORTD0)
525         PORTD |= (1<<PD0);
526         else if(a == eS_PORTD1)
527         PORTD |= (1<<PD1);
528         else if(a == eS_PORTD2)
529         PORTD |= (1<<PD2);
530         else if(a == eS_PORTD3)
531         PORTD |= (1<<PD3);
532         else if(a == eS_PORTD4)
533         PORTD |= (1<<PD4);
534         else if(a == eS_PORTD5)
535         PORTD |= (1<<PD5);
536         else if(a == eS_PORTD6)
537         PORTD |= (1<<PD6);
538         else if(a == eS_PORTD7)
539         PORTD |= (1<<PD7);
540     }
541 }
542
543 //LCD 8 Bit Interfacing Functions
544 void Lcd8_Port(char a)
545 {
546     if(a & 1)
547     pinChange(D0,1);
548     else
549     pinChange(D0,0);
550
551     if(a & 2)
552     pinChange(D1,1);
553     else
554     pinChange(D1,0);
555
556     if(a & 4)
557     pinChange(D2,1);
558     else
559     pinChange(D2,0);
560
561     if(a & 8)
562     pinChange(D3,1);
563     else
564     pinChange(D3,0);
565
566     if(a & 16)
567     pinChange(D4,1);
568     else
569     pinChange(D4,0);
570
571     if(a & 32)
572     pinChange(D5,1);
573     else
574     pinChange(D5,0);
575
576     if(a & 64)
577     pinChange(D6,1);
578     else
```

```
579     pinChange(D6,0);
580
581     if(a & 128)
582     pinChange(D7,1);
583     else
584     pinChange(D7,0);
585 }
586
587 void Lcd8_Cmd(char a)
588 {
589     pinChange(RS,0);              // => RS = 0
590     Lcd8_Port(a);              //Data transfer
591     pinChange(EN,1);              // => E = 1
592     _delay_ms(1);
593     pinChange(EN,0);              // => E = 0
594     _delay_ms(1);
595 }
596
597 void Lcd8_Clear()
598 {
599     Lcd8_Cmd(1);
600 }
601
602 void Lcd8_Set_Cursor(char a, char b)
603 {
604     if(a == 1)
605     Lcd8_Cmd(0x80 + b);
606     else if(a == 2)
607     Lcd8_Cmd(0xC0 + b);
608 }
609
610 void Lcd8_Init()
611 {
612     pinChange(RS,0);
613     pinChange(EN,0);
614     _delay_ms(20);
615     ////////////// Reset process from datasheet /////////
616     Lcd8_Cmd(0x30);
617     _delay_ms(5);
618     Lcd8_Cmd(0x30);
619     _delay_ms(1);
620     Lcd8_Cmd(0x30);
621     _delay_ms(10);
622     ////////////////////////////////////////////////////
623     Lcd8_Cmd(0x38);    //function set
624     Lcd8_Cmd(0x0C);    //display on,cursor off,blink off
625     Lcd8_Cmd(0x01);    //clear display
626     Lcd8_Cmd(0x06);    //entry mode, set increment
627 }
628
629 void Lcd8_Write_Char(char a)
630 {
631     pinChange(RS,1);              // => RS = 1
632     Lcd8_Port(a);              //Data transfer
633     pinChange(EN,1);              // => E = 1
634     _delay_ms(1);
635     pinChange(EN,0);              // => E = 04
636     _delay_ms(1);
637 }
638
639 void Lcd8_Write_String(char *a)
640 {
641     int i;
642     for(i=0;a[i]!='\0';i++)
643     Lcd8_Write_Char(a[i]);
644 }
```

```
645
646 void Lcd8_Shift_Right()
647 {
648     Lcd8_Cmd(0x1C);
649 }
650
651 void Lcd8_Shift_Left()
652 {
653     Lcd8_Cmd(0x18);
654 }
655 //End LCD 8 Bit Interfacing Functions
656
657 //LCD 4 Bit Interfacing Functions
658
659 void Lcd4_Port(char a)
660 {
661     if(a & 1)
662     pinChange(D4,1);
663     else
664     pinChange(D4,0);
665
666     if(a & 2)
667     pinChange(D5,1);
668     else
669     pinChange(D5,0);
670
671     if(a & 4)
672     pinChange(D6,1);
673     else
674     pinChange(D6,0);
675
676     if(a & 8)
677     pinChange(D7,1);
678     else
679     pinChange(D7,0);
680 }
681 void Lcd4_Cmd(char a)
682 {
683     pinChange(RS,0);              // => RS = 0
684     Lcd4_Port(a);
685     pinChange(EN,1);             // => E = 1
686     _delay_ms(1);
687     pinChange(EN,0);              // => E = 0
688     _delay_ms(1);
689 }
690
691 void Lcd4_Clear()
692 {
693     Lcd4_Cmd(0);
694     Lcd4_Cmd(1);
695 }
696
697 void Lcd4_Set_Cursor(char a, char b)
698 {
699     char temp,z,y;
700     if(a == 1)
701     {
702         temp = 0x80 + b;
703         z = temp>>4;
704         y = (0x80+b) & 0x0F;
705         Lcd4_Cmd(z);
706         Lcd4_Cmd(y);
707     }
708     else if(a == 2)
709     {
710         temp = 0xC0 + b;
```

```
711         z = temp>>4;
712         y = (0xC0+b) & 0x0F;
713         Lcd4_Cmd(z);
714         Lcd4_Cmd(y);
715     }
716 }
717
718 void Lcd4_Init()
719 {
720     Lcd4_Port(0x00);
721     _delay_ms(20);
722     ////////////// Reset process from datasheet /////////
723     Lcd4_Cmd(0x03);
724     _delay_ms(5);
725     Lcd4_Cmd(0x03);
726     _delay_ms(11);
727     Lcd4_Cmd(0x03);
728     /////////////////////////////////////////////////////
729     Lcd4_Cmd(0x02);
730     Lcd4_Cmd(0x02);
731     Lcd4_Cmd(0x08);
732     Lcd4_Cmd(0x00);
733     Lcd4_Cmd(0x0C);
734     Lcd4_Cmd(0x00);
735     Lcd4_Cmd(0x06);
736 }
737
738 void Lcd4_Write_Char(char a)
739 {
740     char temp,y;
741     temp = a&0x0F;
742     y = a&0xF0;
743     pinChange(RS,1);             // => RS = 1
744     Lcd4_Port(y>>4);             //Data transfer
745     pinChange(EN,1);
746     _delay_ms(1);
747     pinChange(EN,0);
748     _delay_ms(1);
749     Lcd4_Port(temp);
750     pinChange(EN,1);
751     _delay_ms(1);
752     pinChange(EN,0);
753     _delay_ms(1);
754 }
755
756 void Lcd4_Write_String(char *a)
757 {
758     int i;
759     for(i=0;a[i]!='\0';i++)
760     Lcd4_Write_Char(a[i]);
761 }
762
763 void Lcd4_Shift_Right()
764 {
765     Lcd4_Cmd(0x01);
766     Lcd4_Cmd(0x0C);
767 }
768
769 void Lcd4_Shift_Left()
770 {
771     Lcd4_Cmd(0x01);
772     Lcd4_Cmd(0x08);
773 }
774 //End LCD 4 Bit Interfacing Functions
```

Listing 12: lcd.c