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 19, 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 was to design a high-performance inspection system, implement the proposed design using an ATMEL AVR microcontroller and to demonstrate it's functionality. The inspection system was 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 Analog 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 was to design a high-performance inspection system, implement the proposed design using an ATMEL AVR microcontroller and demonstrate it's functionality. The inspection system was required to sort four types of cylindrically shaped objects which were either black plastic, white plastic, steel, or aluminum material. For the final performance testing, the inspection system needed to sort 48 of such items into their respective bins in less than 60 seconds. Also, there had to be a pause button which paused the inspection system and a ramp-down button which stopped the system after sorting all the items on the conveyor. The details such as the number of each item sorted and unsorted items on the belt were to be displayed using LEDs or optionally using LCD display.

1.2 Design Purpose and Overview

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

1.2.1 AT90USB 1287 Microcontroller

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. An image of the MCU is shown in Figure 1.

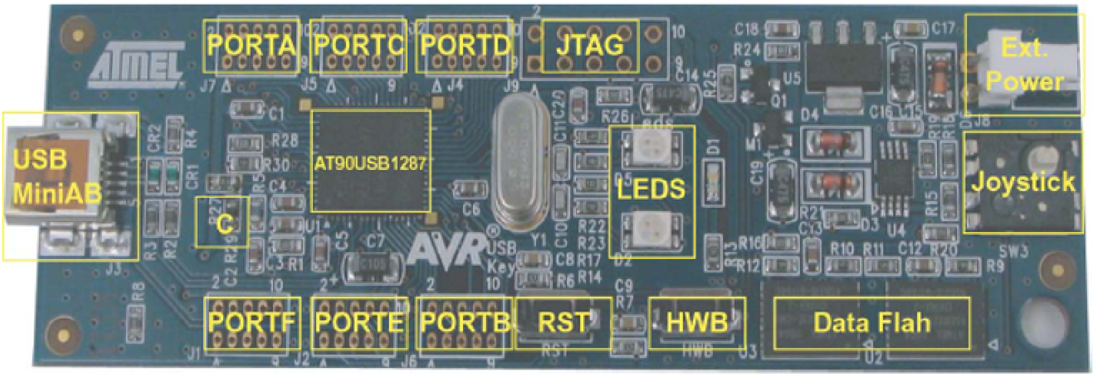


Figure 1: ATMEL AT90USB 1287 MCU

1.2.2 Conveyor System

A conveyor belt was used to transport the items that were to be sorted. It was driven by a DC motor and the speed of the conveyor was controlled by the PWM signal. Guides were provided at the loading point of the conveyor which aligned the loaded items on the belt and ensured that there was a sufficient gap between adjacent items for the sensors to read correctly. The belt could be started or stopped as required but the speed of the belt was to be constant during the operation. A model of the system is shown in Figure 2.

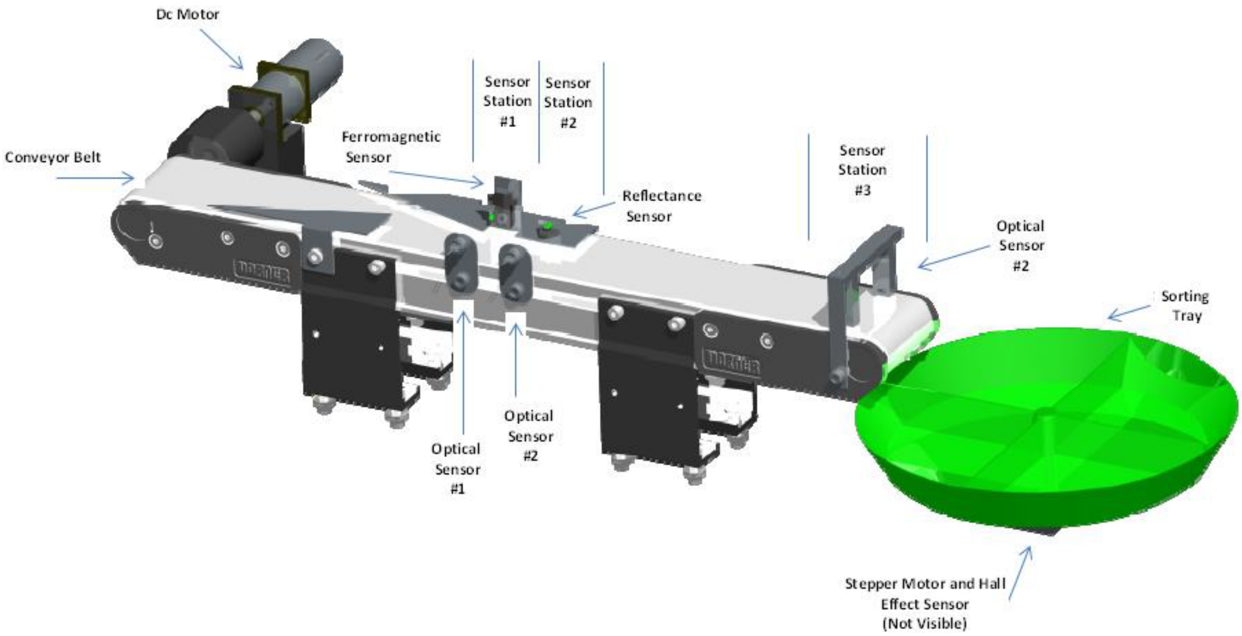


Figure 2: Model of the conveyor system and inspection stations

1.2.3 Inspection Stations

As the items traveled on the conveyor belt, they passed through various sensor stations which classified the material and visual characteristics of the items. Three sensors were 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 was used to identify if an item was in the range of the reflectance sensor. Only when an object was between the pre-reflectance optical sensor and the reflectance sensor, the analog signal was measured and converted into a digital value.

- Reflectance Sensor:

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

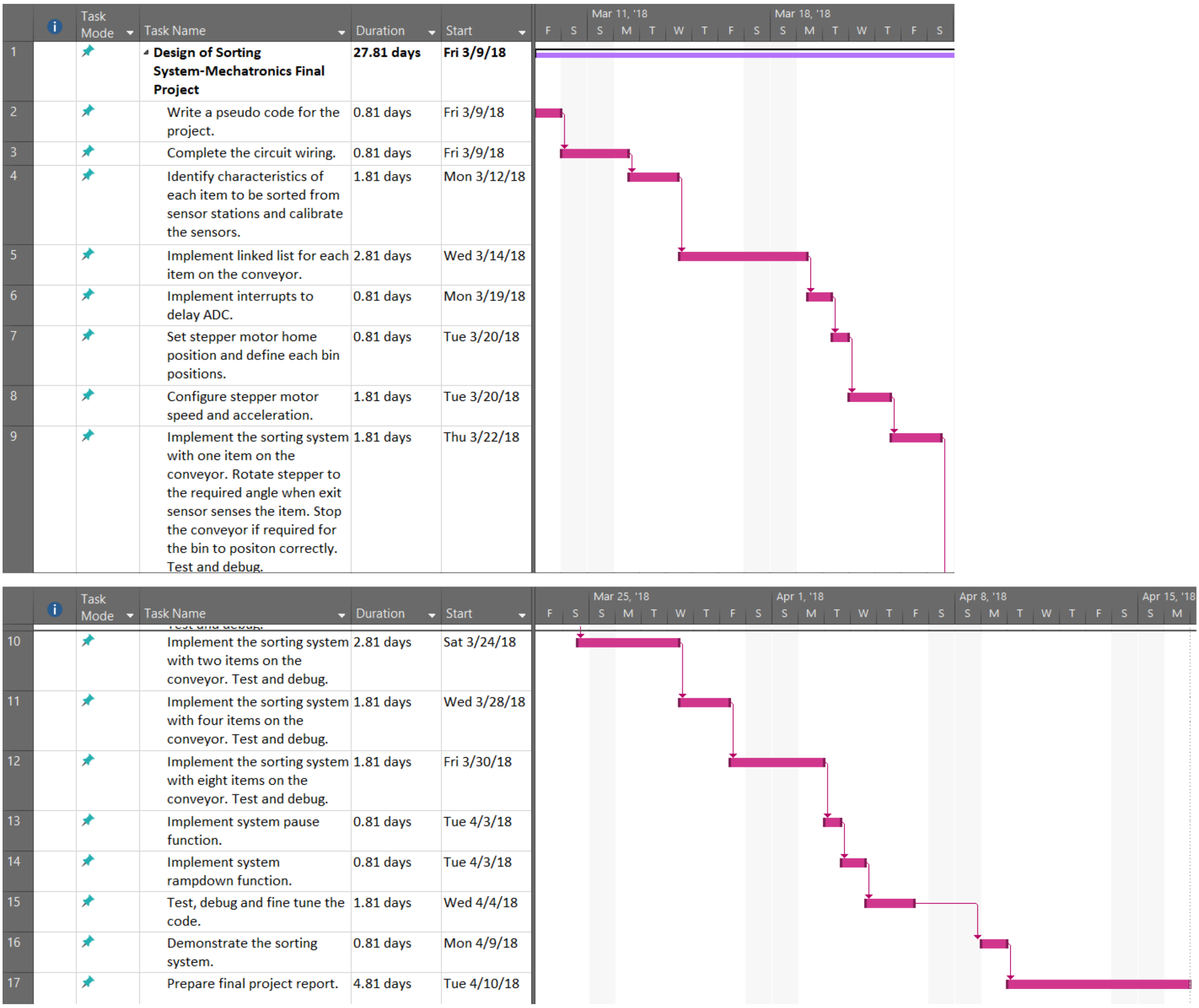
- Exit Optical Sensor

  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 3: 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 were loaded onto the conveyor belt which took them through the sensor stations. The first sensor station detected the type of the item and the exit sensor station made sure that the tray was positioned correctly before the item was allowed to fall down into the bin.

The conveyor belt was driven by a DC motor using PWM signal. The speed of the DC motor was controlled by the width of the PWM signal. The speed was optimized for minimum traveling time with the time available for the ADC as a constraint. Having a faster belt speed resulted in overlapping ADC reflectance values for the different objects and hence resulted in identifying incorrect object type. Also, at very high speed, when the belt was stopped at the exit sensor, the object sometimes slipped and fell into the incorrect bin due to the higher inertia caused by the faster belt speed.

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

When the object reached the exit optical sensor station, the first item in the linked list was dequeued and the object type was 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 was compared with the current tray position. If the tray position matched that of the object type, the item was allowed to fall into the correct bin without stopping the conveyor belt. If the current bin did not match the object type, the belt was stopped and a command was given to the stepper motor to rotate the tray to the correct bin. Once the correct bin was positioned, the belt was started and the object allowed to fall into the correct bin.

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

The design is mainly interrupt driven. A large chunk of the code is executed inside the interrupts. The 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. The fourth and fifth interrupts 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 when activated. When the pushbutton is pressed and released, the interrupt stops the conveyor belt immediately and displays the information such as the 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 was 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

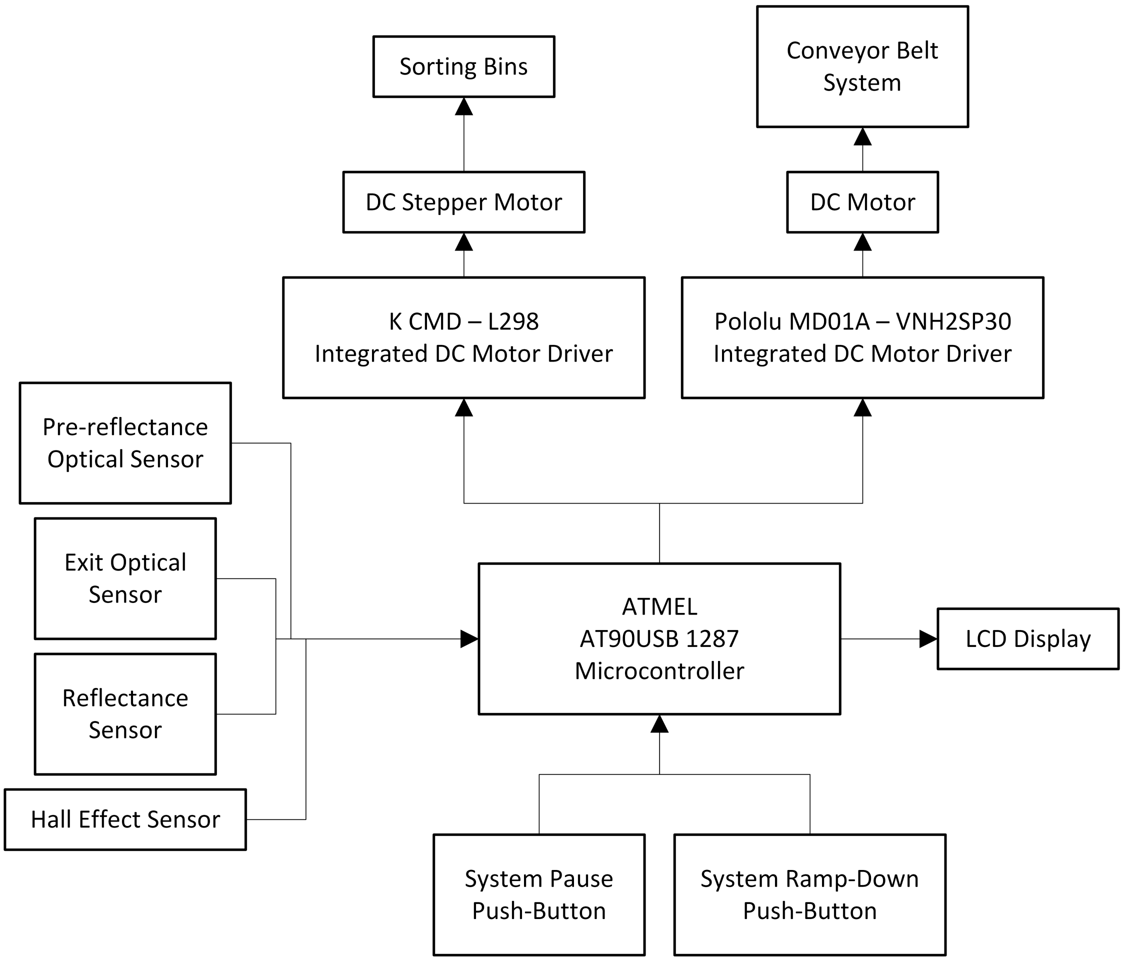


Figure 4: Block diagram of complete system

4.2 Circuit Diagram

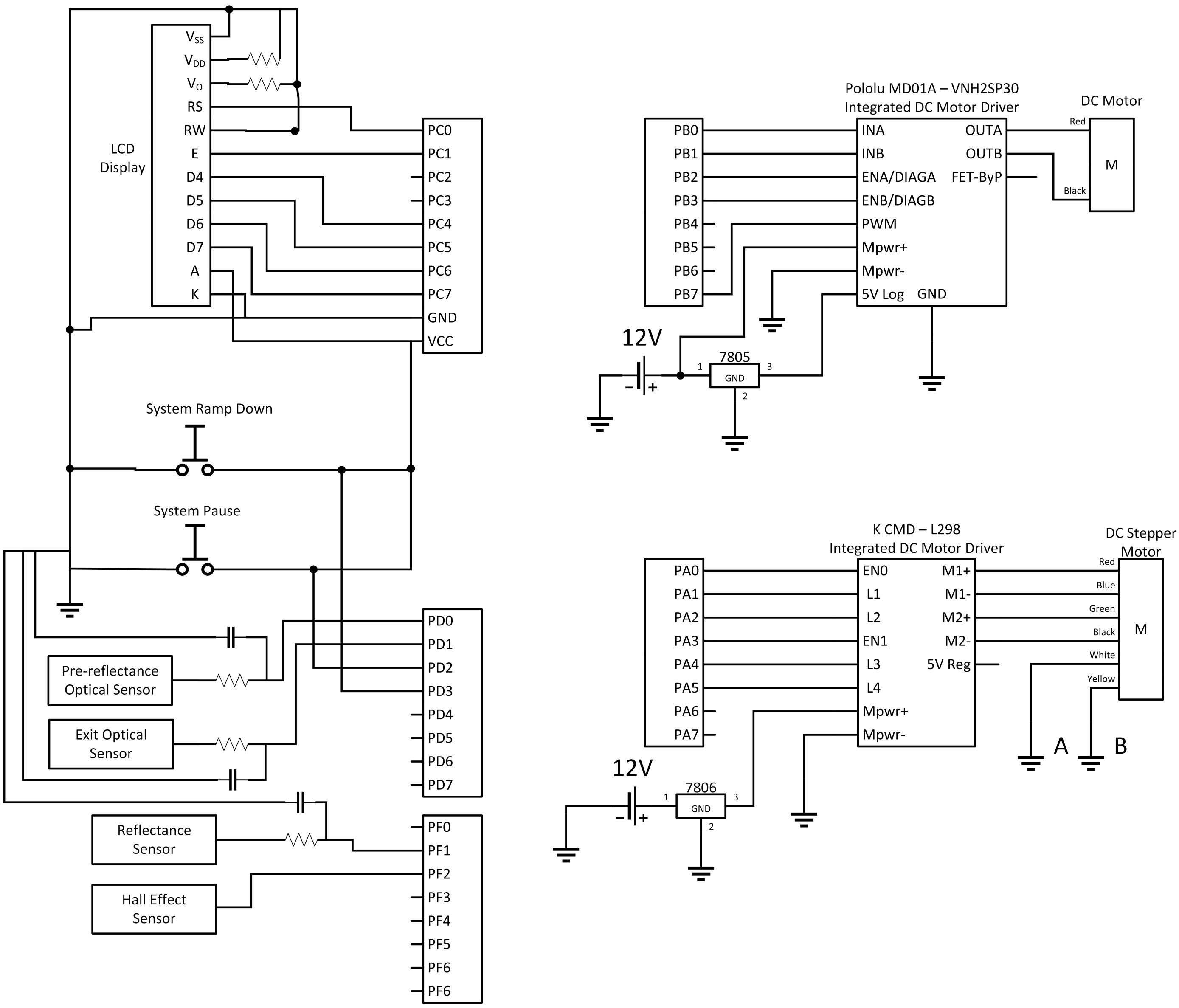


Figure 5: Circuit diagram of complete system

4.3 System Algorithm/Flowchart

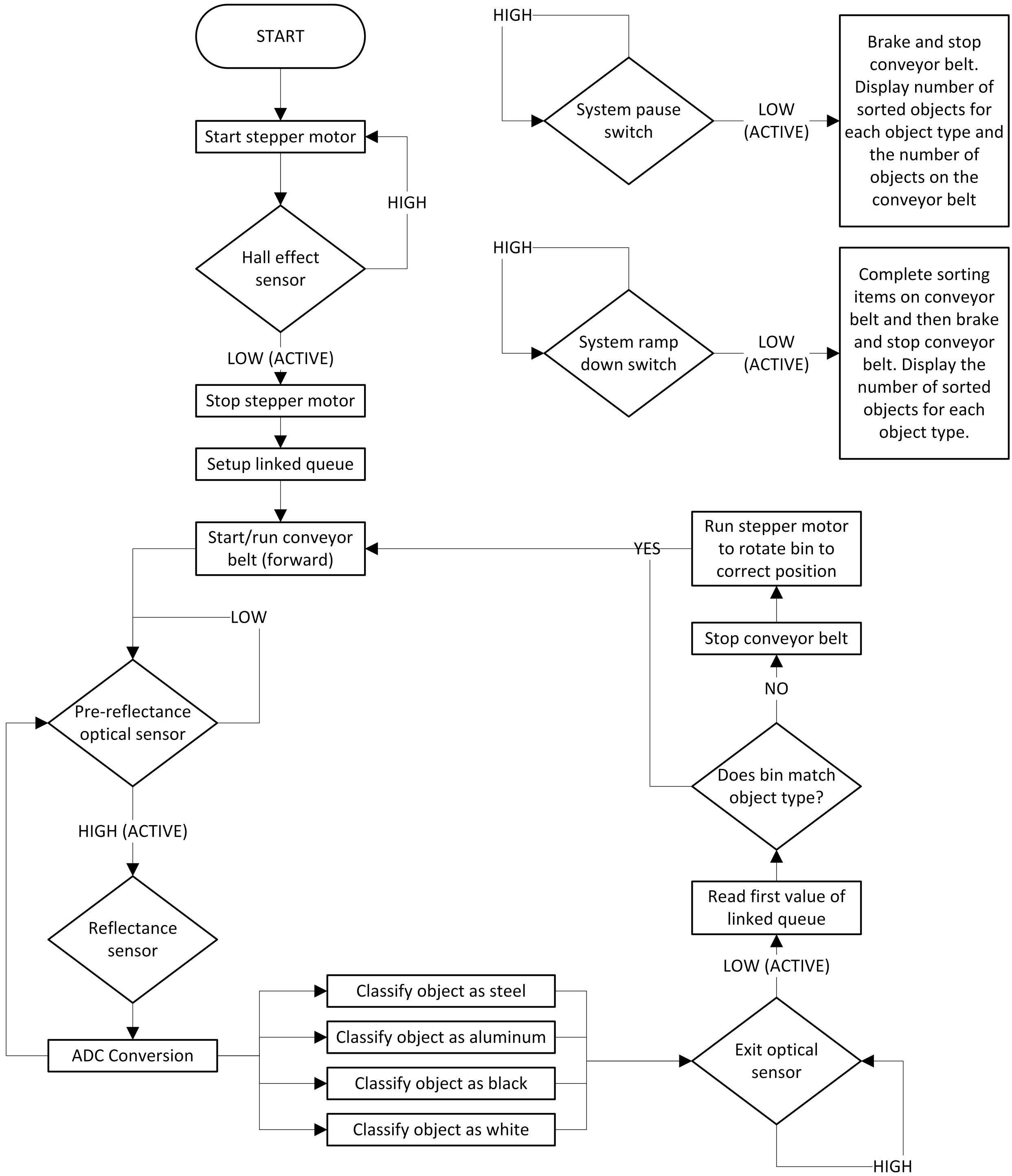


Figure 6: System algorithm/flowchart

4.4 System Operation

The inspection system uses external power with an input voltage of 6.6V for the stepper motor and DC motor. Initially, the power to the system is turned on and then the MCU is reset. The system starts with setting the stepper motor to its home position. Then the conveyor belt is started and the inspection system is ready for loading items.

Then, items are loaded onto the conveyor belt in batches, up to 8 items at a time. The items can be placed as close as possible, as the guides provided on the conveyor, will ensure sufficient gap between adjacent items. As the items pass through the sensor stations, the analog sensor reads the reflectance value of the item. The MCU then identifies the type of the item based on the reflectance value and updates the information in a FIFO dynamic linked list. As the item reaches the exit optical sensor, the first item on the linked list is dequeued, and the object type is compared to the current bin position. If required, the conveyor is stopped and the stepper is rotated to adjust the bin to its correct position and then the conveyor belt is restarted and the object is allowed to fall and collected in the appropriate bin.

Due to the limitation of the code in handling the linked list, we had to ensure that the last object of the current batch of 8 items had reached the exit optical sensor before the first item of the next batch of 8 items reached the pre-reflectance optical sensor.

4.5 Technical Description

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

- DC Motor

  A brushed DC motor is used to drive the conveyor belt that which carry the items that are to be sorted. The speed of the motor is controlled using a PWM signal. Dual full bridge driver L298 which can accept standard TTL logic levels is used to drive the DC motor. Two enable inputs are available in the interfacing circuit to enable or disable the DC motor independently of their input signals.

- Pulse Width Modulation (PWM)

  Pulse Width Modulation (PWM) helps the MCU to generate a signal with a voltage anywhere between 0V and 3.3V rather than the discrete values of 0V and 3.3V. This can be used to control the input voltage to devices such as speakers, motors, and LEDs. Based on the input voltage the output of the particular device vary. For this project, the PWM signal is given to the DC motor. The speed of the DC motor is dependent on the duty cycle of the PWM signal.

  The 8-bit timer of the MCU is used to generate the PWM signal. The timer is set to fast PWM. The counter TCNT0 is used together with OCR0A to set the duty cycle. The PWM signal is received at the PIN 7 of PORT B.

- Timer Delay

A timer function is created to force a delay in the processing. The delay is required for the operation of the stepper motor, for the display of information using LEDs during the testing phase, for the system ramp-down function and for synchronizing the movement of the conveyor belt that makes sure that the previous object is dropped before the belt is stopped for the next item.The 16 bit counter TCNT1 and output compare register OCR1A are used to create a unit delay of 1 milliseconds.

- Stepper Motor

  A 6V 0.8A unipolar stepper motor is used to rotate the tray for collecting the falling items in the correct bin. The stepper has a step angle of 1.8 degrees. Though the stepper is unipolar, it is used in a bipolar mode for maximum torque.

- Analog to Digital Conversion (ADC)

  Analog reflectance sensor is used to measure the reflectance value of the items to be sorted. 10-bit resolution is used for the ADC. Since the ADC requires a clock frequency between 50 KHz and 200 KHz for maximum resolution, a pre-scale of 16 is selected for the ADC using the register ADCSRA. The ADC result is read from the ADC data registers ADCL and ADCH.ADC interrupt is enabled to trigger an interrupt on completion of one ADC so that another ADC can be started if the item is in the range of the sensor.

- Pre-Reflectance Optical Sensor

  It is an active high digital sensor that is used to check if an item is in the range of the reflectance sensor. An ADC is started only when an item is in the range of the analog sensor. This helps to avoid a free running Analog to Digital Conversion which would have taken a lot of CPU time. A filter is used to filter out the noise created by the moving parts used in the project.

- Reflectance Sensor

  Reflectance sensor is an analog sensor that measures the reflectance value of the items which are to be sorted. It is the most critical component in identifying the correct type of the object. A 10 bit ADC resolution is used to read the reflectance value of the four objects used in this project. A filter is used in the sensor circuit for filtering out the noise caused by the stepper motor and the DC motor.

- Exit Optical Sensor

  It is an active-low digital sensor used to identify if an object has reached the end of the conveyor belt. An interrupt is triggered on the exit optical sensor detection that ensures that the bin is correctly positioned before dropping the item off the conveyor.

- Hall-Effect Sensor

  It is an active low digital sensor used to set the bin to its home position. All other bin locations are defined based on their relative positions from the home bin. The tray is rotated 90 degrees clockwise, 90 degrees counterclockwise or 180 degrees as required for adjusting the bin to its correct position before dropping the object from the conveyor belt.

- System Pause Push-button

  A push-button is used as the system pause button which disables the DC motor and displays information about the sorted items in each bin and also about the number of items that are partially processed. The system resumes sorting once the pause button is pressed again.

- System Ramp-Down Push-Button

  Another pushbutton is used as the system ramp-down button which when activated, will sort all the items remaining on the conveyor belt and then shuts down the system. The number of objects collected in each bin is displayed on the LCD after the system is shut down. The inspection system needs to be reset before restarting.

- LCD Display An LCD display is used to display the number of sorted objects individually by its type and also the number of unsorted objects.

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 sorted 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 it as the incorrect object type. We used a LCD display for the final demonstration to display the details of items sorted. However, for the calibration of the relectance sensor, we used 8 LEDs on PORTC and the 2 inbuilt LEDs on PORTD as this was only a one-time activity before start-up of the system. A separate code was maintained for calibration. The 10-bit ADC values were stored in a spreadsheet for each item type and after multiple trials, the range of reflectance values corresponding to each object type were identified and updated on the main code.

The reflectance range of steel, aluminum, and the plastics were sufficiently spread apart and hence the system never gave any errors in differentiating between them. However, the range of black plastic and white plastic were very close and hence frequent calibrations were required to make sure that the system identified the 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, about 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 low-pass 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 speeds were 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, the reflectance sensor was giving less accurate readings due to less available time for ADC to read the reflectance sensor values.

7 Limitations and System Trade-offs

The major limitations and trade-offs of the design are summarized below:

- The major limitation of the system was that it could not sort items with continuous loading. The items needed to be loaded in batches of 8 for the inspection system to work accurately. This was due to a limitation in the code which was unable to handle two simultaneous interrupts - one by the exit optical sensor to dequeue the linked list and one by the pre-reflectance optical sensor to initiate a new link in the FIFO linked list. This significantly increased the time required to sort the items.
- The inspection system only used the reflectance sensor for differentiating the different items. Though it was sufficient for identifying object type between steel, aluminum and plastic materials, it was not always very accurate in differentiating between white plastic and black plastic items as their reflectance values were 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 shorter available time for the ADC and also due to the items slipping off the belt when stopped at the exit sensor. This compromised on the time required for sorting items as the travel time was higher.
- The acceleration of the stepper was 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

LCD Display

The system pause and system ramp-down functions need to display the details of the items sorted in each bin and the number of partially sorted items if any. The project objectives require using LEDs to display these numbers. However, an option is given to display the information on an LCD display which would fetch additional marks. We successfully implemented the LCD display and showed all the required information on the LCD display rather than using the LEDs.

From a technical perspective, the LCD was implemented using the SPI library and connected to PORT C of the MCU, in place of the original 8 LEDs. An open-source library was found from the Internet, which allowed us to easily implement the LCD display. The library code is shown in Listing 12 of Appendix B.

The LCD was connected to the MCU as shown in the circuit diagram in Figure 5. The LCD and its pins are shown in Figure 7 and its pin descriptions are given in Table 1.

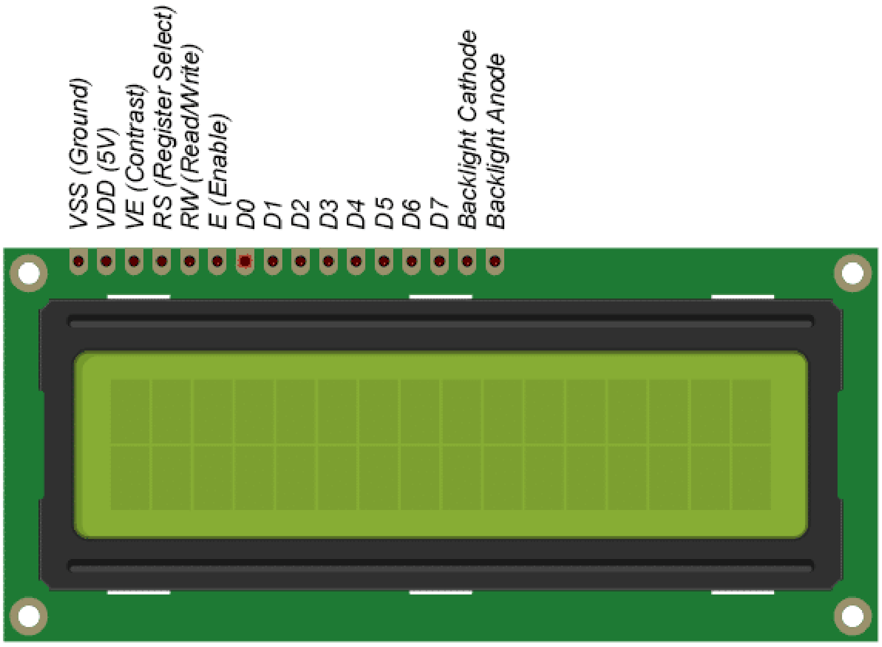


Figure 7: LCD pins

Pin	Description
VSS	Ground (0V)
VDD	Supply voltage (5V)
VE	Contrast adjustment
RS	Register Select; selects the command register or display register
RW	Read/Write; selects to write to the register or to read from the register
E	Enable; sends data to data pins
D0-D7	8-bit data pins (only pins D4-D7 were used in the project)
K	Backlight cathode; backlight VCC (5V)
A	Backlight anode; backlight ground (0V)

Table 1: LCD pin descriptions

9 Experience and Recommendations

Our overall experience in this project was very positive. We gained many new skills related to embedded systems, programming, and mechatronics.

Although we consider the conveyor system to he well-designed and robust, some improvements could be made with the sensors. The sensors sometimes provided incorrect readings even after using low-pass filters. With regards to the whole lab-experience, we have no recommendations. All labs and milestones were well-designed to allow students to learn effectively.

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. The 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 was 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 was enormous. We got a chance to learn about the microcontroller architecture, Analog-to-Digital Converter (ADC), Pulse Width Modulation (PWM), 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 registers 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**
Sensor calibration	Rushi & Sreejith
Linked list implementation	Rushi & Sreejith
Stepper motor acceleration & deceleration	Sreejith
LCD display	Rushi
System pause & system ramp-down push-button implementation	Rushi & Sreejith
Testing & debugging	Rushi & Sreejith
Project timeline	Sreejith
Block diagram, circuit diagram, & system algorithm/flowchart	Rushi

Table 2: Summary of contributions

Appendix B Source Code

```c
#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

```c
#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
```

```c
    sei();  // Note this sets the Global Enable for all interrupts

    goto POLLING_STAGE;

    // POLLING STATE
    POLLING_STAGE:
        switch(STATE){
            case (0) :
                goto POLLING_STAGE;
                break;  //not needed but syntax is correct
            case (1) :
                goto PAUSE_STAGE;
                break;
            case (2) :
                goto RAMPDOWN_STAGE;
                break;
            case (5) :
                goto END;
            default :
                goto POLLING_STAGE;
        }//switch STATE

    // PAUSE STATE
    PAUSE_STAGE:
        //Reset the state variable
        STATE = 0;
        goto POLLING_STAGE;

    // RAMPDOWN STATE
    RAMPDOWN_STAGE:
        mTimer(6000);
        PORTB = 0x00;
        cli();
        printLCD();

        //Reset the state variable
        STATE = 0;
        goto POLLING_STAGE;

    END:
        // The closing STATE ... how would you get here? ANS: When ramp down button is pressed?
        PORTA = 0xF0;   // Indicates this state is active
        // Stop everything here...'MAKE SAFE'
    return(0);
}

// INT0 is triggered by pre-reflectance optical sensor
ISR(INT0_vect){
    ADCSRA |= _BV(ADSC); // initialize the ADC, start one conversion
    minADC = 0xffff;
    objectCount += 1;
}

// triggered when each ADC conversion is complete
ISR(ADC_vect){
    ADC_result = ADCL | ADCH<<8;
    if(ADC_result < minADC){
        minADC = ADC_result;
    }

    if((PIND & 0b00000001) == 1){
        ADCSRA |= _BV(ADSC); // initialize the ADC, start one conversion
    } else {
        initLink(&newLink);
        newLink->o.reflectance = minADC;
        enqueue(&head, &tail, &newLink);
```

```
        if((newLink->o.reflectance >= minReflectanceAluminum) && (newLink->o.reflectance < maxReflectanceAluminum)){
            newLink->o.objectType = 0;
        } else if ((newLink->o.reflectance >= minReflectanceSteel) && (newLink->o.reflectance < maxReflectanceSteel)){
            newLink->o.objectType = 1;
        } else if ((newLink->o.reflectance >= minReflectanceWhite) && (newLink->o.reflectance < maxReflectanceWhite)){
            newLink->o.objectType = 2;
        } else if ((newLink->o.reflectance >= minReflectanceBlack) && (newLink->o.reflectance < maxReflectanceBlack)){
            newLink->o.objectType = 3;
        } else {
            while(1){
                stopDCMotor();
            }
        }
        STATE = 0;
    }
}

// INT1 triggered by exit optical sensor
ISR(INT1_vect){
    //mTimer(30);
    dequeue(&head, &tail, &rtnLink);
    if(rtnLink->o.objectType == 0){ //  PORTC = 1;
        if(currentBin == 0){
            // do nothing
        } else if(currentBin == 1){
            stopDCMotor();
            rotate_cw(100);
        } else if(currentBin == 2){
            stopDCMotor();
            rotate_ccw(50);
        } else if(currentBin == 3){
            stopDCMotor();
            rotate_cw(50);
        }
        currentBin = 0;
    } else if(rtnLink->o.objectType == 1){ //   PORTC = 2;
        if(currentBin == 0){
            stopDCMotor();
            rotate_cw(100);
        } else if(currentBin == 1){
            // do nothing
        } else if(currentBin == 2){
            stopDCMotor();
            rotate_cw(50);
        } else if(currentBin == 3){
            stopDCMotor();
            rotate_ccw(50);
        }
        currentBin = 1;
    } else if(rtnLink->o.objectType == 2){ //   PORTC = 3;
        if(currentBin == 0){
            stopDCMotor();
            rotate_cw(50);
        } else if(currentBin == 1){
            stopDCMotor();
            rotate_ccw(50);
        } else if(currentBin == 2){
            // do nothing
        } else if(currentBin == 3){
            stopDCMotor();
            rotate_cw(100);
        }
```

```
        currentBin = 2;
    } else if(rtnLink->o.objectType == 3){  //      PORTC = 4;
        if(currentBin == 0){
            stopDCMotor();
            rotate_ccw(50);
        } else if(currentBin == 1){
            stopDCMotor();
            rotate_cw(50);
        } else if(currentBin == 2){
            stopDCMotor();
            rotate_cw(100);
        } else if(currentBin == 3){
            // do nothing;
        }
        currentBin = 3;
    }

    startDCMotor();
    mTimer(20);

    if(rtnLink->o.objectType == 0){ //  PORTC = 1;
        countAlum++;
    } else if(rtnLink->o.objectType == 1){ //   PORTC = 2;
        countSteel++;
    } else if(rtnLink->o.objectType == 2){ //   PORTC = 3;
        countWhite++;
    } else if(rtnLink->o.objectType == 3){  //      PORTC = 4;
        countBlack++;
    }

    objectCount -= 1;
    free(rtnLink);
    STATE = 0;
}

// INT3 triggered by system pause button
ISR(INT2_vect){
    mTimer(30); /* Wait 20ms */
    while((PIND & 0b00000100) == 0);    /* Wait till button is released */
    mTimer(30); /* Wait 20ms */
    if(PAUSE == 0){
        stopDCMotor();
        PAUSE = 1;
        STATE = 1;
        printLCD();
    } else {
        startDCMotor();
        PAUSE = 0;
        STATE = 0;
    }
}

// INT4 triggered by system ramp-down button
ISR(INT3_vect){
    mTimer(30); /* Wait 20ms */
    while((PIND & 0b00001000) == 0);    /* Wait till button is released */
    mTimer(30); /* Wait 20ms */

    //rampDown = 1;
    STATE = 2;
}

ISR(BADISR_vect){
    while(1){
        Lcd4_Init();
        Lcd4_Clear();
```

```c
        Lcd4_Set_Cursor(1,0);
        Lcd4_Write_String("********************" );
        Lcd4_Set_Cursor(2,0);
        Lcd4_Write_String("**      ERROR!      **" );
        Lcd4_Set_Cursor(3,0);
        Lcd4_Write_String("**                  **" );
        Lcd4_Set_Cursor(4,0);
        Lcd4_Write_String("********************" );
    }
}
```

Listing 2: main.c

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

void initPWM();
void initADC();
void initInterrupts();
void initLinkedQueue();
void configSensors(int station);

volatile uint16_t minReflectanceAluminum;
volatile uint16_t maxReflectanceAluminum;
volatile uint16_t minReflectanceSteel;
volatile uint16_t maxReflectanceSteel;
volatile uint16_t minReflectanceWhite;
volatile uint16_t maxReflectanceWhite;
volatile uint16_t minReflectanceBlack;
volatile uint16_t maxReflectanceBlack;
```

Listing 3: init.h

```c
#include "init.h"

void initPWM(){
    /* Generating PWM */
     TCCR0A|=1<<WGM00|1<<WGM01;      /*Set timer to fast PWM mode, update OCRA at TOP=0xFF*/
     //TIMSK0|=0<<OCIE0A;            /*Interrupt is disabled for TCNT0 on A Compare */
     //TIFR0|=1<<OCF0A;              /*Reset TIFR0 and initiate counter */
     TCCR0A|=1<<COM0A1;              /*Set output mode to non-inverted, OC0A clear on compare, set
          at TOP */
     TCCR0B|=1<<CS01;                /*Set Timer pre-scale to 8    */
     OCR0A=90;                       /*Duty Cycle out of 255 */
     DDRB|=1<<DDB7;                  /*Set PORTB PIN7 as output to enable OC0A to control PIN
         output */
}

void initADC(){
    // by default, the ADC input (analog input is set to be ADC0 / PORTF0
    ADCSRA |= _BV(ADEN); // enable ADC
    ADCSRA |= _BV(ADIE); // enable interrupt of ADC
    ADCSRA |= _BV(ADPS2); // set ADC prescaler to 128
    ADMUX |= _BV(REFS0); // set reference voltage to VCC
    ADMUX |= _BV(MUX0); // use ADC channel 1
    // ADLAR = 0 (default)
}

void initInterrupts(){
    // config the external interrupt ======================================
    EIMSK|= _BV(INT0); // enable INT0 for pre-reflectance optical sensor
    EIMSK|= _BV(INT1); // enable INT1 for exit optical sensor
    EIMSK|= _BV(INT2); // enable INT2 for system pause PB
    EIMSK|= _BV(INT3); // enable INT3 for system ramp-down PB

    EICRA |= _BV(ISC01) | _BV(ISC00); // rising edge INT0 for pre-reflectance optical sensor
```

```c
    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

```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

```c
#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 */
```

```
            }
            position = 3;
        } else if (position==3){
            PORTA =0b00101011; /* half step clockwise rotation through 0.9 degrees, M2, L3 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 = 4;
        }// end if
    } /* end while */
    return(0);
} /* end rotate_ccw */

//function for counter clockwise rotation
int rotate_ccw(int steps){
    int count=0;
    int delay;
    int acceleration=1;
    int delta;
    delta=maxdelay-mindelay;
    delay=maxdelay;
    while(steps>count){
        if(position==4){
            PORTA= 0b00101011; /*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 = 0b00101101; /* 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=0b00011101; /*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 = 3;
        } else if (position==3){
            PORTA =0b00011011; /* half step clockwise rotation through 0.9 degrees, M2, L3
                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 = 4;
        }
    } /* end while */
    return(0);
} /* end rotate_ccw */
```

Listing 6: motors.c

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

void mTimer(int count);
```

Listing 7: mTimer.h

```
#include "mTimer.h"

//##########################################################################
//function for timer in milliseconds
void mTimer(int count){
    TCCR1B|=(1<<CS10)|(1<<WGM12); //set timer control prescaling to none and mode
    //to Clear Time on Compare, CTC.
    OCR1A=1000; // Output compare register is set to 1000D.
    TCNT1=0; // Initialize counter
    //TIMSK1|= (0<<1);//Timer counter output compare A match is disabled.
    TIFR1|=(1<<OCF1A); //Timer counter interrupt flag is cleared and timer is started.

    int i=0; // for counting the loop
    while(i<count){
        if((TIFR1&(0b00000010))==0b00000010){
            TIFR1|=(1<<OCF1A); //Timer counter interrupt flag is cleared.
            i++;
        }//if
    }//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*/

```

```c
/***************************************************************************************
*  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*/
```

```
    /* Last but not least set the tail to NULL */
    *t = NULL;
    return;
}/*clearQueue*/

/*****************************************************************************************
* DESC: Checks to see whether the queue is empty or not
* INPUT: The head pointer
* RETURNS: 1:if the queue is empty, and 0:if the queue is NOT empty
*/
/* Check to see if the queue is empty */
char isEmpty(link **h){
    /* ENTER YOUR CODE HERE */
    return(*h == NULL);
}/*isEmpty*/

/*****************************************************************************************
* DESC: Obtains the number of links in the queue
* INPUT: The head and tail pointer
* RETURNS: An integer with the number of links in the queue
*/
/* returns the size of the queue*/
int size(link **h, link **t){
    link    *temp;          /* will store the link while traversing the queue */
    int     numObjects;
    numObjects = 0;
    temp = *h;          /* point to the first item in the list */
    while(temp != NULL){
        numObjects++;
        temp = temp->next;
    }/*while*/
    return(numObjects);
}/*size*/
```

Listing 10: LinkedQueue.c

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

```c
#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

```c
#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);
```

```
        break;

        case 7:
        Lcd4_Set_Cursor(1,6);
        Lcd4_Write_Char(0x37);
        break;

        case 8:
        Lcd4_Set_Cursor(1,6);
        Lcd4_Write_Char(0x38);
        break;

        case 9:
        Lcd4_Set_Cursor(1,6);
        Lcd4_Write_Char(0x39);
        break;

        case 10:
        Lcd4_Set_Cursor(1,6);
        Lcd4_Write_Char(0x31);
        Lcd4_Set_Cursor(1,7);
        Lcd4_Write_Char(0x30);
        break;

        case 11:
        Lcd4_Set_Cursor(1,6);
        Lcd4_Write_Char(0x31);
        Lcd4_Set_Cursor(1,7);
        Lcd4_Write_Char(0x31);
        break;

        case 12:
        Lcd4_Set_Cursor(1,6);
        Lcd4_Write_Char(0x31);
        Lcd4_Set_Cursor(1,7);
        Lcd4_Write_Char(0x32);
        break;

        default:
        Lcd4_Set_Cursor(1,6);
        Lcd4_Write_Char(0x30);
    }

    switch (countWhite){
        case 0:
        Lcd4_Set_Cursor(1,14);
        Lcd4_Write_Char(0x30);
        break;

        case 1:
        Lcd4_Set_Cursor(1,14);
        Lcd4_Write_Char(0x31);
        break;

        case 2:
        Lcd4_Set_Cursor(1,14);
        Lcd4_Write_Char(0x32);
        break;

        case 3:
        Lcd4_Set_Cursor(1,14);
        Lcd4_Write_Char(0x33);
        break;

        case 4:
        Lcd4_Set_Cursor(1,14);
```

```
        Lcd4_Write_Char(0x34);
        break;

        case 5:
        Lcd4_Set_Cursor(1,14);
        Lcd4_Write_Char(0x35);
        break;

        case 6:
        Lcd4_Set_Cursor(1,14);
        Lcd4_Write_Char(0x36);
        break;

        case 7:
        Lcd4_Set_Cursor(1,14);
        Lcd4_Write_Char(0x37);
        break;

        case 8:
        Lcd4_Set_Cursor(1,14);
        Lcd4_Write_Char(0x38);
        break;

        case 9:
        Lcd4_Set_Cursor(1,14);
        Lcd4_Write_Char(0x39);
        break;

        case 10:
        Lcd4_Set_Cursor(1,14);
        Lcd4_Write_Char(0x31);
        Lcd4_Set_Cursor(1,15);
        Lcd4_Write_Char(0x30);
        break;

        case 11:
        Lcd4_Set_Cursor(1,14);
        Lcd4_Write_Char(0x31);
        Lcd4_Set_Cursor(1,15);
        Lcd4_Write_Char(0x31);
        break;

        case 12:
        Lcd4_Set_Cursor(1,14);
        Lcd4_Write_Char(0x31);
        Lcd4_Set_Cursor(1,15);
        Lcd4_Write_Char(0x32);
        break;

        default:
        Lcd4_Set_Cursor(1,14);
        Lcd4_Write_Char(0x30);
    }

    //ALUMINUM, STEEL AND UNSORTED
    Lcd4_Set_Cursor(2,0);
    Lcd4_Write_String("ALUM=    STL=       UNSORTED = " );

    switch (countAlum){
        case 0:
        Lcd4_Set_Cursor(2,6);
        Lcd4_Write_Char(0x30);
        break;

        case 1:
        Lcd4_Set_Cursor(2,6);
```

```
        Lcd4_Write_Char(0x31);
        break;

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

        case 3:
        Lcd4_Set_Cursor(2,6);
        Lcd4_Write_Char(0x33);
        break;

        case 4:
        Lcd4_Set_Cursor(2,6);
        Lcd4_Write_Char(0x34);
        break;

        case 5:
        Lcd4_Set_Cursor(2,6);
        Lcd4_Write_Char(0x35);
        break;

        case 6:
        Lcd4_Set_Cursor(2,6);
        Lcd4_Write_Char(0x36);
        break;

        case 7:
        Lcd4_Set_Cursor(2,6);
        Lcd4_Write_Char(0x37);
        break;

        case 8:
        Lcd4_Set_Cursor(2,6);
        Lcd4_Write_Char(0x38);
        break;

        case 9:
        Lcd4_Set_Cursor(2,6);
        Lcd4_Write_Char(0x39);
        break;

        case 10:
        Lcd4_Set_Cursor(2,6);
        Lcd4_Write_Char(0x31);
        Lcd4_Set_Cursor(2,7);
        Lcd4_Write_Char(0x30);
        break;

        case 11:
        Lcd4_Set_Cursor(2,6);
        Lcd4_Write_Char(0x31);
        Lcd4_Set_Cursor(2,7);
        Lcd4_Write_Char(0x31);
        break;

        case 12:
        Lcd4_Set_Cursor(2,6);
        Lcd4_Write_Char(0x31);
        Lcd4_Set_Cursor(2,7);
        Lcd4_Write_Char(0x32);
        break;

        default:
        Lcd4_Set_Cursor(2,6);
```

```c
        Lcd4_Write_Char(0x30);
    }

    switch (countSteel){
        case 0:
        Lcd4_Set_Cursor(2,14);
        Lcd4_Write_Char(0x30);
        break;

        case 1:
        Lcd4_Set_Cursor(2,14);
        Lcd4_Write_Char(0x31);
        break;

        case 2:
        Lcd4_Set_Cursor(2,14);
        Lcd4_Write_Char(0x32);
        break;

        case 3:
        Lcd4_Set_Cursor(2,14);
        Lcd4_Write_Char(0x33);
        break;

        case 4:
        Lcd4_Set_Cursor(2,14);
        Lcd4_Write_Char(0x34);
        break;

        case 5:
        Lcd4_Set_Cursor(2,14);
        Lcd4_Write_Char(0x35);
        break;

        case 6:
        Lcd4_Set_Cursor(2,14);
        Lcd4_Write_Char(0x36);
        break;

        case 7:
        Lcd4_Set_Cursor(2,14);
        Lcd4_Write_Char(0x37);
        break;

        case 8:
        Lcd4_Set_Cursor(2,14);
        Lcd4_Write_Char(0x38);
        break;

        case 9:
        Lcd4_Set_Cursor(2,14);
        Lcd4_Write_Char(0x39);
        break;

        case 10:
        Lcd4_Set_Cursor(2,14);
        Lcd4_Write_Char(0x31);
        Lcd4_Set_Cursor(2,15);
        Lcd4_Write_Char(0x30);
        break;

        case 11:
        Lcd4_Set_Cursor(2,14);
        Lcd4_Write_Char(0x31);
        Lcd4_Set_Cursor(2,15);
        Lcd4_Write_Char(0x31);
```

```
        break;

        case 12:
        Lcd4_Set_Cursor(2,14);
        Lcd4_Write_Char(0x31);
        Lcd4_Set_Cursor(2,15);
        Lcd4_Write_Char(0x32);
        break;

        default:
        Lcd4_Set_Cursor(2,14);
        Lcd4_Write_Char(0x30);
    }

    switch (objectCount){
        case 0:
        Lcd4_Set_Cursor(2,31);
        Lcd4_Write_Char(0x30);
        break;

        case 1:
        Lcd4_Set_Cursor(2,31);
        Lcd4_Write_Char(0x31);
        break;

        case 2:
        Lcd4_Set_Cursor(2,31);
        Lcd4_Write_Char(0x32);
        break;

        case 3:
        Lcd4_Set_Cursor(2,31);
        Lcd4_Write_Char(0x33);
        break;

        case 4:
        Lcd4_Set_Cursor(2,31);
        Lcd4_Write_Char(0x34);
        break;

        case 5:
        Lcd4_Set_Cursor(2,31);
        Lcd4_Write_Char(0x35);
        break;

        case 6:
        Lcd4_Set_Cursor(2,31);
        Lcd4_Write_Char(0x36);
        break;

        case 7:
        Lcd4_Set_Cursor(2,31);
        Lcd4_Write_Char(0x37);
        break;

        case 8:
        Lcd4_Set_Cursor(2,31);
        Lcd4_Write_Char(0x38);
        break;

        case 9:
        Lcd4_Set_Cursor(2,31);
        Lcd4_Write_Char(0x39);
        break;

        case 10:
```

```c
        Lcd4_Set_Cursor(2,31);
        Lcd4_Write_Char(0x31);
        Lcd4_Set_Cursor(2,32);
        Lcd4_Write_Char(0x30);
        break;

        case 11:
        Lcd4_Set_Cursor(2,31);
        Lcd4_Write_Char(0x31);
        Lcd4_Set_Cursor(2,32);
        Lcd4_Write_Char(0x31);
        break;

        case 12:
        Lcd4_Set_Cursor(2,31);
        Lcd4_Write_Char(0x31);
        Lcd4_Set_Cursor(2,32);
        Lcd4_Write_Char(0x32);
        break;

        default:
        Lcd4_Set_Cursor(2,31);
        Lcd4_Write_Char(0x30);
    }
}

void pinChange(int a, int b)
{
    if(b == 0)
    {
        if(a == eS_PORTA0)
        PORTA &= ~(1<<PA0);
        else if(a == eS_PORTA1)
        PORTA &= ~(1<<PA1);
        else if(a == eS_PORTA2)
        PORTA &= ~(1<<PA2);
        else if(a == eS_PORTA3)
        PORTA &= ~(1<<PA3);
        else if(a == eS_PORTA4)
        PORTA &= ~(1<<PA4);
        else if(a == eS_PORTA5)
        PORTA &= ~(1<<PA5);
        else if(a == eS_PORTA6)
        PORTA &= ~(1<<PA6);
        else if(a == eS_PORTA7)
        PORTA &= ~(1<<PA7);
        else if(a == eS_PORTB0)
        PORTB &= ~(1<<PB0);
        else if(a == eS_PORTB1)
        PORTB &= ~(1<<PB1);
        else if(a == eS_PORTB2)
        PORTB &= ~(1<<PB2);
        else if(a == eS_PORTB3)
        PORTB &= ~(1<<PB3);
        else if(a == eS_PORTB4)
        PORTB &= ~(1<<PB4);
        else if(a == eS_PORTB5)
        PORTB &= ~(1<<PB5);
        else if(a == eS_PORTB6)
        PORTB &= ~(1<<PB6);
        else if(a == eS_PORTB7)
        PORTB &= ~(1<<PB7);
        else if(a == eS_PORTC0)
        PORTC &= ~(1<<PC0);
        else if(a == eS_PORTC1)
        PORTC &= ~(1<<PC1);
```

```c
        else if(a == eS_PORTC2)
        PORTC &= ~(1<<PC2);
        else if(a == eS_PORTC3)
        PORTC &= ~(1<<PC3);
        else if(a == eS_PORTC4)
        PORTC &= ~(1<<PC4);
        else if(a == eS_PORTC5)
        PORTC &= ~(1<<PC5);
        else if(a == eS_PORTC6)
        PORTC &= ~(1<<PC6);
        else if(a == eS_PORTC7)
        PORTC &= ~(1<<PC7);
        else if(a == eS_PORTD0)
        PORTD &= ~(1<<PD0);
        else if(a == eS_PORTD1)
        PORTD &= ~(1<<PD1);
        else if(a == eS_PORTD2)
        PORTD &= ~(1<<PD2);
        else if(a == eS_PORTD3)
        PORTD &= ~(1<<PD3);
        else if(a == eS_PORTD4)
        PORTD &= ~(1<<PD4);
        else if(a == eS_PORTD5)
        PORTD &= ~(1<<PD5);
        else if(a == eS_PORTD6)
        PORTD &= ~(1<<PD6);
        else if(a == eS_PORTD7)
        PORTD &= ~(1<<PD7);
    } else {
        if(a == eS_PORTA0)
        PORTA |= (1<<PA0);
        else if(a == eS_PORTA1)
        PORTA |= (1<<PA1);
        else if(a == eS_PORTA2)
        PORTA |= (1<<PA2);
        else if(a == eS_PORTA3)
        PORTA |= (1<<PA3);
        else if(a == eS_PORTA4)
        PORTA |= (1<<PA4);
        else if(a == eS_PORTA5)
        PORTA |= (1<<PA5);
        else if(a == eS_PORTA6)
        PORTA |= (1<<PA6);
        else if(a == eS_PORTA7)
        PORTA |= (1<<PA7);
        else if(a == eS_PORTB0)
        PORTB |= (1<<PB0);
        else if(a == eS_PORTB1)
        PORTB |= (1<<PB1);
        else if(a == eS_PORTB2)
        PORTB |= (1<<PB2);
        else if(a == eS_PORTB3)
        PORTB |= (1<<PB3);
        else if(a == eS_PORTB4)
        PORTB |= (1<<PB4);
        else if(a == eS_PORTB5)
        PORTB |= (1<<PB5);
        else if(a == eS_PORTB6)
        PORTB |= (1<<PB6);
        else if(a == eS_PORTB7)
        PORTB |= (1<<PB7);
        else if(a == eS_PORTC0)
        PORTC |= (1<<PC0);
        else if(a == eS_PORTC1)
        PORTC |= (1<<PC1);
        else if(a == eS_PORTC2)
```

```
        PORTC |= (1<<PC2);
        else if(a == eS_PORTC3)
        PORTC |= (1<<PC3);
        else if(a == eS_PORTC4)
        PORTC |= (1<<PC4);
        else if(a == eS_PORTC5)
        PORTC |= (1<<PC5);
        else if(a == eS_PORTC6)
        PORTC |= (1<<PC6);
        else if(a == eS_PORTC7)
        PORTC |= (1<<PC7);
        else if(a == eS_PORTD0)
        PORTD |= (1<<PD0);
        else if(a == eS_PORTD1)
        PORTD |= (1<<PD1);
        else if(a == eS_PORTD2)
        PORTD |= (1<<PD2);
        else if(a == eS_PORTD3)
        PORTD |= (1<<PD3);
        else if(a == eS_PORTD4)
        PORTD |= (1<<PD4);
        else if(a == eS_PORTD5)
        PORTD |= (1<<PD5);
        else if(a == eS_PORTD6)
        PORTD |= (1<<PD6);
        else if(a == eS_PORTD7)
        PORTD |= (1<<PD7);
    }
}

//LCD 8 Bit Interfacing Functions
void Lcd8_Port(char a)
{
    if(a & 1)
    pinChange(D0,1);
    else
    pinChange(D0,0);

    if(a & 2)
    pinChange(D1,1);
    else
    pinChange(D1,0);

    if(a & 4)
    pinChange(D2,1);
    else
    pinChange(D2,0);

    if(a & 8)
    pinChange(D3,1);
    else
    pinChange(D3,0);

    if(a & 16)
    pinChange(D4,1);
    else
    pinChange(D4,0);

    if(a & 32)
    pinChange(D5,1);
    else
    pinChange(D5,0);

    if(a & 64)
    pinChange(D6,1);
    else
```

```
    pinChange(D6,0);

    if(a & 128)
    pinChange(D7,1);
    else
    pinChange(D7,0);
}

void Lcd8_Cmd(char a)
{
    pinChange(RS,0);              // => RS = 0
    Lcd8_Port(a);              //Data transfer
    pinChange(EN,1);              // => E = 1
    _delay_ms(1);
    pinChange(EN,0);              // => E = 0
    _delay_ms(1);
}

void Lcd8_Clear()
{
    Lcd8_Cmd(1);
}

void Lcd8_Set_Cursor(char a, char b)
{
    if(a == 1)
    Lcd8_Cmd(0x80 + b);
    else if(a == 2)
    Lcd8_Cmd(0xC0 + b);
}

void Lcd8_Init()
{
    pinChange(RS,0);
    pinChange(EN,0);
    _delay_ms(20);
    ///////////// Reset process from datasheet /////////
    Lcd8_Cmd(0x30);
    _delay_ms(5);
    Lcd8_Cmd(0x30);
    _delay_ms(1);
    Lcd8_Cmd(0x30);
    _delay_ms(10);
    /////////////////////////////////////////////////////
    Lcd8_Cmd(0x38);    //function set
    Lcd8_Cmd(0x0C);    //display on,cursor off,blink off
    Lcd8_Cmd(0x01);    //clear display
    Lcd8_Cmd(0x06);    //entry mode, set increment
}

void Lcd8_Write_Char(char a)
{
    pinChange(RS,1);              // => RS = 1
    Lcd8_Port(a);              //Data transfer
    pinChange(EN,1);              // => E = 1
    _delay_ms(1);
    pinChange(EN,0);              // => E = 04
    _delay_ms(1);
}

void Lcd8_Write_String(char *a)
{
    int i;
    for(i=0;a[i]!='\0';i++)
    Lcd8_Write_Char(a[i]);
}
```

```
void Lcd8_Shift_Right()
{
    Lcd8_Cmd(0x1C);
}

void Lcd8_Shift_Left()
{
    Lcd8_Cmd(0x18);
}
//End LCD 8 Bit Interfacing Functions

//LCD 4 Bit Interfacing Functions

void Lcd4_Port(char a)
{
    if(a & 1)
    pinChange(D4,1);
    else
    pinChange(D4,0);

    if(a & 2)
    pinChange(D5,1);
    else
    pinChange(D5,0);

    if(a & 4)
    pinChange(D6,1);
    else
    pinChange(D6,0);

    if(a & 8)
    pinChange(D7,1);
    else
    pinChange(D7,0);
}
void Lcd4_Cmd(char a)
{
    pinChange(RS,0);             // => RS = 0
    Lcd4_Port(a);
    pinChange(EN,1);            // => E = 1
    _delay_ms(1);
    pinChange(EN,0);             // => E = 0
    _delay_ms(1);
}

void Lcd4_Clear()
{
    Lcd4_Cmd(0);
    Lcd4_Cmd(1);
}

void Lcd4_Set_Cursor(char a, char b)
{
    char temp,z,y;
    if(a == 1)
    {
        temp = 0x80 + b;
        z = temp>>4;
        y = (0x80+b) & 0x0F;
        Lcd4_Cmd(z);
        Lcd4_Cmd(y);
    }
    else if(a == 2)
    {
        temp = 0xC0 + b;
```

```c
        z = temp>>4;
        y = (0xC0+b) & 0x0F;
        Lcd4_Cmd(z);
        Lcd4_Cmd(y);
    }
}

void Lcd4_Init()
{
    Lcd4_Port(0x00);
    _delay_ms(20);
    ///////////// Reset process from datasheet /////////
    Lcd4_Cmd(0x03);
    _delay_ms(5);
    Lcd4_Cmd(0x03);
    _delay_ms(11);
    Lcd4_Cmd(0x03);
    /////////////////////////////////////////////////////
    Lcd4_Cmd(0x02);
    Lcd4_Cmd(0x02);
    Lcd4_Cmd(0x08);
    Lcd4_Cmd(0x00);
    Lcd4_Cmd(0x0C);
    Lcd4_Cmd(0x00);
    Lcd4_Cmd(0x06);
}

void Lcd4_Write_Char(char a)
{
    char temp,y;
    temp = a&0x0F;
    y = a&0xF0;
    pinChange(RS,1);             // => RS = 1
    Lcd4_Port(y>>4);             //Data transfer
    pinChange(EN,1);
    _delay_ms(1);
    pinChange(EN,0);
    _delay_ms(1);
    Lcd4_Port(temp);
    pinChange(EN,1);
    _delay_ms(1);
    pinChange(EN,0);
    _delay_ms(1);
}

void Lcd4_Write_String(char *a)
{
    int i;
    for(i=0;a[i]!='\0';i++)
    Lcd4_Write_Char(a[i]);
}

void Lcd4_Shift_Right()
{
    Lcd4_Cmd(0x01);
    Lcd4_Cmd(0x0C);
}

void Lcd4_Shift_Left()
{
    Lcd4_Cmd(0x01);
    Lcd4_Cmd(0x08);
}
//End LCD 4 Bit Interfacing Functions
```

Listing 12: lcd.c