Final Project Report

Fall 2022

By:

Thomas Zoldowski and Josh Rousseau

EGR 227 – Microcontroller Programming and Applications Course

Section 101

Grand Valley State University

Date Performed: 8 December, 2022

Date Submitted: 10 December, 2022

Instructor: Professor Panek

Table of Contents

List of Tables

Appendices

Introduction and Scope

The purpose of this report is to describe and outline the design, construction and configuration of a machine that dispenses selected items via user interface commonly known as a vending machine (Figure 1). User interface will be integrated by use of a Liquid Crystal Display (LCD) for viewing dispensing directions/options, a Coin slot with an integrated Infrared transmitter and receiver to count funds for item dispensing authorization, a Keypad for user interface with the system regarding selection choices (four total) and visual indicators in the form of Light Emitting Diodes (LEDs) to assist the user with machine interface.

The program will begin with a 'Welcome Message' displaying the Team's names (Figure 2). Dispensing of selected items will be controlled through manipulation of two servo motors. User interface options will be provided via a 'Main Dashboard' (Figure 3) displayed on the LCD. Items offered will be clearly available on a custom advertisement at the machine (Figure 4). The welcome message will clear from the screen and the LCD will display the 'Main Dashboard' which will include a count of currently inserted funds. At this point, the user may insert funds into the machine. The user will then make their desired selection on the keypad. A red LED will be powered on if the minimum requirement of funds is not met and no product will be dispensed. If the minimum requirement of funds has been met, a green LED will be activated and a 'Dispensed' message (Figure 5) will appear on the LCD as the item dispenses. Any additional funds will be tracked for reimbursement. Reimbursed funds will be controlled by activation of a push button with a message informing the user of reimbursement (Figure 6).

Additionally, a display will be provided for Administrators (Admin) to allow for viewing of funds, temperature and inventory (Figure 7). Access to this screen will be secured by a Personal Identification Number (PIN) (Figure 8). After correct PIN input, the administrator will have the option to select viewing of internal machine temperature (Figure 9) and current item inventory (Figure 10). The 'Cash' option will allow for an administrator to set the machines funds back to zero, while the 'Inventory' option will allow an administrator to modify machine inventory.

Equipment and peripherals for this project are outlined in Table 1.

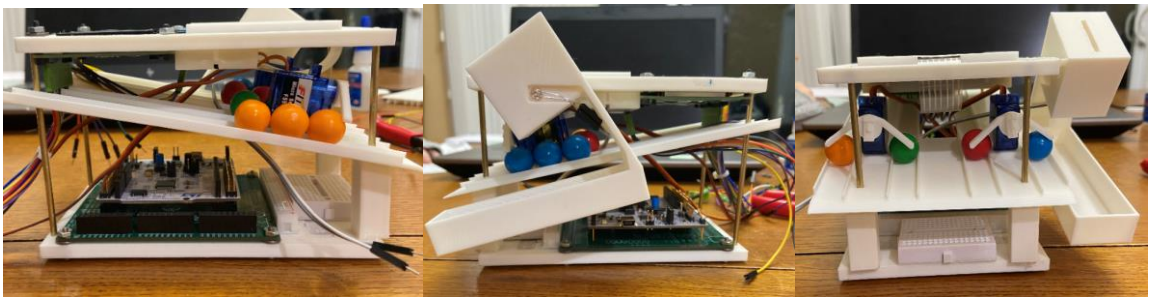


Figure 1: Vending Machine

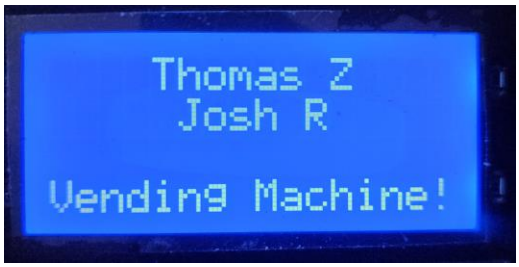


Figure 2: Welcome Screen

Figure 3: Main Dashboard

Figure 5: Dispensed Message

Figure 6: Reimbursement Screen

Figure 7: Admin Screen

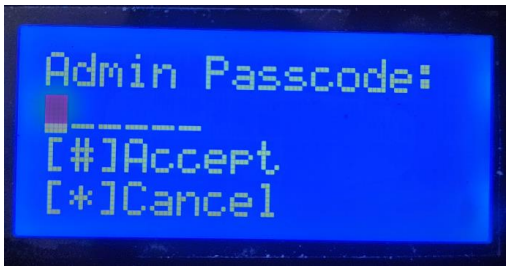


Figure 8: PIN Screen

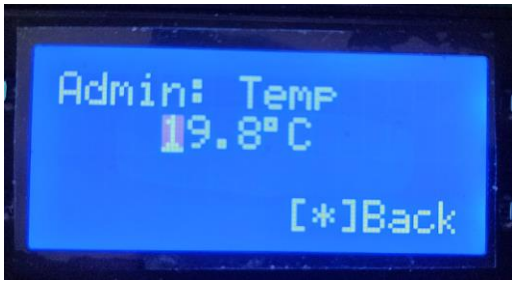


Figure 9: Temperature Screen

Figure 10: Inventory Screen

Table 1: Equipment

EQUIPMENT	QTY	NOTES
STM32F446RETX MICROCONTROLLER	1	KIT SUPPLIED - INTERNAL TEMPERATURE SENSOR USED
GREEN PCB INTERFACE BOARD	1	COURSE SUPPLIED
BREADBOARD	3	KIT SUPPLIED
USB/MINI INTERFACE CABLE	1	KIT SUPPLIED
KEIL Uvision IDE	1	N/A
LAPTOP COMPUTER	1	PERNSONNAL
CONNECTION WIRES		KIT SUPPLIED
POTENTIOMETER	1	10K - KIT SUPPLIED
1604A-V1.2 LIQUID CRYSTAL DISPLAY (LCD)	1	KIT SUPPLIED
PHOTODIODE & INFRARED PAIR	1	KIT SUPPLIED
RESISTOR	2	100 OHM
RESISTOR	3	200 OHM
RESISTOR	1	10K OHM
PUSH BUTTON	1	KIT SUPPLIED
MS18 SERVO MOTOR	2	KIT SUPPLIED - 9g
4x3 MATRIX ARRAY KEYPAD	1	KIT SUPPLIED
TRANSISTOR, MOSFET 27000	1	KIT SUPPLIED
RED GREEN BLUE LIGHT EMMITING DIODE (RGB LED)	1	KIT SUPPLIED
MACHINE BODY	1	SEE APPENDIX B

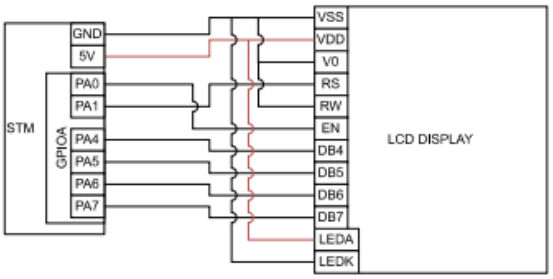
Procedure

The procedure for this project included all aspects of previous laboratory exercises. Freedom was given to allow for creativity with respect of the overall build process. As previously done during the Midterm Project, it was decided to break the project functional requirements down into six manageable parts. This allowed for the overall project to managed and completed as efficiently as possible.

Initially, a flow chart was created to map the projects functionality (Figure 11). After a high-level outline, the project was divided into the six parts of: Keypad and LCD interface, IR receiver/transmitter interface, servo motor control, RGB LED integration, refund push button interface and administration interface.

For Keypad and LCD interface, prior code from laboratory exercises five and six were utilized as a base program and lecture notes and the course literature were used to assist in modifying the program to fit the needs of the project. With the help of these materials, the keypad and LCD interface was able to be accomplished without much trouble. These materials assisted with the pin initializations, timer configurations as well as administrative PIN portion of the program. The program was written, tested and debugged and timers and pins were chosen strategically to ensure further programming would not be hindered. A snippet of the LCD schematic can be seen below in figure 11.

Figure 11. LCD schematic



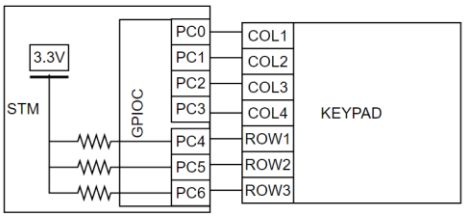
A 'Welcome' message was programmed to appear at program start and remain for three seconds prior to clearing and presenting the 'Main Dashboard'. This was done by taking an array an using a for loop to loop through each index of the array to print each character to the LCD screen. A code snippet of how this was accomplished can be seen below in figure 12.

Figure 12. Welcome Screen To LCD

```
void welcome_screen(void){
char welcome[] =  "    Thomas Z    ";
char welcome1[] = "     Josh R     ";
char welcome2[] = "Vending Machine!";

  Systick_ms_delay(5);
  for(unsigned int i = 0; i< strlen(welcome); i++)
      write_data (welcome[i]);
  write_command(Line_2);
  for(int i = 0; i<16; i++)
      write_data (welcome1[i]);
  write_command(Line_4);
  for(int i = 0; i<16; i++)
      write_data (welcome2[i]);

}
```

The Keypad was used to navigate through the menu as well as select items to purchase from the menu. A snippet of how the keypad was connected to the schematic can be seen below in figure 13.

Figure 13. Keypad Schematic



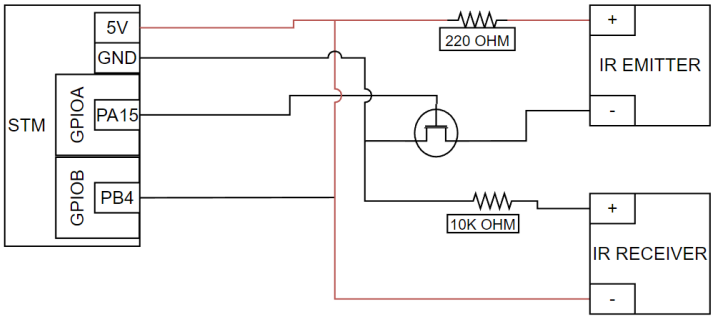
In order to store the correct number, form the key pad, a for loop was used to loop through each column and then check if there was a row that was grounded. It would do this for each column and if a row was grounded, the loop would break, and the row and column would be used to assign the correct key input. A snippet how this was accomplished can be seen below in figure 14.

Figure 14. Keypad input loop

```
  for(col=0; col<3; col++)
  {
      GPIOC->MODER &=~(uint32_t)((3<<(2*Hex2Bit (R0)))|(3<<(2*Hex2Bit (R1)))|(3<<(2*Hex2Bit (R2)))
      |(3<<(2*Hex2Bit (R3)))|(3<<(2*Hex2Bit (C0)))|(3<<(2*Hex2Bit (C1)))|(3<<(2*Hex2Bit (C2))));
      GPIOC->MODER |= (uint32_t)(1<<2*(col+4));              // Set Column i to Output
      GPIOC->ODR   &=~(uint32_t)(1<<(col+4));                // Set Column i to GND
      SysTick_ms_Delay(10);                                     // Delay the while loop
      rows =  GPIOC->IDR & (R0|R1|R2|R3);                    // read all rows: Read must be done b
      while( !(GPIOC->IDR & R0)| !(GPIOC->IDR & R1) | !(GPIOC->IDR & R2) | !(GPIOC->IDR & R3));
      // debouncing loop: if any of the buttons are pressed just put it into the loop

      if (rows != 0x0F) break;                          // if one of the input is GND, some key is
  }
  GPIOC->MODER |= ((1 << (4 * 2)) | (1 << (5 * 2)) | (1 << (6 * 2)));
 // Use the keys array to determine which key was pressed and store it in the key variable
value = nums[row][col];
```

The second part of programming was the set up and interface of the IR receiver/transmitter. The intention of this portion was to house both the receiver and transmitter within the machine to help alleviate any noise from ambient light. With this in mind, the original written program was completed to a point of operation, but not optimization. The IR portion was able to sense, count and store an interrupt of signal, but erroneous counts due to noise and ambient lights were evident. At this time, this issue was not deemed as high risk due to the assumption that when installed into the machine body the shielding of the receiver and transmitter would eliminate most of the outside signal interference. Unfortunately, a failure of the secondary microcontroller that had been installed on the machine body made this a significant problem. However, this problem was able to be overcome with resilient and creative problem solving prior to project deadline. The timer functions that controlled the count of a break in signal was modified to accommodate the additional noise and interference from the ambient environment. This modification was tested and verified with success. Once again, timers and pins were strategically chosen to not hinder further programming. To accomplish this, Timer 2 was used to output a 10Hz single to the emitter and timer 3 was used to capture the signal at 1000KHz. A snippet of the hardware set up can be seen below in figure 15.

Figure 15. IR emitter and receiver



To check if an obstacle was detected, a timing system was used. During tested it was found that when an obstacle was between the emitter and receiver, timer 3's interrupt would stop triggering. This meant that if a counter put inside the interrupt handler that incremented with each cycle, you could compare it to the system counter. If there were different at then an object was detected. A snippet of code is shown below in figure 16 on how this was accomplished.

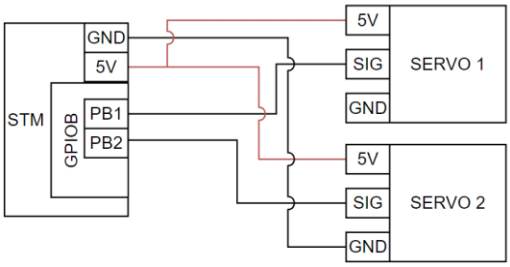
Figure 16. IR obstacle detector code snippet

```
void TIM3_IRQHandler(void){
    if (TIM3->SR & 0x2) {
      last_time = counter;
    TIM3->SR =0;
    }

void SysTick_Handler(void){
  counter++;
  Button_counter++;
  if(counter - last_time > 2000)
    timer3_stopped_updating_flag=1;
  else
    timer3_stopped_updating_flag=0;

}
```

When there is a difference larger the 2000 between the two variables then an obstacle was detected, and the flag was updated.

The third part of programming was the activation and integration of the servo motors. At this time, it was decided that the circuit of the program had been developed enough to create a diagram (Figure 22). Initially, the two servo motors in the circuit were tested separate and offline from the project. This was done since these components had not been used within a laboratory exercise. Offline testing brought confidence and understanding of how a servo motor interfaces with the microcontroller. A function was designed to allow for easy modification of servo movement to ensure ease of integration into the machine body. The servo motors were programmed to move such that the correct item selection would be dispensed and not move if the correct amount of 'cash' was not reach for the respective item. This was done by using two more timers, Timer 10 and 11. Both these timers were configuring the exact same way with both outputting 50hz. A snippet of the schematic can be seen below on how the hardware was configured for both servos in figure 17.

Figure 17. Servo Schematic



This was integrated by creating case statement that check which item was selected and changing the CCR1 register to the correct timer. A snippet of code that accomplished this can be seen below in figure 18.

Figure 18. Servo Position controller

```
switch(num){

  case 1:
    TIM10->CCR1 = 1500;//40 Degrees
  break;
  case 2:
    TIM10->CCR1 = 2500;//50 Degrees
  break;
  case 3:
    TIM11->CCR1 = 1500;//40 Degrees
  break;
  case 4:
    TIM11->CCR1 = 2500;//50 Degrees
  break;
}
```

After the servo had moved, a predefined amount of time would go by, and the servo would move back to 45 degrees.

The next part of programming involved integrating the RGB LED for user notification of authorization of dispensing product. The kit supplied RGB LED was used instead of two separate LEDs to not limit future potential illumination applications. The RGB LED was, once again, tested outside of the project circuit to ensure operation and proper luminosity. Prior laboratory exercises had taught that the kit supplied RGB LED have the capability of producing light bright enough to be deemed uncomfortable. The correct resistance for the RGB LED legs was determined via testing multiple resistors which was found to be 550 ohms. Once a proper brightness was obtained, the RGB LED was programmed to produce a red light if the user made a keypad selection while the amount of 'cash' was less than that of the option minimum and to produce a green light if the user made a keypad selection if the amount of 'cash' was equal to or greater than that of the option minimum. This was done by simple turning on the correct RGB when a certain parameter was met.

The next portion of the programming was to integrate a push button for return of additional 'cash' left over after a selection had been made. The program was designed to simply take the amount of cash that the user had left and "refund" that amount. After the stored cash was reset back to 0 and display on the menu. This was done by creating a push button interrupt that would change the state, reset the

total cash to 0, print the return screen, then set the state back to menu. A snippet of code that accomplished this can be seen below in figure 19.

Figure 19. Push button interrupt Return sequence

```
if (EXTI->PR & (1 << 8)) // Check if interrupt request is pending
{
    // Clear the pending request by writing a 1 to the corresponding bit
    EXTI->PR = (1 << 8);

   if(state == Menu){
     state = Return;
     Button_counter = 0;
}
```

The final portion of programming was the creating of the 'Admin' portion of the project. This portion of the project required secured access to an additional set of program options only accessible by the input of specific, six-digit PIN. This portion of the project was heavily assisted by drawing on the prior laboratory exercise five. Upon entry of the correct PIN, access to the administrative options would become available. Entered PIN characters were programmed to be hidden by only displaying the '*' character and an incorrect entry would not allow access to the administrative options. This was done by creating a input array that would fill with each key input. Then once the '#' key was pressed, the password and input password would be compared and would ether change the state or erase the password for the user to try again. A snippet of code on how this was accomplished can be seen below in figure 20.

Figure 20. Password comparer code snippet

```
if(*key != 12 && *key != 10){
    pass_inputted[count]= *key;    //set keypress into an arrar
    write_data ('*');              //print '*' to screen
    count++;                       //increment counter
  if(count>6){                     //if a key has been pressed
    write_command(1);              //screen screen
    Admin_screen();                //print pass word screen aga
    count=0;                       //set count to 0
  }
}
```

Once the correct PIN was entered, access would be given to view current temperature through on-board microcontroller temperature sensor, access the inventory screen and access the 'cash' screen. The temperature option displayed the current temp as read and calculated by the on-board microcontroller temperature sensor through the ADC by use of the reference voltage and number of bits. In order to display the temperature equation 1 was used to calculate the temperature in Celsius.

$$Temp\ in\ °C = \frac{[(V_{out}\ in\ V) - .76]}{.0025 + 25} \quad (1)$$

A snippet of code on how the was implemented can be seen below in figure 21.

Figure 21. Internal Temp Calculation

```
float Internal_Temperature_Sensor(void){
    float Vref=3.3;
    float result;
    float tempC;
    uint16_t code;

        ADC1->CR2 |= 1<<30;
        while(!(ADC1->SR & 2)) {}
        code = ADC1->DR &0x00000FFF;
        result = ((double)code / 4095 * Vref);
          tempC = ((result-.76) / .0025+25);

          return tempC;
        }
```

The ability to view and manipulate item inventory was also activated after gaining access to the administrative portion of the program. In this portion of the program, an administrator can replenish respective item stock to the full value. Additionally, an option to 'empty the till' is available. An administrator can 'withdraw' all funds to clear the cash que. At any time, the user can return to the main administrative screen by pressing the back (*) button.

At this time, specific pins were designated, and the circuit (Figure 23) was constructed. The initial circuit was constructed without integration of the machine body to ensure confidence in wiring and assembly. After the circuit was built, testing was performed. Initial testing showed that the circuit wiring was correct and interfaced correctly with the program. Fabrication of the machine body was started. The machine body was designed around component sizes, optimal wiring routings and time constraints. Solidworks was utilized for the modeling of individual assembly parts and the IdeaMaker slicing software was used for 3D printing of modeled parts. Parts were printed with use of a Raised 3D Pro printer. An illustrated part breakdown of the machine body is provided in Appendix B. The machine body design is patent pending. Detailed drawings and dimensions will not be provided in this report.

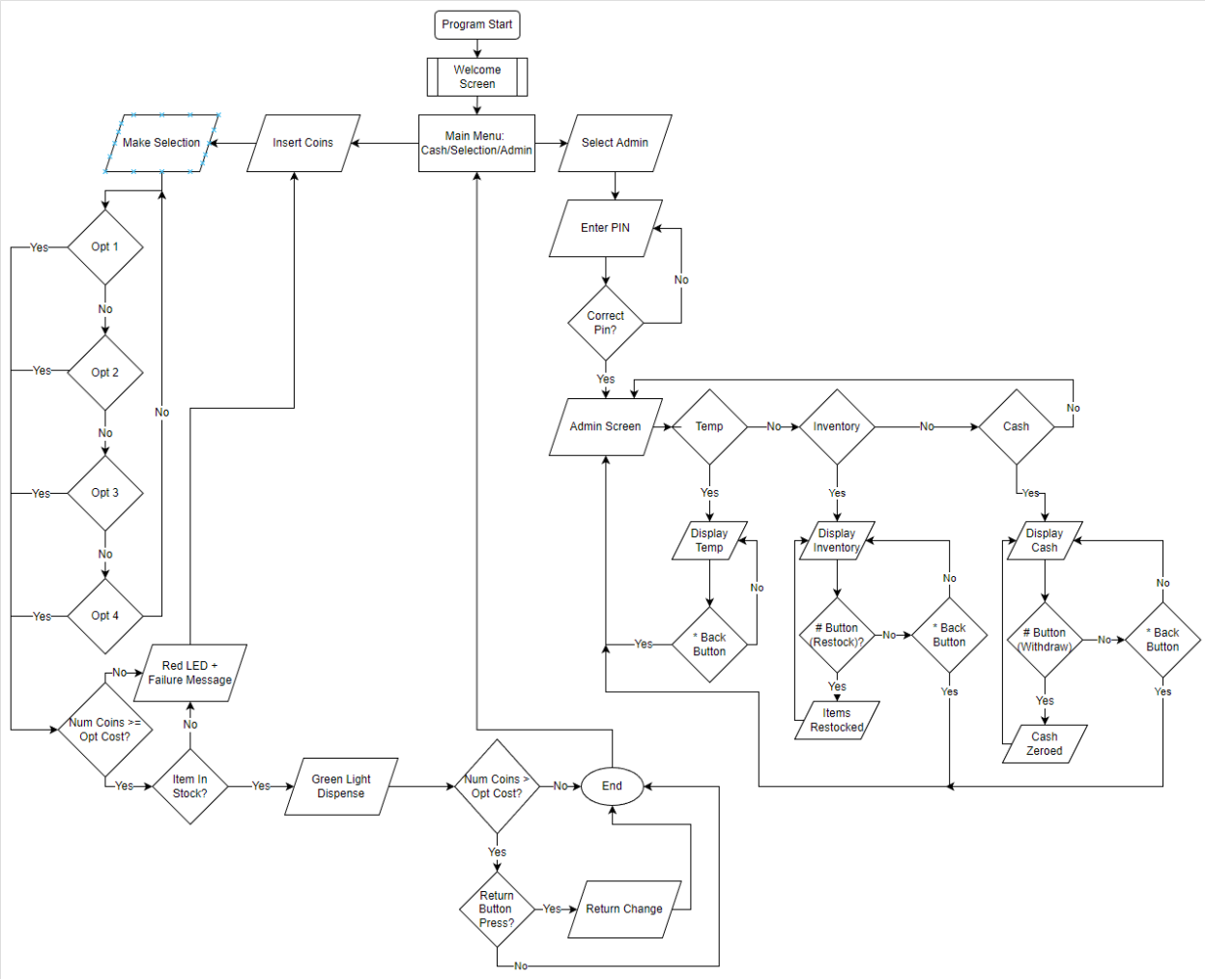


Figure 22: Flow Chart

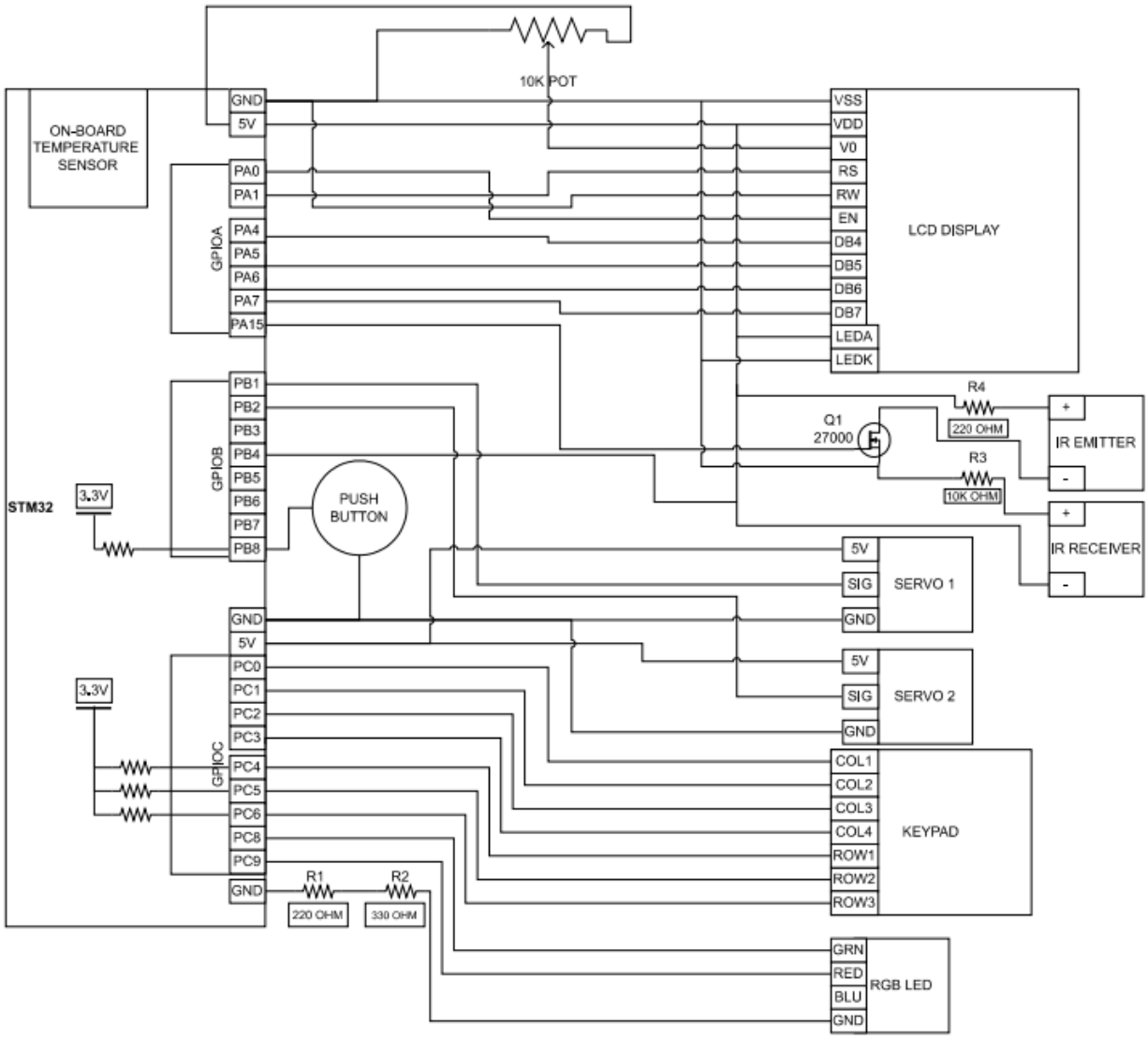


Figure 23: Circuit Diagram

Figure 13: Machine Body Exploded View (Patent Pending)

Conclusion

This project proved to be a challenging but rewarding experience. Overall programming was greatly assisted by use of prior laboratory exercises, lecture material and the textbook. As in the midterm project, breaking the project into manageable parts with thought given to next programming steps regarding choosing pins and timers was very beneficial.

Integration of the IR portion of the project proved to be most challenging aspect. With the number of sensing and counting options available in the field, it would be interesting to investigate and experiment with an alternative. For example, the photoelectric switch in Figure 13 has the capability to detect the absence or presence of an object even moving at high speeds. As the speed of the quarter moving through the path of the IR receiver and transmitter was one of the main causes of difficulty, a sensor such as the photoelectric switch below would address the problem. Of course, cost would need to be considered given a project of this scale.

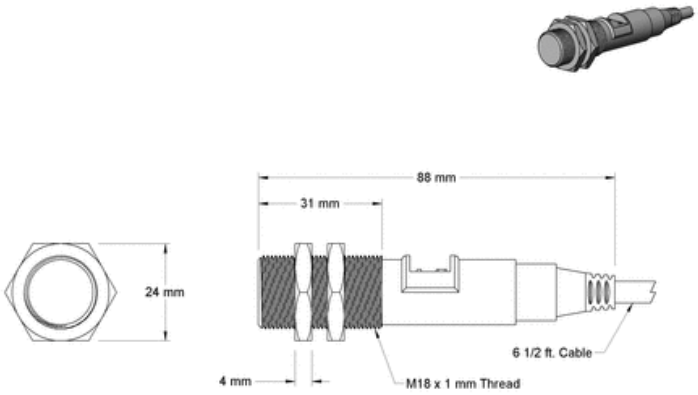


Figure 24: Photoelectric Switch (McMaster-Carr)

Regarding the integration of the electronics to the mechanical aspect of the project, it was unfortunate that the end result did not prove to be successful. The driving factor of this miss was time constraints and management. The idea to build a fully functional vending machine was a very ambitious goal. Because of this, and due to the plan to break the project into manageable parts, construction of a hypothetical machine was prioritized to ensure the minimum project requirements were met. Upon building the secondary system on the actual machine, it was unfortunately discovered that the secondary microcontroller had an internal fault and was not able to perform the programmed functions. Once again, due to time constraints and management, there was not ample time to rebuild and rewire the entire project prior to the due date. The foresight of prioritizing the hypothetical system proved to be the correct decision.

Overall, the project turned out well. The experience of integrating several systems to work together as a whole will be very beneficial for any future projects both academically and professionally.

Appendix A: Code

Figure A-1: Main.c for Final Project

```
/*****************************************************************
**********
 * Name:              Thomas Zoldowski
 * Course:            EGR 227 - Microcontroller Programming and
Applications Lab
 * Project:           Final Project - Vending Machine
 *
 * File:              Final Project.c
 * Description:       Final Project

*****************************************************************
**********/


#define MAX_STOCK 2                                    //Used to Change
the max stock of ALL items
#define PASSWORD "111111"                    //The admin Password


//Price of each Item
#define RED_PRICE .50
#define BLUE_PRICE .75
#define ORANGE_PRICE 1
#define GREEN_PRICE 1.50


#define SERVO_DISPENSE_TIME 10000     //The time it takes for the
servo to retract



#include "Timer.h"                //Timer Library
#include "stm32f4xx.h"
#include "LCD.h"                  //LCD Library
#include "keypad.h"               //Keypad Library
#include <stdio.h>
```

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



void Menu_State(void);     //Changes state to Menu


void Button_Interrupt_Init(void);          //Button interrupt
initialization
void LCD_init_pins (void);                      //LCD pins
initialization
void LCD_initialization(void);             //Initialization of the LCD
void Keypad_Init (void);                        //Keypad pins
initialization
void System_Interrupt_Initi(void);    //system interrupt
initialization
void Systick_Interrupt_Initi(void);   //SysTick timer interrupt
void RGB_Initi_Pins(void);                      //RGB initialization
pins
void TIM3_IRQHandler(void);
void EXTI0_IRQHandler(void);
void SysTick_Handler(void);
void EXTI1_IRQHandler(void);
void EXTI2_IRQHandler(void);
void EXTI3_IRQHandler(void);
void EXTI9_5_IRQHandler(void);


 volatile int flag;


 uint32_t last_time;




float frequency;                                    //IR frequency
```

```
float Last_f;                                                    //last
frequency


uint32_t last_timer3_update_time;

uint32_t timer3_stopped_updating_flag;

 char cash[4];

 volatile float totalCash;            //total cash inserted

 int pressed;
     //keypad press flag

 volatile uint32_t counter;           //systick inturrupt counter


 float cashMade;                                      //Total Cash
made from selling product

 int Button_counter;                                //Systick counter for
Button press

 int flag1;

 float temp;
     //temperature

 char strtemp[4];
     //temperature as a string

 char password[6];                                    //admin password

 char pass_inputted[6];                             //inputted password


 char strTotalCash[5];                              //total cash string


 uint8_t redGum;

 uint8_t blueGum;

 uint8_t   orangeGum;

 uint8_t   greenGum;

 int count;


enum states {

           Welcome,

           Menu,
```

```
            ItemChoosen,
            Return,
            Password,
            Admin,
            Temperature,
            Inventory,
            Cash,
      };
       enum states state;

 char num[2];

int main (void) {
      __disable_irq();
  System_Interrupt_Initi();
      Systick_Interrupt_Initi();

      //--------------LCD Functions(LCD.h)------------------
      LCD_init_pins();                 // LCD pin initialization
  LCD_initialization(); // LCD initialize

      //-------------Keypad Functions(Keypad.h)------------
      Keypad_Init();                   // Keypad initalize

      //---------Timer & ADC Functions(Timer.h)--------------
      TIM2_Initi();                          //Timer 2 initialzation
      TIM2_Count();                          //Timer 2 setting
controller
      TIM3_Initi();                          //Timer 3 initialization
  TIM3_Control();                      //Timer 3 Settings Controller
      TIM10_Initi();                   //Timer 10 Initializaiton
      TIM10_Count();                   //Timer 10 settings controller
```

```
    TIM11_Initi();                  //Timer 11 Initialization
    TIM11_Count();                  //Timer 11 settings controller
    ADC_Init();                             //ADC initialization


    Button_Interrupt_Init();   //push Button inturrupt
initialization
    RGB_Initi_Pins();                   //RGB led pin
initialization


    __enable_irq();


    //Initialize Variables
    last_time = 0;                                          //last
time Timer 3 inturrupt was triggered
    flag=0;

    last_timer3_update_time = 0;
    timer3_stopped_updating_flag=0;
    totalCash=0.00;                                         //Total
Cash Inserted
    counter = 0;
    count = 0;
    password[6] = PASSWORD;                     //Password for Admin
screen
    redGum = MAX_STOCK;                             //Set Max stock
for each item
    blueGum = MAX_STOCK;
    orangeGum = MAX_STOCK;
    greenGum = MAX_STOCK;
    state = Welcome;


    welcome_screen();               //print Welcome screen
```

```
      ////////////////////////////MAIN
LOOP////////////////////////////////////
      while(1){
      switch(state){
            //---------------------Welcome Screen---------------------
            case Welcome:



                  if (counter >=30000){      //After 3 seconds change to
Menu state and print new screen
                        write_command(1);
                        MainMenu_screen();
                        state = Menu;


                  }
                  break;
            //----------------------Menu Screen-----------------------
            case Menu:



                              if(pressed && ((*key == 1)||(*key ==
2)||(*key == 3)||(*key == 4)||(*key == 10))){    //If 1, 2, 3, 4, or
'*'
                                          flag1 = 1;
                                          sprintf(num, "%d", *key);
      //convert key press to character
                                          write_data (num[0]);
      //print keypress
                                          write_command(0x10);
      //shift cursor left


                                          state = ItemChoosen;
      //change state to ItemChooses
                                          pressed = 0;
```

```
                                    last_time=counter;
                                    break;
                              }
                              else if(timer3_stopped_updating_flag){
                                    totalCash+=0.25;
                        //increment totalCash by .25
                                    sprintf(strTotalCash, "%.2f",
totalCash);          //convert floating point to character
                                    write_command(Line_1+0x8);
                                    //print to LCD screen
                                    for(int i = 0; i<4; i++)
                                    write_data(strTotalCash[i]);



     while(timer3_stopped_updating_flag){};     //stop while
obsticule in blocking IR sensor
                                    write_command(Line_2+0xB);
                              //send cursor back to original postition
                                    }
           break;


           //-----------Item has been Choosen--------------------
                 case ItemChoosen:
                       flag1=0;



                            switch(*key){              //Swtich
function to perform differnt stuff for each number press
                                    case 10:        //if '*' is pressed
                                    state = Password;          //change
state to Password state
                                    write_command(1);          //clear
screen
                                    Admin_screen();            //printf
Admin password screen
                                    pressed = 0;
```

```
                                    break;


                                    case 1://Key 1

                                          if(totalCash>=RED_PRICE &&
redGum>0){      //check if enough money has been inserted

                                          pressed = 1;

                                          cashMade += RED_PRICE;
      //Add cash to cash made

                                          redGum -= 1;
                        //decrement stock variable

                                          Dispensed(totalCash-
RED_PRICE, *key);          //print dispense screen

                                          totalCash=0;
                        //set totalcash back to 0

                                          counter = 0;
                        //system clock counter to 0

                                          GPIOC->ODR |= (1<<9);
      //turn on green LED

                                          ServoMotor(*key);
                  //Move servo motor


                                    }else if(redGum==0 && !pressed){
      //check if theres is enough stop

                                          OutOfStock();
                                          //print out of stock screen

                                          pressed = 1;

                                          counter = 0;

                                          GPIOC->ODR |= (1<<8);
                              //turn on red LED

                                          while(counter <=300000){}
                        //wait 3 seconds


                                    }else
if((totalCash<=RED_PRICE)&&!pressed){       //Check if enough cash has
been inserted

                                          pressed = 1;

                                          NotEnough(RED_PRICE);
      //printf the no enough cash screen
```

```
                                        GPIOC->ODR |= (1<<8);
     //Turn on Red led
                                        counter = 0;
                    //set counter to 0
                                        while(counter <=30000){}
     //wait 3 seconds
                                    }


                                        if(counter >=30000){
     //wait until system clock had reach 3 seconds
                                            pressed = 0;
                                            GPIOC->ODR &=
~((1<<9)|(1<<8));     //turn off all LEDs
                                            state = Menu;
                                        //Change state to Menu
                                            Menu_State();
                                        //Print_Keys Menu screen
                                        }else
if(counter>=SERVO_DISPENSE_TIME && counter<=SERVO_DISPENSE_TIME+1000)
                                            TIM10->CCR1 = 2000;
                                //move servo back to 45 degree angle
                                    break;


                                case 2://Key 2
                                    if(totalCash>=BLUE_PRICE &&
blueGum>0){           //check if enough money has been inserted
                                        pressed = 1;
                                        cashMade += BLUE_PRICE;
     //Add cash to cash made
                                        blueGum -= 1;
                    //decrement stock variable
                                        Dispensed(totalCash-
BLUE_PRICE, *key);         //print dispense screen
                                        totalCash=0;
                    //set totalcash back to 0
                                        counter = 0;
                    //system clock counter to 0
```

```
                                        GPIOC->ODR |= (1<<9);
      //turn on green LED
                                        ServoMotor(*key);
                  //Move servo motor
                                    }else if(blueGum==0 &&
!pressed){      //check if theres is enough stop
                                        OutOfStock();
                                        //print out of stock screen
                                        pressed = 1;
                                        counter = 0;
                                        GPIOC->ODR |= (1<<8);
                              //turn on red LED
                                        while(counter <=30000){}
                        //wait 3 seconds
                                    }else
if((totalCash<=BLUE_PRICE)&&!pressed){          //Check if enough
cash has been inserted
                                        pressed = 1;
                                        NotEnough(BLUE_PRICE);
      //print the no enough cash screen
                                        GPIOC->ODR |= (1<<8);
      //Turn on Red led
                                        counter = 0;
                        //set counter to 0
                                        while(counter <=30000){}
      //wait 3 seconds
                                    }
                                        if(counter >=30000){
      //wait until system clock had reach 3 seconds
                                            pressed = 0;
                                            GPIOC->ODR &=
~((1<<9)|(1<<8));     //turn off all LEDs
                                            state = Menu;
                                        //Change state to Menu
                                            Menu_State();
                                        //Print_Keys Menu screen
                                        }else
if(counter>=SERVO_DISPENSE_TIME && counter<=SERVO_DISPENSE_TIME+1000)
```

```
                                        TIM10->CCR1 = 2000;
                                  //move servo back to 45 degree angle
                                        break;


                                  case 3://Key 3
                                        if(totalCash>=ORANGE_PRICE &&
orangeGum>0){         //check if enough money has been inserted
                                        pressed = 1;
                                        cashMade += ORANGE_PRICE;
      //Add cash to cash made
                                        orangeGum -= 1;
                        //decrement stock variable
                                        Dispensed(totalCash-
ORANGE_PRICE, *key);       //print dispense screen
                                        totalCash=0;
                        //set totalcash back to 0
                                        counter = 0;
                        //system clock counter to 0
                                        GPIOC->ODR |= (1<<9);
      //turn on green LED
                                        ServoMotor(*key);
                  //Move servo motor
                                  }else if(orangeGum==0 &&
!pressed){      //check if theres is enough stop
                                        OutOfStock();
                        //print out of stock screen
                                        pressed = 1;


                                        counter = 0;
                        //set counter to 0
                                        GPIOC->ODR |= (1<<8);
      //Turn on Red led
                                        while(counter <=300000){}
      //wait 3 seconds
                                  }else
if((totalCash<=ORANGE_PRICE)&&!pressed){         //Check if enough
cash has been inserted
                                        pressed = 1;
```

```
                                        NotEnough(ORANGE_PRICE);
          //print the no enough cash screen
                                        GPIOC->ODR |= (1<<8);
               //Turn on Red led
                                        counter = 0;
                                  //set counter to 0
                                        while(counter <=30000){}
          //wait 3 seconds
                                     }
                                        if(counter >=30000){
     //wait until system clock had reach 3 seconds
                                             pressed = 0;
                                             GPIOC->ODR &=
~((1<<9)|(1<<8));     //turn off all LEDs
                                             state = Menu;
                                             Menu_State();
                                        //Print_Keys Menu screen
                                        }else
if(counter>=SERVO_DISPENSE_TIME && counter<=SERVO_DISPENSE_TIME+1000)
                                             TIM11->CCR1 = 2000;
                                  //move servo back to 45 degree angle
                                     break;


                                  case 4://Key 4
                                     if(totalCash>=GREEN_PRICE &&
greenGum>0){          //check if enough money has been inserted
                                        pressed = 1;
                                        cashMade += GREEN_PRICE;
     //Add cash to cash made
                                        greenGum -= 1;
                        //decrement stock variable
                                        Dispensed(totalCash-
GREEN_PRICE, *key);        //print dispense screen
                                        totalCash=0;
                        //set totalcash back to 0
                                        counter = 0;
                        //system clock counter to 0
```

```
                                        GPIOC->ODR |= (1<<9);
      //turn on green LED
                                        ServoMotor(*key);
                  //Move servo motor
                                    }else if(greenGum==0 &&
!pressed){      //check if theres is enough stop
                                        OutOfStock();
                        //print out of stock screen
                                        pressed = 1;


                                        counter = 0;
                        //set counter to 0
                                        GPIOC->ODR |= (1<<8);
      //Turn on Red led
                                        while(counter <=30000){}
      //wait 3 seconds
                                    }else
if((totalCash<=GREEN_PRICE)&&!pressed){          //Check if enough
cash has been inserted
                                        pressed = 1;
                                        NotEnough(GREEN_PRICE);
            //printf the no enough cash screen
                                        GPIOC->ODR |= (1<<8);
            //Turn on Red led
                                        counter = 0;
                              //set counter to 0
                                        while(counter <=30000){}
                                    }
                                        if(counter >=30000){
      //wait until system clock had reach 3 seconds
                                             pressed = 0;
                                             GPIOC->ODR &=
~((1<<9)|(1<<8));     //turn off all LEDs
                                             state = Menu;
                                             Menu_State();
                                        //Print_Keys Menu screen
                                        }else
if(counter>=SERVO_DISPENSE_TIME && counter<=SERVO_DISPENSE_TIME+1000)
```

```
                                        TIM11->CCR1 = 2000;
                                  //move servo back to 45 degree angle
                                  break;
                                  default:
                                        state = Menu;
                                  break;


                        }
            break;


                                  case Return:                //Return
state if return button is pressed


                                        Returned(totalCash);
      //print return screen
                                        if(Button_counter >= 50000){//if
3 seconds has pass (extra time has been added to account for delays in
other function)
                                              totalCash=0;    //set
total cash to 0
                                              Menu_State();   //change
state to menu and print menu screen
                                        }
                                  break;
            //======================ADMIN SCREENS
CODE=========================


            //------------------Admin Password Screen------------------
-
            case Password:



                  if(pressed){


                        if(*key != 12 && *key != 10){
```

```
                                    pass_inputted[count]= *key;      //set
keypress into an arrar (*FIX ME*)

                                    write_data ('*');
                    //print '*' to screen

                                    count++;
                                //increment counter

                                    if(count>6){
                               //if a key has been pressed 6 times

                                    write_command(1);
                    //screen screen

                                    Admin_screen();
                    //print pass word screen again

                                    count=0;
                                //set count to 0

                                    }

                               }

                               else if(*key == 12){
          //if '#' has been pressed

                                    if(strcmp(pass_inputted, password)){
          //compare correct password to the inputted password

                                         pass_inputted[0]=0;
                                    //clearerr inputted password variable

                                         state = Admin;
          //chang state to admin state

                                         write_command(1);
          //clear screen

                                         settings_screen();    //print
admin screen

                                         pressed = 0;

                                    }else{

                                         pass_inputted[0]=0;   //if '*'
is pressed go back to Menu screen

                                         write_command(1);

                                         MainMenu_screen();

                                         state = Menu;

                                         count=0;

                                    }
```

```
                        }
                        else{ //if '*' is pressed go back to Menu screen
                              pass_inputted[0]=0;
                              Menu_State();
                        }
                         pressed = 0;
                  }

            break;
            //------------------Admin Screen--------------------
            case Admin:

                  if(pressed){    //if keypad has been pressed

                        switch(*key){
                              case 1:
                                    state = Temperature;       //change
state to temperature
                                    write_command(1);
      //clear screen
                                    Temperature_screen();      //print
temperature screen
                                    pressed = 0;
                                    counter=4000;
      //set counter to 4000 (determines how quickly temp prints to
screen)
                                    break;
                              case 2:
                                    state = Inventory;          //change
state to inventory
                                    write_command(1);
                                    Inventory_screen();        //print
inventory screen
```

```
     Stock(redGum,blueGum,orangeGum,greenGum);  //send stock values
to be printed to the LCD
                              pressed = 0;
                                   break;
                              case 3:
                                   state = Cash;
     //change state to Cash
                                   write_command(1);
                                   Cash_screen();
     //print Cash screen
                                   write_command(Line_2+0x1); //
                                   sprintf(strTotalCash, "%f",
cashMade);
                                   for(int i = 0; i<4; i++)
                                   write_data(strTotalCash[i]);
                                   pressed = 0;


                                   break;
                              case 10:
                                   state = Menu;   //change state to
menu
                                   write_command(1);
                                   MainMenu_screen();    //print menu
screen
                                   pressed = 0;
                                   write_command(Line_1+0x8);
                                   for(int i = 0; i<4; i++)
                                   write_data(strTotalCash[i]);
                                   write_command(Line_2+0xB);
                                   break;
                         }
                    }


           break;
```

```
            //--------------Temperature Screen-------------------------
            case Temperature:

            if(counter==5000){
                        temp = Internal_Temperature_Sensor();      //get
temperature

                        sprintf(strtemp, "%f", temp);         //convert
temp int to char

                              //print temperature
                              for(int i=0;i<4; i++)
                                    write_data(strtemp[i]);
                                    write_command(Line_2+0x4);
                                    counter=0;
            }
                        if((*key == 10)&&(pressed)){     //If "*" is
pressed send to Admin state
                                    state = Admin;
                                    write_command(1);
                                    settings_screen();
                                    pressed = 0;
                        }


            break;
            //----------------Inventory Screen----------------------
            case Inventory:

                  if((*key==12)&&(pressed)){ //if '#' restock all items
to max stock
                        redGum = MAX_STOCK;
                        blueGum = MAX_STOCK;
```

```
                    orangeGum = MAX_STOCK;
                    greenGum = MAX_STOCK;
                    Stock(redGum,blueGum,orangeGum,greenGum);
     //print new stock value to LCD
                    pressed = 0;
               }
               else if((*key==10)&&(pressed)){       //if '*' go back
to menu screen
                              state = Admin;
                              write_command(1);
                              settings_screen();
                              pressed = 0;
               }
          break;
          //---------------Cash-------------------------
          case Cash:

               if(pressed && *key == 12){ //if '#' is pressed set
cash made to 0
                    cashMade = 0;
                    write_command(Line_2+0x1);
                    sprintf(strTotalCash, "%f", cashMade);
                    for(int i = 0; i<4; i++)
                    write_data(strTotalCash[i]);
                    pressed = 0;
               }
               if(pressed && *key == 10){ //if '*' is pressed go back
to main menu
                              state = Admin;
                              write_command(1);
                              settings_screen();
                              pressed = 0;
               }
```

```
                break;


                //====================== END
============================
     }
     }
}
////////////////////////////////END OF MAIN
LOOP//////////////////////////////////////



/***| Menu_State() |*********************************//*
*Brief: Go to Menu state function
*Params:
*           None
*Returns:
*           None
*********************************************************/
void Menu_State(void){
     state = Menu;
     write_command(1);
     MainMenu_screen();
     pressed = 0;
     write_command(Line_1+0x8);
     sprintf(strTotalCash, "%.2lf", totalCash);
     for(int i = 0; i<4; i++)
     write_data(strTotalCash[i]);
     write_command(Line_2+0xB);
}


/***| Button_Interrupt_Init() |*********************************//*
*Brief:         Push Button Inturrupt pins initialization
*Pin Config: PA8
```

```
*Params:
*           None
*Returns:
*           None
**********************************************************/
void Button_Interrupt_Init(void)
{
    //RCC->AHB1ENR |= 1;          // Enable clock for GPIOA
    GPIOA->MODER &= ~(3 << 16); // Set PA8 to input mode
    GPIOA->PUPDR &= ~(3 << 16); // Set PA8 to pull-up
    GPIOA->PUPDR |= (1 << 16);
           SYSCFG->EXTICR[2] &= ~0x0F; // Set EXTI8 to use GPIOA
    //SYSCFG->EXTICR[2] |= 0x1000;
           EXTI->IMR |= (1 << 8);       // Unmask interrupt request
    EXTI->FTSR |= (1 << 8);      // Set interrupt on falling edge
    NVIC_EnableIRQ(EXTI9_5_IRQn); // Enable EXTI9_5 interrupt in NVIC
}

/***| RGB_Initi_Pins() |*********************************//*
*Brief:          RGB chip pins initialization
*Pin Config: PC8、PC9
*Params:
*           None
*Returns:
*           None
**********************************************************/
void RGB_Initi_Pins(void){
      GPIOC->MODER &= ~(3<<18)|(3<<16);
      GPIOC->MODER |= (1<<18)|(1<<16);
}

/***| TIM3_IRQHandler() |*********************************//*
```

```
*Brief:          Inturrupt handler for timer 3
*Params:
*           None
*Returns:
*           None
*****************************************************/
void TIM3_IRQHandler(void){
     uint32_t current;
     uint32_t period;

     int last;
          if (TIM3->SR & 0x2) {



               last_time = counter;

               current = TIM3->CCR1;
               period = current - last;
               last = current;
               // Compute the frequency from the period
               frequency = ((2000.0f / period) / 2.0f);

          TIM3->SR =0;
          }
}

/***| System_Interrupt_Initi() |*********************************//*
*Brief:          System Interrupt initialization pins
*Params:
*           None
*Returns:
*           None
```

```
**********************************************************/
void System_Interrupt_Initi(void){
      RCC->APB2ENR |=0x4000;

      //keypad inturrupts
      SYSCFG->EXTICR[0] &= ~(15 << 0) | (15 << 4) | (15 << 8) | (15 <<
12);
      SYSCFG->EXTICR[0] |= (2 << 0) | (2 << 4) | (2 << 8) | (2 << 12);

      // Enable external interrupts for the rows
      EXTI->IMR |= (1 << 0) | (1 << 1) | (1 << 2) | (1 << 3);
      EXTI->FTSR |= (1 << 0) | (1 << 1) | (1 << 2) | (1 << 3);

      //NVIC_EnableIRQ(EXTI9_5_IRQn);
      NVIC_EnableIRQ(EXTI0_IRQn);
      NVIC_EnableIRQ(EXTI1_IRQn);
      NVIC_EnableIRQ(EXTI2_IRQn);
      NVIC_EnableIRQ(EXTI3_IRQn);

}

/***| SysTick_Handler() |**********************************//*
*Brief:          Systick Interrupt Handler
*Params:
*           None
*Returns:
*           None
**********************************************************/
void SysTick_Handler(void){
      counter++;
      Button_counter++;
      if(counter - last_time > 2000)
```

```
            timer3_stopped_updating_flag=1;
      else
            timer3_stopped_updating_flag=0;

}

/***| Systick_Interrupt_Initi() |*********************************//*
*Brief:          Systick Interrupt Initialization
*Params:
*           None
*Returns:
*           None
*********************************************************/
void Systick_Interrupt_Initi(void){

      SysTick->LOAD = 0x00FFFFFF -1;
      SysTick->VAL = 0;
      SysTick->CTRL = 7;
}

/***| EXTI0_IRQHandler() |*********************************//*
*Brief:          EXTI0 Interrupt Handler
*Params:
*           None
*Returns:
*           None
*********************************************************/
void EXTI0_IRQHandler(void) {
  pressed = 1;
      keypad_isr(0);
}
```

```
/***| EXTI1_IRQHandler() |*********************************//*
*Brief:          EXTI1 Interrupt Handler
*Params:
*           None
*Returns:
*           None
*********************************************************/
void EXTI1_IRQHandler(void) {
     pressed = 1;
  keypad_isr(1);
}

/***| EXTI2_IRQHandler() |*********************************//*
*Brief:          EXTI2 Interrupt Handler
*Params:
*           None
*Returns:
*           None
*********************************************************/
void EXTI2_IRQHandler(void) {
     pressed = 1;
  keypad_isr(2);
}

/***| EXTI3_IRQHandler() |*********************************//*
*Brief:          EXTI3 Interrupt Handler
*Params:
*           None
*Returns:
*           None
*********************************************************/
void EXTI3_IRQHandler(void) {
```

```
      pressed = 1;
  keypad_isr(3);
}


/***| EXTI9_5_IRQHandler() |**********************************//*
*Brief:          EXTI9_5 Interrupt Handler
*Params:
*           None
*Returns:
*           None
*************************************************************/
void EXTI9_5_IRQHandler(void)//Push Button on PB6
{
    if (EXTI->PR & (1 << 8)) // Check if interrupt request is pending
for line 6
    {
        // Clear the pending request by writing a 1 to the
corresponding bit
        EXTI->PR = (1 << 8);


       if(state == Menu){
                        state = Return;
                        Button_counter = 0;
    }
}
}
```

Figure A-2: Keypad.c for Final Project

```
/*****************************************************************
**********
* Name:
* Course:           EGR 226 - Microcontroller Programming and
Applications
* Project:          Final Project
* File:             keypad.c
* Description:      Includes functions to initialize and sense the
keypad, print
*                   the pressed key
* Pin Configs:      PC0-PC3=> Rows and PC4-PC6=> Cols
*                   Keypad pin7-PC0
*                   Keypad pin1-PC6
*****************************************************************
**********/


/***| Standard Library Includes |***/
#include "stm32f4xx.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "keypad.h"


/***| Systick_Init() |*********************************//*
*Brief: Systick Initilization Function
*Params:
*           None
*Returns:
*           None
*********************************************************/
void SysTick_Init(void)                                  // -----
Initialize SysTick ------
{
```

```
    SysTick -> CTRL = 0;                                    // disable
SysTick during setup
    SysTick -> LOAD = 0x00FFFFFF;                           // maximum
reload value
    SysTick -> VAL  = 0;                                    // any write
to current clears it
    SysTick -> CTRL = 0x00000005;                           // enable
SysTick, 16MHz, no interrupts
}


/***| Systick_Delay() |*********************************//*
*Brief: Systick Millisecond Delay Function, gets 16MHz clock
*Brief: 1/16 e-6s is one period. Max load is 2^24
*Params:
*           (uint16) msdelay: delay in the units of milliseconds
*Returns:
*           None
*************************************************************/
void SysTick_ms_Delay(uint16_t delay)                       // ----
Configurable Systick delay ---
{
    SysTick -> LOAD = ((delay*16000) - 1);                 // 1ms count
down to zero
    SysTick -> VAL = 0;                                     // any write
to CVR clears it and COUNTFLAG in CSR
    while((SysTick -> CTRL & 0x00010000) == 0);             // Wait for
flag to be SET (Timeout happened)
}



/***| Keypad_Init() |***********************************//*
*Brief: Standard Port Initializations, select the responsible
*Brief: pins and assign them as GPIO input with rows pull up enabled
*Brief: Since the reading will be done over the rows, rows are set up
*Brief: as pull up resistors
```

```
*Params:
*           None
*Returns:
*           None
*********************************************************/
void Keypad_Init (void)
{
    RCC->AHB1ENR |= 4;
      // Set the mode of the rows and columns to input
    GPIOC->MODER &= ~((3 << (0 * 2)) | (3 << (1 * 2)) | (3 << (2 * 2))
| (3 << (3 * 2)) | (3 << (4 * 2)) | (3 << (5 * 2)) | (3 << (6 * 2)));
            // Set the rows and columns to use the internal pull-up
resistors
    GPIOC->PUPDR |= ((1 << (0 * 2)) | (1 << (1 * 2)) | (1 << (2 * 2))
| (1 << (3 * 2)));

    // Configure the columns as outputs with push-pull mode
    GPIOC->MODER |= ((1 << (4 * 2)) | (1 << (5 * 2)) | (1 << (6 *
2)));
    GPIOC->OTYPER &= ~((1 << 4) | (1 << 5) | (1 << 6));
    GPIOC->OSPEEDR |= ((2 << (4 * 2)) | (2 << (5 * 2)) | (2 << (6 *
2)));
}

/***| Print_Keys() |*************************************//*
*Brief: Prints the pressed button, if *,0 or # is pressed
*Brief: reformulates them as *numptr is another value in the
*Brief: rowxcol array => 4x3
*Params:
*           (uint16) *numptr: pressed number pointer
*Returns:
*           None
*********************************************************/
void Print_Keys(uint16_t *numptr)
```

```
{
    printf("YOU PRESSED:  ");
    if (*numptr == 12) printf("#\n");
    if (*numptr == 10) printf("*\n");
    if (*numptr <  10) printf("%d\n",*numptr);
           if (*numptr == 11) {
                 *numptr = 0;
                 printf("0\n");
           }

}

/***| Read_Keypad() |*********************************//*
*Brief: Checks for the button press by grounding each col as
*Brief: assigning each col as GND one by one while the rest
*Brief: are floating input with infinite impedance. As a result,
*Brief: if any press is hit, that row will be grounded over the
*Brief: col that is assigned as GND.
*|--|--| -> Row0=> GND if 1 is pressed
*|--|--| -> Row1=> GND if 4 is pressed
*|--|--| -> Row2=> GND if 7 is pressed
*|--|--| -> Row3=> GND if * is pressed
*|=>GND
*Brief: Caution: If two cols are hit at the same time and they
*Brief: are assigned as both OUT, there will be a short circuit
*Brief: that can fry MCU. Hence, multiple repeating IN settings
*Brief: are placed for safety.
*Params:
*           (uint16) *numptr: pressed number pointer
*Returns:
*           None
*****************************************************/
```

```
// Define a global variable that is a pointer to a character
int value;
int* key = &value;


// Define a two-dimensional array of characters that stores the keys
and their corresponding row and column values
const int nums[4][3] = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};



void keypad_isr(int row) {
uint8_t col,rows;                                    // will need to
loop for rows and cols
    for(col=0; col<3; col++)
    {
        GPIOC->MODER &=~(uint32_t)((3<<(2*Hex2Bit
(R0)))|(3<<(2*Hex2Bit (R1)))|(3<<(2*Hex2Bit (R2)))|(3<<(2*Hex2Bit
(R3)))|(3<<(2*Hex2Bit (C0)))|(3<<(2*Hex2Bit (C1)))|(3<<(2*Hex2Bit
(C2))));        // Set Columns to inputs for safety not to burn MCU
        GPIOC->MODER |= (uint32_t)(1<<2*(col+4));                //
Set Column i to Output
                      GPIOC->ODR   &=~(uint32_t)(1<<(col+4));
                           // Set Column i to GND
                      SysTick_ms_Delay(10);
// Delay the while loop
        rows =  GPIOC->IDR & (R0|R1|R2|R3);                    // read
all rows: Read must be done before the debouncing
                    while( !(GPIOC->IDR & R0)| !(GPIOC->IDR & R1) |
!(GPIOC->IDR & R2) | !(GPIOC->IDR & R3));
        // debouncing loop: if any of the buttons are pressed just put
it into the loop
```

```
        if (rows != 0xF) break;                          // if one of
the input is GND, some key is pressed.
    }
           GPIOC->MODER |= ((1 << (4 * 2)) | (1 << (5 * 2)) | (1 << (6
* 2)));
   // Use the keys array to determine which key was pressed and store
it in the key variable
  value = nums[row][col];
     if(value == 11) value = 0;

  // Clear the interrupt pending bit
  EXTI->PR |= (1 << row);
           Systick_ms_delay(6);
}
//------------------------------------------------------------------
```

Figure A-3: LCD.c for Final Project

```
/*
 * LCD.c
 *
 *  Created on: Oct 15, 2021
 *      Author: Cakmak
 */


/**********************************************************************
**********
* Name:
* Course:           EGR 226 - Microcontroller Programming and
Applications
* Project:          Lab 06 - LCD Introduction
* File:             LCD.c
* Description:      Set of functions to initialize LCD (Hitachi
HD44780)
```

```
*
* Pin Configs:      LEDA, VDD=>5V, VSS, RW (always write), LEDK=>GND,
VO=>POT
*                   RS=>PA1 (Command/Data), E=>PA0 (Enable)
*                   DB0-DB3=>Disconnected
*                   DB4-DB7=>PA4-PA7 (4-bit transfer)
**********************************************************************
**********/


/***| Standard Library Includes |***/
#include "stm32f4xx.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "LCD.h"


/***| Systick_init() |*********************************//*
*Brief: Systick Initilization Function
*Params:
*           None
*Returns:
*           None
*********************************************************/
void Systick_init ()
{
    SysTick -> CTRL    = 0;                              // disable
SysTick during step
    SysTick -> LOAD    = 0x00FFFFFF;                     // max reload
value
    SysTick -> VAL     = 0;                              // any write
to current clears it
    SysTick -> CTRL    = 0x00000005;                     // enable
SysTick, 3MHz, No Interrupts
}
```

```
/***| Systick_msdelay() |*****************************//*
*Brief: Systick Millisecond Delay Function, gets 16MHz clock
*Brief: 62.5ns is one period. 1 us=(62.5*16)ns, Max load is 2^24
*Params:
*           (uint16) msdelay: delay in the units of milliseconds
*Returns:
*           None
*****************************************************/
void Systick_us_delay (uint16_t usdelay)
{
    SysTick -> LOAD = ((usdelay*16)-1);                  // delay for
1 us delay value
    SysTick -> VAL  = 0;                                 // any write
to CVR clears it
    while ( (SysTick -> CTRL  &  0x00010000) == 0);      // wait for
flag to be SET (16th bit)
}


/***| Systick_msdelay() |*****************************//*
*Brief: Systick Millisecond Delay Function, gets 16MHz clock
*Brief: 62.5ns is one period. 1 ms=(62.5*16000)ns, Max load is 2^24
*Params:
*           (uint16) msdelay: delay in the units of milliseconds
*Returns:
*           None
*****************************************************/
void Systick_ms_delay (uint16_t msdelay)
{
    SysTick -> LOAD = ((msdelay*16000)-1);               // delay for
1 ms delay value
    SysTick -> VAL  = 0;                                 // any write
to CVR clears it
```

```
    while ( (SysTick -> CTRL  &  0x00010000) == 0);     // wait for
flag to be SET (16th bit)
}


/***| Hex2Bit() |*****************************//*
*Brief: Is used for 2y:2y+1 type definitions in registers
*Brief: where yth bit is given as the input
*Params:
*           (uint32) hex_num: yth bit is given
*Returns:
*           eg. 01000000 => returns counter bit6, y=6
*********************************************************/
uint8_t Hex2Bit (uint32_t hex_num)
{
           uint8_t bit_count=0;
           while(hex_num>>1)
           {
                bit_count++;
                hex_num=hex_num>>1;
           }
           return bit_count;
}


/***| LCD_init_pins() |*********************************//*
*Brief: Initialize the MCU Port (PA) responsible for LCD control
*Params:
*           None
*Returns:
*           None
*********************************************************/
void LCD_init_pins ()
{
```

```
    RCC->AHB1ENR  |= A;

            GPIOA->MODER  &=~(uint32_t) ((3<<(2*Hex2Bit
(RS)))|(3<<(2*Hex2Bit (E)))|(3<<(2*Hex2Bit (DB4)))|(3<<(2*Hex2Bit
(DB5)))|(3<<(2*Hex2Bit (DB6)))|(3<<(2*Hex2Bit (DB7))));  // 2y+1:2y
bits reset

            GPIOA->MODER  |= (uint32_t) ((1<<(2*Hex2Bit
(RS)))|(1<<(2*Hex2Bit (E)))|(1<<(2*Hex2Bit (DB4)))|(1<<(2*Hex2Bit
(DB5)))|(1<<(2*Hex2Bit (DB6)))|(1<<(2*Hex2Bit (DB7))));  // set as
output

            GPIOA->OTYPER &=~(uint32_t) (RS|E|DB4|DB5|DB6|DB7);

            GPIOA->OSPEEDR&=~(uint32_t) ((3<<(2*Hex2Bit
(RS)))|(3<<(2*Hex2Bit (E)))|(3<<(2*Hex2Bit (DB4)))|(3<<(2*Hex2Bit
(DB5)))|(3<<(2*Hex2Bit (DB6)))|(3<<(2*Hex2Bit (DB7))));  // 2y+1:2y
bits reset

}


/***| pulseEnablePin() |*********************************//*

*Brief: Setup Enable Signal which needs to be toggling from

*Brief: 0 to 1 and then to 0 for data/command transfer

*Brief: Pulse width should be min. 230ns, satisfied with 10us pulse
width

*Brief: Data is already present in PA4-7, min. data setup 80ns,
satisfied with initial 10us+10us pulse width

*Brief: Data hold time is min. 10ns, satisfied with 10us final delay

*Brief: RW is always GND, no problem with setup time with RW signal
setup

*Brief: RS always is set up first, initial 10us delay satisfies setup
time for RS (min. 40ns)

*Brief: Final 10us makes sure that hold time after E becomes 0 is
satisfied (min. 10ns)

*Params:

*           None

*Returns:

*           None

*****************************************************/

void pulseEnablePin ()

{
```

```
    GPIOA->ODR&=~E;
    Systick_us_delay(10);
    GPIOA->ODR|=E;
    Systick_us_delay(10);
    GPIOA->ODR&=~E;
    Systick_us_delay(10);
}

/***| push_nibble() |*********************************//*
*Brief: Setup 4-bit transfer using 8-bit data
*Brief: 0 to 1 and then to 0 for data/command transfer
*Params:
*           (uint8_t) nibble: 8-bit data, but 4 most significant
*           digits are important only
*Returns:
*           None
*****************************************************/
void push_nibble (uint8_t nibble)
{
    GPIOA->ODR &=~(uint32_t) (DB4|DB5|DB6|DB7);
     // Clear PA4-7
          GPIOA->ODR |= nibble;

          // Send nibble with OR, CRITICAL: PA0-1 are
                                                    // NOT touched
thanks to the OR as nibble is made sure
                                                    // to be
nibble=XXXX0000;
    pulseEnablePin();                               // Transfer
with Enable Signal
}

/***| push_Byte() |***********************************//*
```

```
*Brief: Send 8 bit data to LCD with 2 nibbles of 4 bit length
*Params:
*           (uint8_t) byte: 8-bit data, send the first 4 first
*           followed by the last 4 bits using push_nibble()
*           CRITICAL: To avoid conflict nibble must be
*           nibble==XXXX0000 so that RS and E signals will not
*           be conflicting.
*Returns:
*           None
*****************************************************/
void push_Byte(uint8_t byte)
{
    uint8_t nibble;
    nibble = (byte & 0xF0);                             // Get the
most significant 4 digits of the nibble
    push_nibble(nibble);
    nibble = ((byte & 0x0F)<<4);                        // Get the
least significant 4 digits of the nibble
    push_nibble(nibble);                                // Delays in
the Enable signal are enough for cascaded 4-bit transfer
    Systick_us_delay(100);
}


/***| write_command() |*********************************//*
*Brief: RS=0, Writing commands to LCD
*Params:
*           (uint8_t) command: sending hexadecimal commands
*           using push_Byte()
*Returns:
*           None
*****************************************************/
void write_command(uint8_t command)
{
```

```
    GPIOA->ODR&= ~RS;
                                                //RS line=>0
    push_Byte(command);
}


/***| LCD_initialization() |*********************************//*
*Brief:     LCD is initialized with special instruction set
*Brief:     following Mazidi & NAIMI Book. Dr. Kandalaft's
initialization
*Brief:     has been commented. The waiting period between the
initialization
*Brief:     steps have been taken 10ms for all, but some instructions
(higher
*Brief:     number hexadecimal) can be done much faster.
*Params:
*           (uint8_t) command: sending hexadecimal commands
*           using push_Byte()
*Returns:
*           None
*****************************************************/
void LCD_initialization()
{
    //write_command(0x03);
    write_command(0x30);                                // set up LCD
hardware 8-bit interface, 1 line 5*7 pixel
    Systick_ms_delay(10);                               // same 10ms
delay for all
    write_command(0x30);                                // done x3 in
the book
    //write_command(0x03);
           Systick_ms_delay(10);
    write_command(0x30);
    //write_command(0x03);
           Systick_ms_delay(10);
    //write_command(0x02);
```

```
    write_command(0x20);                                    // 4-bit
interface 1 line 5*7 pixels

    Systick_ms_delay(10);

    //write_command(0x02);

           //Systick_ms_delay(10);

           write_command(0x28);                                  // 4-
bit interface 2 lines 5*7 pixels

    //write_command(0x08);

           Systick_ms_delay(10);

    write_command(0x06);                                    // Move cursor
right after each character

    //write_command(0x0F);

           Systick_ms_delay(10);

    write_command(0x01);                                    // Clear
screen, move cursor to home

    Systick_ms_delay(10);

    write_command(0x02);                                    // Move cursor
to top left character position

    //write_command(0x06);

           Systick_ms_delay(10);

    write_command(0x08);                                    // Blank the
display (without clearing)

    Systick_ms_delay(10);

    write_command(0x0F);                                    // Turn on
visible blinking-block cursor CAUTION: 0C makes it invisible!

    Systick_ms_delay(10);

}


/***| write_data() |*************************************//*

*Brief: RS=1, Writing data to LCD

*Params:

*           (uint8_t) command: sending hexadecimal data

*           using push_Byte()

*Returns:

*           None
```

```
**********************************************************/
void write_data(uint8_t data)
{
    GPIOA->ODR|= RS;                                    //RS line=>1
    push_Byte(data);
}



/***| welcome_screen() |*************************************//*
*Brief:          Prints welcome screen
*Params:
*           None
*Returns:
*           None
**********************************************************/
void welcome_screen(void){
char welcome[] =  "    Thomas Z    ";
char welcome1[] = "     Josh R     ";
char welcome2[] = "Vending Machine!";

     Systick_ms_delay(5);
     //write_command(Line_1);
     for(unsigned int i = 0; i< strlen(welcome); i++)
               write_data (welcome[i]);
     write_command(Line_2);
     for(int i = 0; i<16; i++)
               write_data (welcome1[i]);
     write_command(Line_4);
     for(int i = 0; i<16; i++)
               write_data (welcome2[i]);
```

```
}



/***| MainMenu_screen() |************************************//*
*Brief:          Prints Main Menu screen
*Params:
*           None
*Returns:
*           None
*******************************************************/
void MainMenu_screen(void){
char MainMenu[] =  "Cash:  $0.00";
char MainMenu1[] = "Selection:";
char MainMenu2[] = "[*]Admin";

      Systick_ms_delay(5);
      write_command(Line_1);
      for(int i = 0; i<12; i++)
                write_data (MainMenu[i]);
      write_command(Line_2);
      for(int i = 0; i<10; i++)
                write_data (MainMenu1[i]);
      write_command(Line_4);
      for(int i = 0; i<8; i++)
                write_data (MainMenu2[i]);
      write_command(Line_2+0xB);
}



/***| Admin_screen() |***************************************//*
```

```
*Brief:          Prints Admine password screen
*Params:
*           None
*Returns:
*           None
*****************************************************/
void Admin_screen(void){
char admin[] = "Admin Passcode:";
char admin1[] = "______";
char admin2[] = "[#]Accept";
char admin3[] = "[*]Cancel";

     Systick_ms_delay(5);
  write_command(Line_1);
     for(unsigned int i = 0; i< strlen(admin); i++)
               write_data (admin[i]);
     write_command(Line_2);
     for(unsigned int i = 0; i<strlen(admin1); i++)
               write_data (admin1[i]);
     write_command(Line_3);
     for(unsigned int i = 0; i<strlen(admin2); i++)
               write_data (admin2[i]);
     write_command(Line_4);
     for(unsigned int i = 0; i<strlen(admin3); i++)
               write_data (admin3[i]);
     write_command(Line_2);
     Systick_ms_delay(5);
}



/***| settings_screen() |*********************************//*
```

```
*Brief:          Prints Admin screen
*Params:
*           None
*Returns:
*           None
*******************************************************/
void settings_screen(void){
char settings[] = "     Admin";
char settings1[] = "[1]Temperature";
char settings2[] = "[2]Inventory";
char settings3[] = "[3]Cash  [*]Back";

     Systick_ms_delay(2);
  write_command(Line_1);
     for(unsigned int i = 0; i< strlen(settings); i++)
               write_data (settings[i]);
     write_command(Line_2);
     for(unsigned int i = 0; i<strlen(settings1); i++)
               write_data (settings1[i]);
     write_command(Line_3);
     for(unsigned int i = 0; i<strlen(settings2); i++)
               write_data (settings2[i]);
     write_command(Line_4);
     for(unsigned int i = 0; i<strlen(settings3); i++)
               write_data (settings3[i]);
     write_command(Line_2);
     Systick_ms_delay(2);
}



/***| Temperature_screen() |************************************//*
```

```
*Brief:          Prints Temperature screen
*Params:
*           None
*Returns:
*           None
*********************************************************/
void Temperature_screen(void){
char Temp[] = "Admin: Temp";
char Temp2[] = "[*]Back";

     Systick_ms_delay(2);
  write_command(Line_1);
     for(unsigned int i = 0; i< strlen(Temp); i++)
                write_data (Temp[i]);
     write_command(Line_2+0x8);
     write_data(0xDF);
     write_data (0x43);
     write_command(Line_4+0x9);
     for(unsigned int i = 0; i<strlen(Temp2); i++)
                write_data (Temp2[i]);
}



/***| Inventory_screen() |*************************************//*
*Brief:          Prints Inventory screen
*Params:
*           None
*Returns:
*           None
*********************************************************/
void Inventory_screen(void){
```

```
char inventory[] = "Admin: Inventory";
char inventory1[]= "  Red      Blue ";
char inventory2[]= "  Orange   Green";
char inventory3[]= "[#]Stock [*]Back";
     Systick_ms_delay(2);
  write_command(Line_1);
     for(unsigned int i = 0; i< strlen(inventory); i++)
               write_data (inventory[i]);
     write_command(Line_2);
     for(unsigned int i = 0; i<strlen(inventory1); i++)
               write_data (inventory1[i]);
     write_command(Line_3);
     for(unsigned int i = 0; i<strlen(inventory2); i++)
               write_data (inventory2[i]);
     write_command(Line_4);
     for(unsigned int i = 0; i<strlen(inventory3); i++)
               write_data (inventory3[i]);
     Systick_ms_delay(2);

}

/***| Stock() |*********************************//*
*Brief:          Prints Inventory screen
*Params:
*           Stock values for each item
*Returns:
*           None
*****************************************************/
void Stock(uint8_t red, uint8_t blue, uint8_t orange, uint8_t green1){
     char strRed[2], strBlue[2], strOrange[2], strGreen[2];

     write_command(Line_2);
```

```
      sprintf(strRed, "%d", red);
      write_data(strRed[0]);

      write_command(Line_2+0x9);
      sprintf(strBlue, "%d", blue);
      write_data(strBlue[0]);

      write_command(Line_3);
      sprintf(strOrange, "%d", orange);
      write_data(strOrange[0]);

      write_command(Line_3+0x9);
      sprintf(strGreen, "%d", green1);
      write_data(strGreen[0]);
      Systick_ms_delay(2);
}

/***| Cash_screen() |*************************************//*
*Brief:          Prints Cash screen
*Params:
*           None
*Returns:
*           None
*************************************************************/
void Cash_screen(void){
char Cash[] = "Admin: Cash";
char Cash2[]= "[#]Withdraw";
char Cash3[]= "[*]Back";

      Systick_ms_delay(2);
  write_command(Line_1);
      for(unsigned int i = 0; i< strlen(Cash); i++)
```

```
                write_data (Cash[i]);
     write_command(Line_2);
                write_data ('$');
     write_command(Line_3);
     for(unsigned int i = 0; i<strlen(Cash2); i++)
                write_data (Cash2[i]);
     write_command(Line_4);
     for(unsigned int i = 0; i<strlen(Cash3); i++)
                write_data (Cash3[i]);
     Systick_ms_delay(2);

}

/***| Dispensed() |*************************************//*
*Brief:          Prints Displensed screen
*Params:
*           Key that was pressed and amount of money inserted
*Returns:
*           None
*********************************************************/
void Dispensed(float money, int num){
char dispensed[] = "Dispensed Red";
char dispensed1[] ="Dispensed Blue";
char dispensed2[] ="Dispensed Orange";
char dispensed3[] ="Dispensed Green";

char change[] = "Change $";

     char strnum[5];
     write_command(Line_3);
     switch(num){
          case 1:
```

```
      for(unsigned int i = 0; i< strlen(dispensed); i++)
                  write_data (dispensed[i]);
            break;
            case 2:
                        for(unsigned int i = 0; i< strlen(dispensed1);
i++)
                              write_data (dispensed1[i]);
            break;
            case 3:
                        for(unsigned int i = 0; i< strlen(dispensed2);
i++)
                              write_data (dispensed2[i]);
            break;
            case 4:
                        for(unsigned int i = 0; i< strlen(dispensed3);
i++)
                              write_data (dispensed3[i]);
            break;
      }
      write_command(Line_4);
      for(unsigned int i = 0; i< strlen(change); i++)
                              write_data (change[i]);
      sprintf(strnum, "%.2lf", money);
      for(unsigned int i = 0; i<4; i++)
                                    write_data(strnum[i]);
}

/***| NotEnough() |*********************************//*
*Brief:          Prints Not enough screen
*Params:
*           price of item
*Returns:
*            None
```

```
*********************************************************/
void NotEnough(float price){
char notenough[] = "Not Enough Money";
char notenough1[] = "Need:  $";

     char strprice[5];
     write_command(Line_3);
     for(unsigned int i = 0; i< strlen(notenough); i++)
                          write_data (notenough[i]);
     write_command(Line_4);
     for(unsigned int i = 0; i< strlen(notenough1); i++)
                          write_data (notenough1[i]);
     sprintf(strprice, "%.2lf", price);
     for(unsigned int i = 0; i<4; i++)
                     write_data(strprice[i]);
}

/***| Returned() |*********************************//*
*Brief:          Prints return screen
*Params:
*          amount of money inserted
*Returns:
*           None
*********************************************************/
void Returned(float money){
char returned[] = "Return Requested";
char returned1[] = " Returned";
char strmoney[5];

     Systick_ms_delay(2);
     write_command(Line_3);
     for(unsigned int i = 0; i< strlen(returned); i++)
```

```
                        write_data (returned[i]);
    write_command(Line_4);
    write_data('$');
    sprintf(strmoney, "%.2lf", money);
    for(unsigned int i = 0; i<4; i++)
                    write_data(strmoney[i]);
    for(unsigned int i = 0; i< strlen(returned1); i++)
                        write_data (returned1[i]);
}

/***| OutOfStock() |*********************************//*
*Brief:          Prints Out of stock screen
*Params:
*               None
*Returns:
*           None
*******************************************************/
void OutOfStock(void){
char outStock[] = "Out Of Stock";
char outStock1[] = "            ";

    write_command(Line_3);
    for(unsigned int i = 0; i< strlen(outStock); i++)
                        write_data (outStock[i]);
    write_command(Line_4);
    for(unsigned int i = 0; i< strlen(outStock1); i++)
                        write_data (outStock1[i]);

}
```

Figure A-4: Timer.c for Final Project

```
#include "stm32f4xx.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <math.h>
#include "Timer.h"




/***| TIM2_Initi() |*************************************//*
*Brief:          Initialization for Timer 2 for IR emmitter
*Pin Config:PA15
*Params:
*           None
*Returns:
*           None
*************************************************************/
void TIM2_Initi(void){
    RCC->AHB1ENR |= 1;     /* enable GPIOA clock */
    GPIOA->MODER &= ~(3 << (15 * 2));
            GPIOA->MODER |= 2 << (15 * 2);
            GPIOA->AFR[1] &= ~(0xF << (15 - 8) * 4);
            GPIOA->AFR[1] |= 1 << (15 - 8) * 4; // AF1 for TIM2_CH1
            RCC->APB1ENR |= 1;
}
/***| TIM2_Count() |*************************************//*
*Brief:          Controller for Timer 2 to output 10Hz
*
*Params:
*           None
```

```
*Returns:
*           None
**********************************************************/
void TIM2_Count(void){
    /* setup TIM1 */
    TIM2->PSC = 100 - 1;
    TIM2->ARR = 16000 - 1;
    TIM2->CNT = 0;
    TIM2->CCMR1 = 0x0060;
    TIM2->CCER = 1;          /* enable PWM Ch1 */
    TIM2->CCR1 = TIM2->ARR / 2;
    TIM2->CR1 = 1;
}
/***| TIM3_Initi() |*************************************//*
*Brief:          Initialization for Timer 3 For IR Reciever
*Pin Config:PB4
*Params:
*           None
*Returns:
*           None
**********************************************************/
void TIM3_Initi(void){
    RCC->AHB1ENR |= 2;     /* enable GPIOB clock */
    GPIOB->AFR[0] &= ~0xF0000;
           GPIOB->AFR[0] |= 2<<16;     /* PB4 pin for tim2 */
    GPIOB->MODER &= ~(3<<8);
    GPIOB->MODER |= 2<<8;
           RCC->APB1ENR |=2;     /* enable TIM3 clock */
}
/***| TIM3_Control() |***********************************//*
*Brief:          Controller for Timer 3 to capture at 1000KHz
*
```

```
*Params:
*           None
*Returns:
*           None
**********************************************************/
void TIM3_Control(void){//PB4
           TIM3->PSC = 8000  - 1;

           TIM3->CNT = 0;
    TIM3->CCMR1 = 0x0041;        /* PWM mode */
    TIM3->CCER = 0xB;              /* enable PWM Ch1 */
    TIM3->CR1 = 1;
           TIM3->DIER |= 3;
           NVIC_EnableIRQ(TIM3_IRQn);

}
/***| TIM10_Initi() |*********************************//*
*Brief:         Initialization for Timer 10 for servo 1
*Pin Config:PB8
*Params:
*           None
*Returns:
*           None
**********************************************************/
void TIM10_Initi(void){
    RCC->AHB1ENR |= 2;     // enable GPIOB clock
    GPIOB->MODER &= ~(3 << (8 * 2));
    GPIOB->MODER |= 2 << (8 * 2);
    GPIOB->AFR[1] &= ~(0xF << (8 - 8) * 4);
    GPIOB->AFR[1] |= 3 << (8 - 8) * 4; // AF3 for TIM10_CH1

    RCC->APB2ENR |= 1<<17;   // enable TIM10 clock
```

```
}
/***| TIM10_Count() |*************************************//*
*Brief:          Controller for Timer 10 to output at 50Hz
*
*Params:
*           None
*Returns:
*           None
*********************************************************/
void TIM10_Count(void){
    /* setup TIM10 */
    TIM10->PSC = 10 - 1;
    TIM10->ARR = 32000  - 1;
    TIM10->CNT = 0;
    TIM10->CCMR1 = 0x0060;
    TIM10->CCER = 1;          // enable PWM Ch1
    TIM10->CCR1 = 2000;
    TIM10->CR1 = 1;
}
/***| TIM11_Initi() |*************************************//*
*Brief:          Initialization for Timer 11 for servo 2
*Pin Config:PB9
*Params:
*           None
*Returns:
*           None
*********************************************************/
void TIM11_Initi(void){
    RCC->AHB1ENR |= 2;     // enable GPIOB clock
    GPIOB->MODER &= ~(3 << (9 * 2));
    GPIOB->MODER |= 2 << (9 * 2);
    GPIOB->AFR[1] &= ~(0xF << (9 - 8) * 4);
```

```
    GPIOB->AFR[1] |= 3 << (9 - 8) * 4; // AF3 for TIM11_CH1


    RCC->APB2ENR |= 1<<18;   // enable TIM11 clock
}
/***| TIM11_Count() |***********************************//*
*Brief:          Controller for Timer 11 to output at 50Hz
*
*Params:
*           None
*Returns:
*           None
*************************************************************/
void TIM11_Count(void){
    /* setup TIM11 */
    TIM11->PSC = 10 - 1;
    TIM11->ARR = 32000  - 1;
    TIM11->CNT = 0;                                 // set the counter
to 0
    TIM11->CCMR1 = 0x0060;                          // set the
capture/compare mode to PWM mode 1
    TIM11->CCER = 1;                                // enable PWM
output on channel 1
    TIM11->CCR1 = 2000;                             // set the duty
cycle to 50%
    TIM11->CR1 = 1;                                 // enable the TIM11
counter
}


/***| ADC_Init() |*********************************//*
*Brief:          Initialization for ADC 1 for internal temp sensor


*Params:
*           None
*Returns:
```

```
*           None
*************************************************************/
void ADC_Init(void)
{
           RCC->APB2ENR |= 1<<8;
                                              // ENABLE ADC1 RCC
           ADC->CCR |= 0x800000;
    ADC->CCR &= ~0x400000;          /* VBATE must be disabled for temp
sensor */
           ADC1->SMPR1 = 0x40000000;        /* sampling time minimum 10
us */
           ADC1->SMPR2      |=   0;
                                                    // Sampling
Time: 3 cycles in Ch10
           ADC1->CR1             = 0x10;
                                                  // CH10 Selected with
Watchdog, 12 bit res
           ADC1->CR2             = 0;
                                                    // Reset and
thereby right align
           ADC1->CR2             = 1<<0;
                                                  // ADC enabled,
Single Conversion
           ADC1->SQR1      = 0;
                                                  // 1 conversion will
take place
           ADC1->SQR3      = 18;
                                        // CH1 ON
           ADC1->CR2 |= 1;                  /* enable ADC1 */
                                                    // CH1 ON
}


/***| Internal_Temperature_Sensor()
|*********************************//*
*Brief:          calculate the temperature in Celsius
*
*Params:
*           None
```

```
*Returns:
*           None
*********************************************************/
float Internal_Temperature_Sensor(void){
           float Vref=3.3;
                                                    // reference
voltage
           float result;
           float tempC;
           uint16_t code;


                      ADC1->CR2 |= 1<<30;
                                 /* start a conversion */
                      while(!(ADC1->SR & 2)) {}
                      /* wait for conv complete */
        code = ADC1->DR &0x00000FFF;                                /*
read conversion result */
                      result = ((double)code / 4095 * Vref);
                            tempC = ((result-.76) / .0025+25);


                            return tempC;
                      }


/***| ServoMotor() |*************************************//*
*Brief:          Move correct servo arm depending on what key was
pressed
*
*Params:
*           key pressed from keypad
*Returns:
*           None
*********************************************************/
void ServoMotor(int num){
     switch(num){
```

```
            case 1:
                  TIM10->CCR1 = 1500;
            break;
            case 2:
                  TIM10->CCR1 = 2500;
            break;
            case 3:
                  TIM11->CCR1 = 1500;
            break;
            case 4:
                  TIM11->CCR1 = 2500;
            break;
      }

}
```

Figure A-5: Keypad.h for Final Project

```
/*
 * keypad.h
 *
 *  Created on: Dec 1, 2022
 *      Author: Thomas Zoldowski
 */
#include "stm32f4xx.h"

#ifndef KEYPAD_H_
#define KEYPAD_H_

#define C   (uint32_t) 0x04
                        // Port C
```

```
#define R0 (uint32_t) 0x01
                            // Row bits 0
#define R1 (uint32_t) 0x02
                            // Row bits 1
#define R2 (uint32_t) 0x04
                            // Row bits 2
#define R3 (uint32_t) 0x08
                            // Row bits 3
#define C0 (uint32_t) 0x10
                            // Col bits 0
#define C1 (uint32_t) 0x20
                            // Col bits 1
#define C2 (uint32_t) 0x40
                            // Col bits 2


uint8_t Hex2Bit (uint32_t hex_num);
void Keypad_Init (void);                                    // Initialize
Functions
void SysTick_Init(void);                                    // Initialize
Functions
void SysTick_ms_Delay(uint16_t delay);                        //
Initialize Functions
void Print_Keys (uint16_t *numptr);                         // Print
Keypress
void Print_Keys2 (uint16_t *numptr);
uint8_t Read_Keypad(uint16_t *numptr);                      // Keypad scan
subroutine
uint8_t *PIN_Entry(uint16_t *digitptr, uint8_t *pin, uint8_t turn,
uint16_t *numptr, uint16_t *exitptr);



extern int* key;


// PIN Entry prompt Function
void keypad_isr(int row);
#endif /* KEYPAD_H_ */
```

Figure A-6: LCD.h for Final Project

```
/*
 * LCD.h
 *
 *  Created on: Dec 1, 2022
 *      Author: Thomas Zoldowki
 */


#ifndef LCD_H_
#define LCD_H_


#define A   (uint32_t)  0x01
                              // Port A
#define RS (uint32_t)  0x02
                             // PA1
#define E  (uint32_t)  0x01
                             // PA0
#define DB4 (uint32_t) 0x10
                             // PA4
#define DB5 (uint32_t) 0x20
                             // PA5
#define DB6 (uint32_t) 0x40
                             // PA6
#define DB7 (uint32_t) 0x80
                         // PA7
#define Line_1 0x80
#define Line_2 0xC0
#define Line_3 0x90
#define Line_4 0xD0


void Systick_init (void);
void Systick_us_delay (uint16_t usdelay);
void Systick_ms_delay (uint16_t msdelay);
void LCD_init_pins (void);
```

```
void pulseEnablePin (void);
void push_nibble (uint8_t nibble);
void push_Byte (uint8_t byte);
void LCD_initialization(void);
void write_command (uint8_t command);
void write_data(uint8_t data);
uint8_t Hex2Bit (uint32_t hex_num);


//menus
void welcome_screen(void);

void MainMenu_screen(void);

void Admin_screen(void);

void settings_screen(void);

void Temperature_screen(void);

void Inventory_screen(void);

void Stock(uint8_t red, uint8_t blue, uint8_t orange, uint8_t green);

void Cash_screen(void);

void Dispensed(float money, int num);

void NotEnough(float price);

void Returned(float money);
```

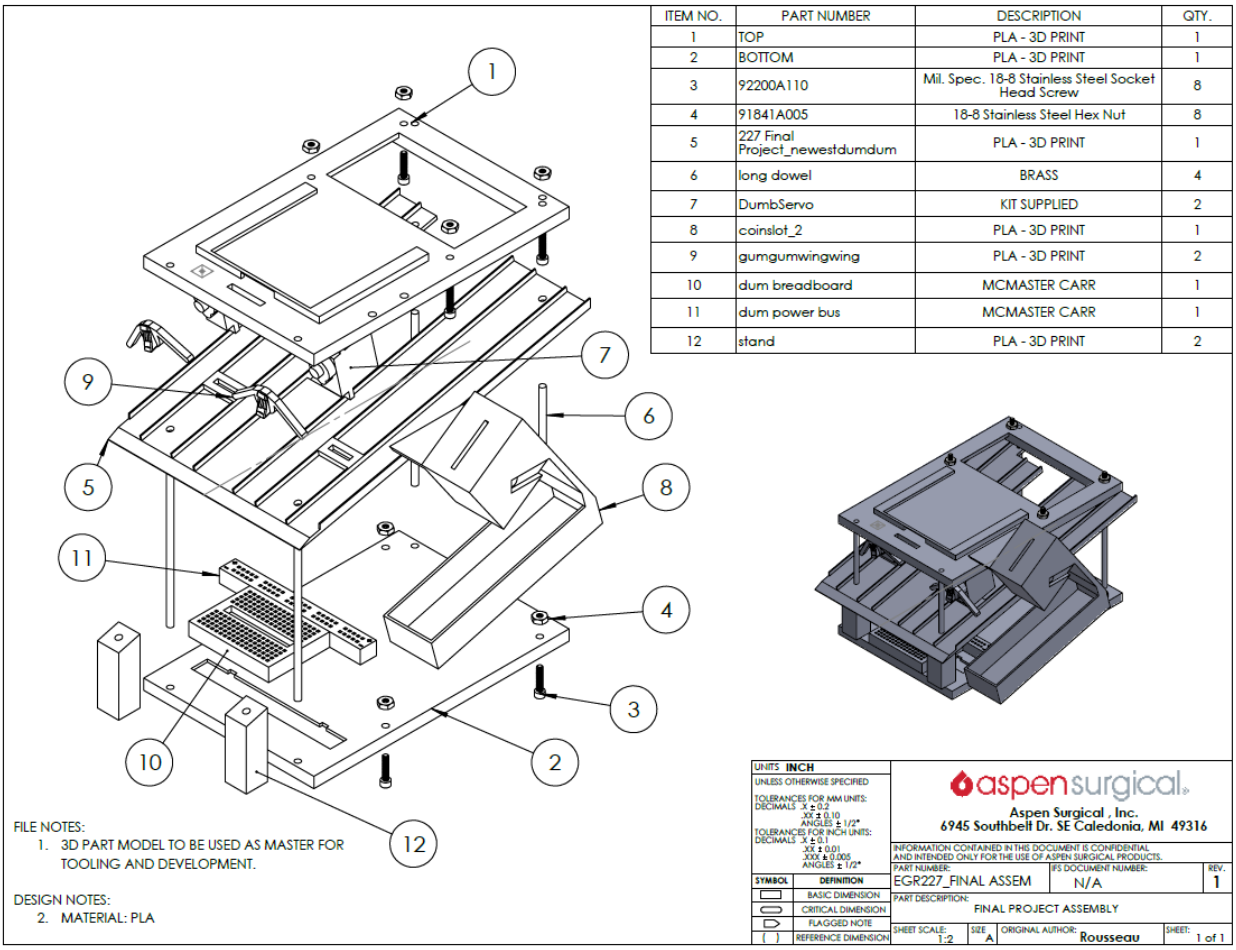
```
void OutOfStock(void);


#endif /* LCD_H_ */
```

Figure A-7: Timer.h for Final Project

```
/*
 * keypad.h
 *
 *  Created on: Dec 1, 2022
 *      Author: Thomas Zoldowski
 */
#include "stm32f4xx.h"


void TIM2_Initi(void);     //Timer 2 initialization
void TIM2_Count(void);     //Timer 2 controller
void TIM3_Initi(void);     //Timer 3 initialization
void TIM3_Control(void);//Timer 3 controller
void TIM10_Initi(void);    //Timer 10 initialization
void TIM10_Count(void);    //Timer 10 controller
void ADC_Init(void);       //ADC initialization
float Internal_Temperature_Sensor(void);   //Temperature Calculater
void ServoMotor(int num);  //Servo Mover
void TIM11_Initi(void);          //Timer 11 initialization
void TIM11_Count(void);          //Timer 11 Controller
```

Appendix B: Drawings

Figure B-1: Project Illustrated Parts Breakdown



ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	TOP	PLA - 3D PRINT	1
2	BOTTOM	PLA - 3D PRINT	1
3	92200A110	Mil. Spec. 18-8 Stainless Steel Socket Head Screw	8
4	91841A005	18-8 Stainless Steel Hex Nut	8
5	227 Final Project_newestdumdum	PLA - 3D PRINT	1
6	long dowel	BRASS	4
7	DumbServo	KIT SUPPLIED	2
8	coinslot_2	PLA - 3D PRINT	1
9	gumgumwingwing	PLA - 3D PRINT	2
10	dum breadboard	MCMASTER CARR	1
11	dum power bus	MCMASTER CARR	1
12	stand	PLA - 3D PRINT	2

UNITS INCH
UNLESS OTHERWISE SPECIFIED
TOLERANCES FOR MM UNITS:
DECIMALS .X ± 0.2
.XX ± 0.10
ANGLES ± 1/2°
TOLERANCES FOR INCH UNITS:
DECIMALS .X ± 0.1
.XX ± 0.01
.XXX ± 0.005
ANGLES ± 1/2°

SYMBOL	DEFINITION
	BASIC DIMENSION
	CRITICAL DIMENSION
	FLAGGED NOTE
( )	REFERENCE DIMENSION

aspensurgical
Aspen Surgical , Inc.
6945 Southbelt Dr. SE Caledonia, MI 49316

INFORMATION CONTAINED IN THIS DOCUMENT IS CONFIDENTIAL AND INTENDED ONLY FOR THE USE OF ASPEN SURGICAL PRODUCTS.

PART NUMBER:	IFS DOCUMENT NUMBER:	REV.
EGR227_FINAL ASSEM	N/A	1

PART DESCRIPTION: FINAL PROJECT ASSEMBLY

SHEET SCALE:	SIZE	ORIGINAL AUTHOR:	SHEET:
1:2	A	Rousseau	1 of 1