

# EE260 Final Project

Section 10 - Team 12

Joseph Drahos 0777509

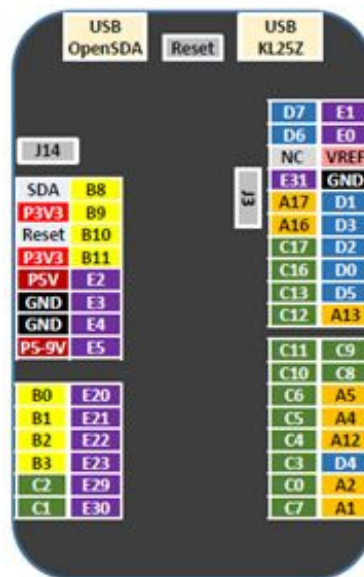
Tarak Patel 0756267

## Introduction

The FRDM-KL25Z microcontroller is a versatile board that allows the use of many unique input and output elements. In this project, various peripherals were used with the microcontroller along with functionalities of the microcontroller such as analog to digital conversions (ADC) and serial communication from the UART of the board to PuTTY. This report includes details of the design, the design code, and pictures of the system.

## Objective

The objective of this project was to use various peripherals and functionalities of the FRDM KL25Z microcontroller to design an interrupt driven system. The system includes use of peripherals such as the LCD, keypad, Servo motor, DC Motor, ultrasonic sensor and passive buzzer. The system also uses analog to digital conversion (ADC) and the Universal Asynchronous Receiver/Transmitter (UART) to provide input to the system and to serve as a way to control various peripherals in the system. The port connections of the FRDM KL25Z microcontroller are as follows.



## Design

The functions of the system are as follows:

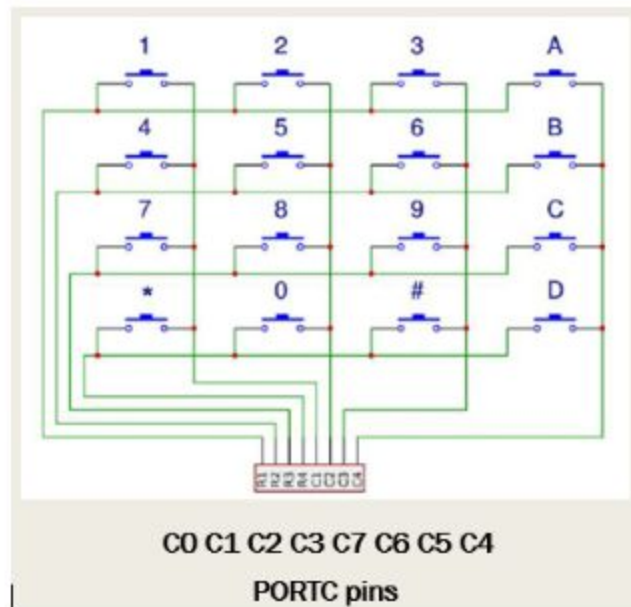
- control the position of a Servo using PWM using the hardware and software
- control the speed of a dc motor using the hardware and software
- put the servo in scan mode so it moves from left to right and back
- control the position of the servo from a PC terminal

- play a siren on the passive buzzer speaker
- display messages and data on the LCD panel and on a PC serial terminal

The interrupt driven system uses a keypad that allows the user to select which of the five modes the system operates on. The five modes of the system are described below:

#### Keypad:

The keypad is used as a selector system within this project that uses port D and the port D interrupt to change between the 5 different modes within the project.



Port D pins are used instead of Port C on the keypad as seen on the table below.

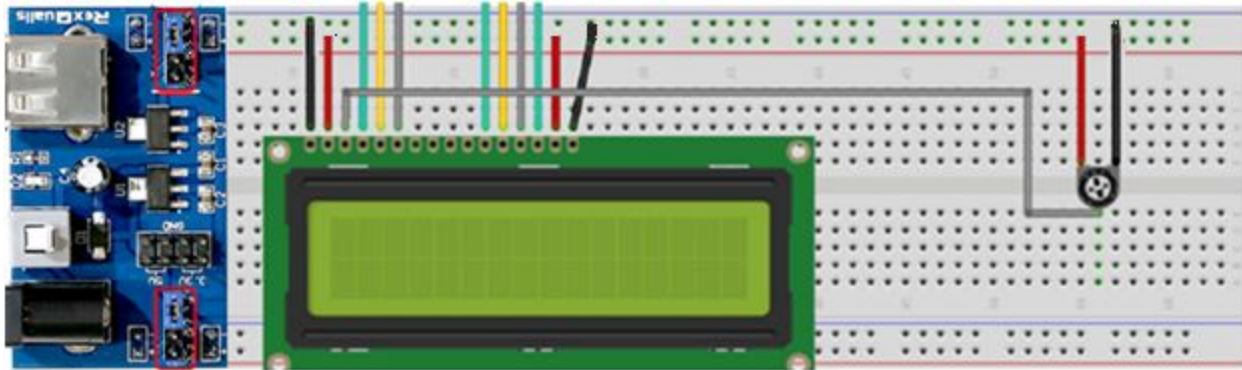
FRDM	PTD0	PTD1	PTD2	PTD3	PTD7	PTD6	PTD5	PTD4
Keyboard	Pin 8	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1
Keyboard	R1	R2	R3	R4	C1	C2	C3	C4
C-Code	Row 1	Row 2	Row 3	Row 4				

The keypad is initialized with the port D interrupt on PTD5 as that is the column with the '#' character which causes the system to select a new mode. Each key is converted to a character using the character array `char hexKeys[] = {0x0,'A','3','2','1','B','6','5','4','C','9','8','7','D','E','0','F'}.`

Within the port D interrupt handler it runs the initialization functions for the corresponding mode to set up all the software and peripherals.

### LCD:

The LCD is utilized to display which mode the system is currently running on and to also supply useful information on which mode. The LCD is sent data using the following pin configuration.



FRDM	PTC13	PTC12	PTC11	PTC10		PTC9		PTC8
LCD	D7	D6	D5	D4		EN		RS

The pin configuration is initialized within the code as follows:

```
PORTC->PCR[8] = 0x100; /* make PTC8 pin as GPIO */
PORTC->PCR[9] = 0x100; /* make PTC9 pin as GPIO */
PORTC->PCR[10] = 0x100; /* make PTC10 pin as GPIO */
PORTC->PCR[11] = 0x100; /* make PTC11 pin as GPIO */
PORTC->PCR[12] = 0x100; /* make PTC12 pin as GPIO */
PORTC->PCR[13] = 0x100; /* make PTC13 pin as GPIO */
PTC->PDDR |= 0x3F00; /* make PTC 13-8 as output pins */
```

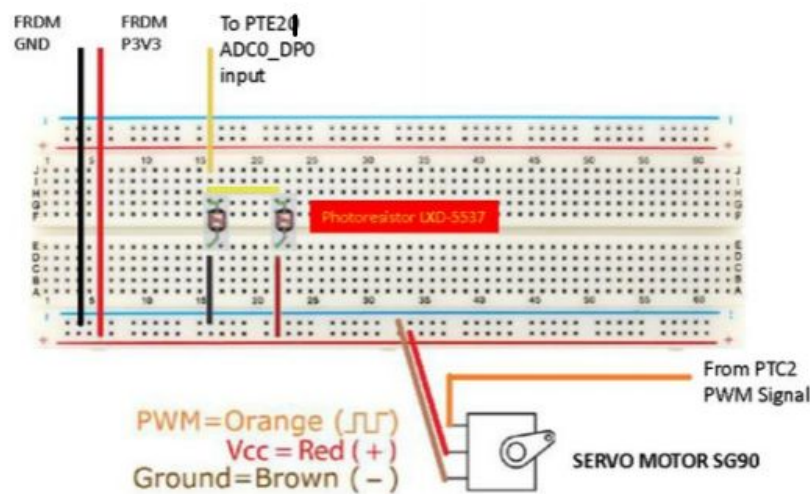
### PuTTY/UART:

PuTTY is used to communicate back and forth between the FRDM board using UART0. A system clock of 48 MHz was used with a Baud Rate of 115200 along with an oversampling ratio (OSR) of 15 ( $15+1=16$ ). Port A pin 1 was used as the UART0 receive pin and port A pin 2 was used as the UART0 transmit pin. The BDH register contained 0 while the BDL register contained 26 to establish the baud rate of 115200.

### Mode 1: Potentiometer and Photoresistor based voltage divider controls servo

The analog to digital (A/D) values from the potentiometer and photoresistor based voltage divider controls the position of the servo. The servo is primarily controlled by the potentiometer unless the mid point voltage of the photoresistor voltage divider is within the range of  $1.65 \pm 0.35$  V. If the mid-point voltage is outside this range then the potentiometer does not have control of the servo. The potentiometer varies between 0-3.3V and the photoresistor based voltage divider uses a 3.3V source. The pin connections of this mode are as follows:

Connection	Port
Analog input voltage from Potentiometer	PTE20
Analog input voltage from Photoresistor based voltage divider	PTE21
PWM Signal for Servo	PTC2



The following calculations were done to set the values of the result variable from the photoresistor based voltage divider to meet the project requirements:

$$1615 * 3300 / 4096 = 1.3 \text{ V}$$

$$2485 * 3300 / 4096 = 2.0 \text{ V}$$

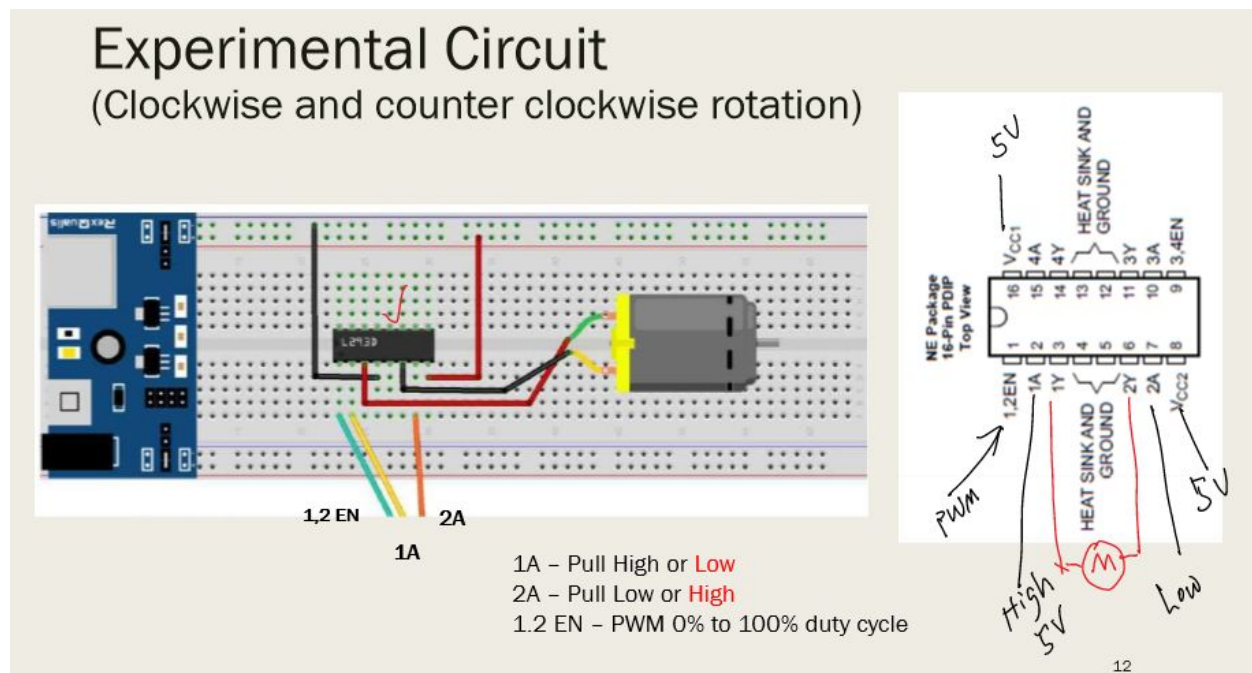
Where 1615 and 2485 are the lower and upper limits of the result value for the photoresistor based voltage divider that allows for the potentiometer to control the servo.

Mode 2: Potentiometer and Ultrasonic Sensor controls the DC Motor

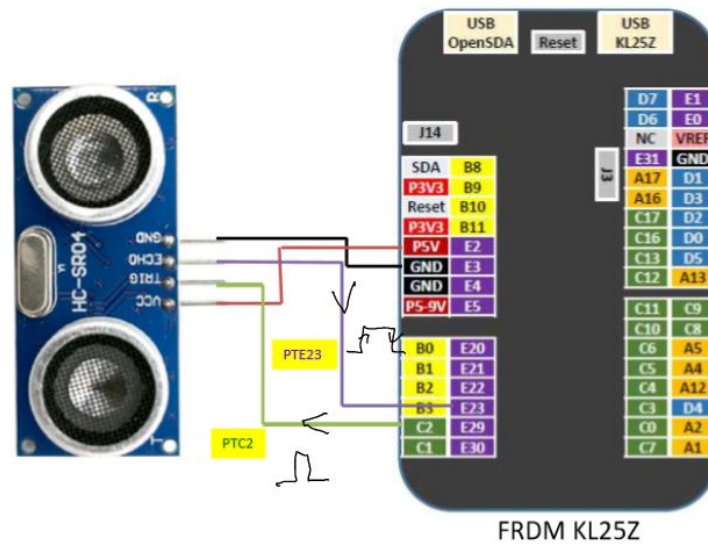
Mode 2 uses Analog to Digital Conversion (A/D) to take input from the potentiometer in the same format as Mode 1 and Pulse Width Modulation (PWM) to send a soundwave using the ultrasonic sensor and takes input from the reflection of the wave to calculate the distance between the sensor and the nearest object. The (A/D) conversion is used to control the DC motor with a pulse controlled by the calculation

$$\text{TPM0} \rightarrow \text{CONTROLS}[3].\text{CnV} = \text{result1} * 14.$$

This conversion varies the duty cycle to the motor from 0% to 93% where result1 is the A/D result and the TPM0 channel 3 CnV register controls the duty cycle of the wave sent to the motor. This PWM is sent to pin C4 which in turn is sent to pin 1 on an L293D Chip which acts as an H-Bridge for the DC Motor, this setup is seen in the diagram below.



The configuration for the Ultrasonic Sensor was used as in the following diagram.



The distance calculation is done by first calculating the pulse width of the reflected wave after 2 TPM2 interrupts. The counter results are stored in TPM2->C1V registers cnt[0] & cnt[1] and the pulse width is calculated by

$$\text{cnt}[1] - \text{cnt}[0], \text{ if } \text{cnt}[1] > \text{cnt}[0]$$

$$\text{or, } \text{cnt}[1] - \text{cnt}[0] + \text{MOD} + 1, \text{ if } \text{cnt}[1] < \text{cnt}[0].$$

Using the value of pulse width the distance to the object can be calculated by the equation,

$$\text{Distance} = \text{pulsetwidth} * (4/3 \text{ us}) / 2 * (0.0343) \text{ cm/us}.$$

If this distance is less than 2 feet the blue LED is put on to alert the user that an object is within 2 feet of the sensor. If the distance is less than 1 foot the red LED is activated to alert the user that an object is within 1 foot of the sensor. In addition a 0 is written into the TPM0 channel 3 CnV register to stop the motor from spinning while an object is within a foot.

### Mode 3: Servo scan mode

Mode 3 uses edge-aligned Pulse Width Modulation to control the servo motor and Universal Asynchronous Receiver/Transmitter (UART) as an input to select the different speeds the motor turns at. The setup of the servo motor is the same as in Mode 1 where its PWM signal comes from pin C2. The servo motor can angle from -90° to 90° using duty cycles of 2.7% to 12.6%. This mode makes the servo motor scan from -90° to 90° by setting the TPM0 channel 1 CnV register as follows,

$$\text{TPM0->CONTROLS}[1].\text{CnV} = 1620 + i \cdot (5940/1800).$$

Where  $i$  alternates counting up to a value of 1800 and down to a value of 0 and the speed at which it scans is controlled by the line,

$$\text{delayMs}(7/\text{result}).$$

Result is taken as an input using UART from PuTTY. The baud rate is set to 115200 on a system clock of 42 MHz. The user inputs a speed 1-4 into PuTTY and the result is used to reduce the delay between the increase/decrease of  $i$  and in turn a decrease in the time between the changes in the angle.

#### Mode 4: Manual position control of the servo from the PC serial terminal PuTTY

Mode 4 also used edge-aligned PWM such as in modes 1 and 3 to control the servo motor with a duty cycle ranging from 2.7% to 12.6%. In this mode UART is also used as the input from PuTTY but instead controls the exact angle of the motor in contrast with Mode 3 which had the motor scan back and forth. The user enters a 4 digit number on PuTTY which will represent an angle 0.0-180.0 where the fourth digit is the tenths place. This input is then converted to PWM by the conversion

$$\text{TPM0->CONTROLS}[1].\text{CnV} = 1620 + (\text{result}) \cdot (5940/1800).$$

Result is the 4 digit number from PuTTY and any value from 0000 to 1800 will send a PWM signal to the servo with a duty cycle ranging from 2.7% to 12.6%.

#### Mode 5: Manual speed control of the DC Motor from the PC serial terminal PuTTY

In this mode, the user controls the speed of the motor through input in the PC serial terminal PuTTY. The ultrasonic sensor is also used as it was in mode 2. The user can input the speed of the motor on a scale from 1-5 with 5 being the maximum speed. The ultrasonic sensor's function is the same as its function in mode 2 where if a moving object is within two feet it will alert the user with a blue LED. If the object is within 1 foot, then the ultrasonic sensor alerts the user by turning on a red LED and stopping the motors functionality until the object is outside of the 1 foot distance. The circuit setup of this mode is the same setup as mode 2. The use of the passive buzzer as a siren with varying frequency was not able to be satisfied as the implementation of it in the system caused a number of errors and prevented the rest of the system from functioning.

The checklist below shows which requirements were satisfied and which requirements were unable to be satisfied.

Core Requirements	Done	Not done
<b>Mode Selection capability – Human Interface</b>		



Matrix Keyboard can select one of five operating modes	x	
<b>Mode 1</b>		
Potentiometer(Pot) & photo sensor shall control the position of a Servo Motor	x	
The servo shall be controller primarily by the Potentiometer	x	
The Pot shall control servo, if the photo sensor circuit remains $1.65 \pm 0.35$ V	x	
<b>Mode 2</b>		
The DC motor shall be controlled by a potentiometer	x	
The DC motor shall use 5 V with a h-bridge (L295D)	x	
The Ultrasonic sensor shall alert with a blue LED when object between 2 and 1 foot	x	
The Ultrasonic sensor shall alert with a siren when object between 2 and 1 foot		x
The siren frequency shall vary between 500 to 3000 Hz in the above distance		x
The DC motor stops when the object is within 1 foot from the ultrasonic sensor	x	
<b>Mode 3</b>		
The shall repeatedly scan from -90 to +90 degrees and back	x	
User shall be able to control the scan rate using putty (serial terminal)	x	
<b>Mode 4</b>		
The user shall be able to control the angular position of the servo	x	
The angular resolution should be less than 0.1 degree	x	
<b>Mode 5</b>		
The user should be able to control the speed of the motor using putty	x	
The ultrasonic sensor overrides user as in Mode 2	x	
<b>Other requirements</b>		
<b>LCD Display:</b>		
Operating Mode is displayed on the first line	x	
Other useful information shown on the second line	x	
<b>UART – communication, human interface and display</b>		
Message and the LCD data should also be displayed on the Putty	x	
<b>Design Constraint</b>		
No software function and polling used (except uart tx function)	x	

The criteria that could not be met in the design were the use of the passive buzzer as a siren when using the ultrasonic sensor due to the fact that when the buzzer was added, our system failed and we had a time constraint to see what was interfering and due to the time constraint we could not use SysTick to create delays and instead used the delayMs and delayUs functions to create the delays needed.

### Code

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

```

#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"

#define RS 1 /* BIT0 mask */
#define RW 2 /* BIT1 mask */
#define EN 4 /* BIT2 mask */
void keypad_init(void);
void keypad_disable(void);
char keypad_getkey(void);
void LCD_nibble_write(unsigned char data, unsigned char control);
void LCD_command(unsigned char command);
void LCD_data(unsigned char data);
void LCD_init(void);
void delayMs(int n);
char hexKeys[] = {0x0,'A','3','2','1','B','6','5','4','C','9','8','7','D','#','0','*'};
char keypressed;
bool modeSelect;
unsigned char key;
void delayUs(int n);

void UART0_init(void);
void UART0_Transmit_Poll(uint8_t);
uint8_t UART0_Receive_Poll(void);

//mode1
void ADC0_init(void);
short int result1;
short int result2;
bool alt = true;

//mode2/5
void LED_init(void);
void Trigger_Timer2_init(void);
void Capture_Timer2_init(void);
void PWM_init25(void);
char buffer5[7] = {"0900\r\n"};
volatile uint32_t pulse_width;
volatile float distance;
int tmpInt1;
float tmpFrac;
int tmpInt2;
volatile uint32_t cont[2];
short int result;

```

```

char buffer[30];
#define mod 14999

//mode 3/4
void UART0Mode34_init(void);
void UART0Tx(char c);
void UART0_puts(char* s);
void PWM_init(void);
short int result;
short int prevresult = -1;
char buffer4[7] = {"0000\r\n"};
char buffer3[4] = {"1\r\n"};
int mode;
int number = 0;
int i = 0;
int i2 = 0;

int main(void) {

    BOARD_InitBootClocks();

    __disable_irq();
    UART0_init();
    keypad_init();
    PWM_init();
    LCD_init();
    __enable_irq();

    short int i = 0;
    bool up = true;

    while(1) {
        if(mode == 1){
            if(hexKeys[keypad_getkey()] == '#'){
                UART0_init();
                NVIC_DisableIRQ(15);    //Disable IRQ 15
            }
        }
        if(mode == 2){
            if(hexKeys[keypad_getkey()] == '#'){
                UART0_init();
            }
        }
        if(mode == 3){

```

```

        if(hexKeys[keypad_getkey()] == '#'){
            UART0_init();
        }

result = atoi(buffer3);    /* read conversion result and clear COCO flag */

if(prevresult != result){
    UART0_Transmit_Poll('S');
    UART0_Transmit_Poll('p');
    UART0_Transmit_Poll('e');
    UART0_Transmit_Poll('e');
    UART0_Transmit_Poll('d');
    UART0_Transmit_Poll(':');
    UART0_Transmit_Poll(' ');
    UART0_puts(buffer3);
}
prevresult = result;

if(result > 0 && result <= 4)
    delayMs(7/result);
else if(result <= 0)
    delayMs(7/1);
else if(result > 4)
    delayMs(7/4);

//servo angle
TPM0->CONTROLS[1].CnV = 1620 + i*(5940/1800);

if(up){
    if(i >= 1800){
        up = false;
    }
    i++;
}else{
    if(i <= 0){
        up = true;
    }
    i--;
}
}
if(mode == 4){
    if(hexKeys[keypad_getkey()] == '#'){
        UART0_init();
    }
    result = atoi(buffer4);    /* read conversion result and clear COCO flag */

    if(prevresult != result){

```

```

        UART0_Transmit_Poll('A');
        UART0_Transmit_Poll('n');
        UART0_Transmit_Poll('g');
        UART0_Transmit_Poll('l');
        UART0_Transmit_Poll('e');
        UART0_Transmit_Poll(':');
        UART0_Transmit_Poll(' ');

        UART0_puts(buffer4);
    }
    prevresult = result;
    delayMs(1000);
    //servo control
    TPM0->CONTROLS[1].CnV = 1620 + (result)*(5940/1800);
}

if(mode == 5){
    if(hexKeys[keypad_getkey()] == '#'){
        UART0_init();
    }

    result = atoi(buffer3);    /* read conversion result and clear COCO flag */

    if(prevresult != result){
        UART0_Transmit_Poll('S');
        UART0_Transmit_Poll('p');
        UART0_Transmit_Poll('e');
        UART0_Transmit_Poll('e');
        UART0_Transmit_Poll('d');
        UART0_Transmit_Poll(':');
        UART0_Transmit_Poll(' ');

        UART0_puts(buffer3);
    }

    prevresult = result;
    delayMs(1000);

    if(distance < (float)12.0)
        TPM0->CONTROLS[3].CnV = 0;
    else
        TPM0->CONTROLS[3].CnV = result*14000; /* Set up channel value between 0% -
93%*/
    }
}
return 0;
}

//keypad interrupt

```

```

void PORTD_IRQHandler(void){
    key = keypad_getkey();

    if(key != 0){
        keypressed = hexKeys[key];

        if(keypressed == '#'){
            LCD_command(1);
            LCD_command(0x84);
            LCD_data('M');
            LCD_data('o');
            LCD_data('d');
            LCD_data('e');
            LCD_data(':');
            LCD_data(' ');
            UART0_Transmit_Poll('M');
            UART0_Transmit_Poll('o');
            UART0_Transmit_Poll('d');
            UART0_Transmit_Poll('e');
            UART0_Transmit_Poll(':');
            UART0_Transmit_Poll(' ');
            modeSelect = true;
        }

        if(keypressed != '#' && modeSelect){
            if(keypressed == '1'){
                mode = 1;
                LCD_data(keypressed);
                UART0_Transmit_Poll(keypressed);
                LCD_command(0xC3);
                LCD_data('A');
                LCD_data('/');
                LCD_data('D');
                LCD_data(' ');
                LCD_data('S');
                LCD_data('E');
                LCD_data('R');
                LCD_data('V');
                LCD_data('O');

                ADC0_init();          /* Configure ADC0 */
                ADC0->SC1[0] = 0x40;
                PORTD->ISFR |= 0x00000020;
            }
            if(keypressed == '2'){
                mode = 2;
                LCD_data(keypressed);
                UART0_Transmit_Poll(keypressed);
            }
        }
    }
}

```

```

        LCD_command(0xC2);
        LCD_data('M');
        LCD_data('O');
        LCD_data('T');
        LCD_data('O');
        LCD_data('R');
        LCD_data(' ');
        LCD_data('S');
        LCD_data('E');
        LCD_data('N');
        LCD_data('S');
        LCD_data('O');
        LCD_data('R');

        LED_init();
        ADC0_init();          /* Configure ADC0 */
        PWM_init25();         /* Configure PWM on TPM0_CH3 */
        Trigger_Timer2_init(); /* Configure PWM on TPM2_CH0 */
        Capture_Timer2_init();
        ADC0->SC1[0] = 0x40;    //start conversion on channel 0
        PORTD->ISFR |= 0x00000020;
    }
    if(keypressed == '3'){
        mode = 3;
        LCD_data(keypressed);
        UART0_Transmit_Poll(keypressed);
        LCD_command(0xC2);
        LCD_data('S');
        LCD_data('E');
        LCD_data('R');
        LCD_data('V');
        LCD_data('O');
        LCD_data(' ');
        LCD_data('P');
        LCD_data('U');
        LCD_data('T');
        LCD_data('T');
        LCD_data('Y');
        LCD_data('3');

        UART0Mode34_init();
        PORTD->ISFR |= 0x00000020;
    }
    if(keypressed == '4'){
        mode = 4;
        LCD_data(keypressed);
        UART0_Transmit_Poll(keypressed);

```

```

        LCD_command(0xC2);
        LCD_data('S');
        LCD_data('E');
        LCD_data('R');
        LCD_data('V');
        LCD_data('O');
        LCD_data(' ');
        LCD_data('P');
        LCD_data('U');
        LCD_data('T');
        LCD_data('T');
        LCD_data('Y');
        LCD_data('4');

        UART0Mode34_init();
        PORTD->ISFR |= 0x00000020;
    }
    if(keypressed == '5'){
        mode = 5;
        LCD_data(keypressed);
        UART0_Transmit_Poll(keypressed);
        LCD_command(0xC1);
        LCD_data('M');
        LCD_data('O');
        LCD_data('T');
        LCD_data('O');
        LCD_data('R');
        LCD_data(' ');
        LCD_data('W');
        LCD_data('/');
        LCD_data(' ');
        LCD_data('P');
        LCD_data('U');
        LCD_data('T');
        LCD_data('T');
        LCD_data('Y');

        LED_init();           /* Configure LEDs */
        UART0Mode34_init();    /* initialize UART0 for output */
        PWM_init25();
        Trigger_Timer2_init();
        Capture_Timer2_init();
        PORTD->ISFR |= 0x00000020;
    }else{}

    modeSelect = false;

}

```



```

        delayMs(300);
    }
}

void UART0_IRQHandler(void) {
    char c;
    c = UART0->D;      /* read the char received */
    if(mode == 4){
        buffer4[i%4] = c;
        i++;
    } else if(mode == 3 || mode == 5){
        buffer3[0] = c;
    }
}

void ADC0_IRQHandler(void)
{
    if(mode == 1){
        if(alt){
            result1 = ADC0->R[0];      /* read conversion result and clear COCO flag */
        } else if (!alt){
            result2 = ADC0->R[0];
        }

        if (result1 < 1500 || result1 > 2485){
            TPM0->CONTROLS[1].CnV = 0;          //potentiometer loses control of
servo
        } else{
            TPM0->CONTROLS[1].CnV = 1500 +result2*3/2;          //potentiometer controls
servo
        }

        if(alt){
            ADC0->SC1[0] = 0x40;      /* start conversion on channel 0 */
        } else{
            ADC0->SC1[0] = 0x44;          //start conversion on channel 4
        }

        alt = !alt;
    }
    if(mode == 2){
        result1 = ADC0->R[0];          /* read conversion result and clear COCO flag */
        if(distance < (float)12.0){
            TPM0->CONTROLS[3].CnV = 0;

```

```

        }else{
            TPM0->CONTROLS[3].CnV = result1*14; /* For DC Motor: Set up channel value
between 0% - 93%*/

        }
        ADC0->SC1[0] = 0x40;
    }else{
        result1 = ADC0->R[0];
    }
}

void TPM0_IRQHandler()
{
    TPM0->CONTROLS[3].CnSC |= 0x80;                //Clear CHF
    TPM0->CONTROLS[3].CnV = 0;
    ADC0->SC1[0] &= ~0x1F;                        //start conversion on channel 0
    TPM0->SC |= 0x80;                             //clear TOF
}

void TPM2_IRQHandler(void) {
    while(!(TPM2->CONTROLS[1].CnSC & 0x80)) { } /* wait until the CHF is set */
    cont[i2%2] = TPM2->CONTROLS[1].CnV;
    if(i2%2 == 1){

        if(cont[1] > cont[0] ){
            pulse_width = cont[1] - cont[0];
        }
        else {
            pulse_width = cont[1] - cont[0] + mod + 1;
        }

        if(mode == 5){
            distance = (float)(pulse_width*2/3)*0.0343;

            tmpInt1 = distance;
            tmpFrac = distance - tmpInt1;
            tmpInt2 = (tmpFrac * 10000);

            distance = (float)(distance/2.54);

            tmpInt1 = distance;
            tmpFrac = distance - tmpInt1;
            tmpInt2 = (tmpFrac * 10000);
        }

        if(mode == 2){

```

```

        sprintf(buffer, "Pulse width %d \r\n", pulse_width); /* convert to string */
        UART0_puts(buffer);
        distance = (float)(pulse_width*2/3)*0.0343;

        tmpInt1 = distance;
        tmpFrac = distance - tmpInt1;
        tmpInt2 = (tmpFrac * 10000);

        sprintf(buffer, "Distance %d.%04d cm\r\n", tmpInt1, tmpInt2); /* convert to string */
        UART0_puts(buffer);

        distance = (float)(distance/2.54);

        tmpInt1 = distance;
        tmpFrac = distance - tmpInt1;
        tmpInt2 = (tmpFrac * 10000);

        sprintf(buffer, "Distance %d.%04d inches\r\n", tmpInt1, tmpInt2); /* convert to string */
        UART0_puts(buffer);
        ADC0->SC1[0] = 0x40;
    }

    if(distance < (float)12.0)
        PTB->PCOR |= 0x40000; /* Clear PTB18 to turn on red LED */
        //insert code for DC Motor stops until object out of range
    else
        PTB->PSOR |= 0x40000; /* Set PTB18 to turn off red LED */

    if(distance < (float)24.0 && distance > (float)12.0)
    {
        PTD->PCOR |= 0x2; /* Clear PTB18 to turn on blue LED */
    }
    else
        PTD->PSOR |= 0x2; /* Set PTB18 to turn off blue LED */

    }
    i++;
}
/*-----*/
TPM2->CONTROLS[1].CnSC |= 0x80; /* clear CHF */
TPM2->SC |= 0x80;
}

void PWM_init(void)
{
    SIM->SCGC5 |= 0x800; /* enable clock to Port C */
    PORTC->PCR[2] = 0x0400; /* PTC2 used by TPM0 */
    SIM->SCGC6 |= 0x01000000; /* enable clock to TPM0 */
    SIM->SOPT2 |= 0x01000000; /* use MCGFLLCLK as timer counter clock */
}

```

```

    TPM0->SC = 0; /* disable timer */
    TPM0->CONTROLS[1].CnSC |= TPM_CnSC_MSB_MASK | TPM_CnSC_ELSB_MASK; //Enable TPM0_CH1
as edge-aligned PWM
    TPM0->MOD = 60000; /* Set up modulo register for 50 Hz - 48.00 MHz */
    TPM0->CONTROLS[1].CnV = 1500; /* Set up channel value for 2.5% duty-cycle */
    TPM0->SC |= 0x0C; /* enable TPM0 with pre-scaler /16 */
}

```

```

void keypad_init(void)
{
    SIM->SCGC5 |= 0x01000; /* enable clock to Port D */
    PORTD->PCR[0] = 0x103; /* make PTD0 pin as GPIO and enable pullup*/
    PORTD->PCR[1] = 0x103; /* make PTD1 pin as GPIO and enable pullup*/
    PORTD->PCR[2] = 0x103; /* make PTD2 pin as GPIO and enable pullup*/
    PORTD->PCR[3] = 0x103; /* make PTD3 pin as GPIO and enable pullup*/
    PORTD->PCR[4] = 0x103; /* make PTD4 pin as GPIO and enable pullup*/
    PORTD->PCR[5] = 0xA0103; /* make PTD5 pin as GPIO and enable pullup and enable interrupt*/
    PORTD->PCR[6] = 0x103; /* make PTD6 pin as GPIO and enable pullup*/
    PORTD->PCR[7] = 0x103; /* make PTD7 pin as GPIO and enable pullup*/
    PTD->PDDR = 0x0F; /* make PTD7-4 as input pins, PTD3-0 as outputs */
    NVIC_EnableIRQ(31); //enable portD interrupt
}

```

```

char keypad_getkey(void)
{
    int row, col;
    const char row_select[] = {0x01, 0x02, 0x04, 0x08}; /* one row is active */

    /* check to see any key pressed */
    PTD->PDDR |= 0x0F; /* enable all rows */
    PTD->PCOR = 0x0F;
    delayUs(2); /* wait for signal return */
    col = PTD->PDIR & 0xF0; /* read all columns */
    PTD->PDDR = 0; /* disable all rows */
    if (col == 0xF0)
        return 0; /* no key pressed */

    /* If a key is pressed, it gets here to find out which key.
    * It activates one row at a time and read the input to see
    * which column is active. */
    for (row = 0; row < 4; row++)
    {
        PTD->PDDR = 0; /* disable all rows */
        PTD->PDDR |= row_select[row]; /* enable one row */
        PTD->PCOR = row_select[row]; /* drive the active row low */
        delayUs(2); /* wait for signal to settle */
        col = PTD->PDIR & 0xF0; /* read all columns */
    }
}

```

```

        if (col != 0xF0) break;      /* if one of the input is low, some key is pressed. */
    }
    PTD->PDDR = 0;                  /* disable all rows */
    if (row == 4)
        return 0;                  /* if we get here, no key is pressed */

    /* gets here when one of the rows has key pressed, check which column it is */
    if (col == 0xE0) return row * 4 + 1; /* key in column 0 */
    if (col == 0xD0) return row * 4 + 2; /* key in column 1 */
    if (col == 0xB0) return row * 4 + 3; /* key in column 2 */
    if (col == 0x70) return row * 4 + 4; /* key in column 3 */

    return 0; /* just to be safe */
}

void LCD_init(void)
{
    SIM->SCGC5 |= 0x800; /* enable clock to Port C */
    PORTC->PCR[8] = 0x100; /* make PTC8 pin as GPIO */
    PORTC->PCR[9] = 0x100; /* make PTC9 pin as GPIO */
    PORTC->PCR[10] = 0x100; /* make PTC10 pin as GPIO */
    PORTC->PCR[11] = 0x100; /* make PTC11 pin as GPIO */
    PORTC->PCR[12] = 0x100; /* make PTC12 pin as GPIO */
    PORTC->PCR[13] = 0x100; /* make PTC13 pin as GPIO */

    PTC->PDDR |= 0x3F00; /* make PTC 13-8 as output pins */

    delayMs(30); /* initialization sequence */
    LCD_nibble_write(0x30, 0);
    delayMs(10);
    LCD_nibble_write(0x30, 0);
    delayMs(1);
    LCD_nibble_write(0x30, 0);
    delayMs(1);
    LCD_nibble_write(0x20, 0); /* use 4-bit data mode */
    delayMs(1);

    LCD_command(0x28); /* set 4-bit data, 2-line, 5x7 font */
    LCD_command(0x06); /* move cursor right */
    LCD_command(0x01); /* clear screen, move cursor to home */
    LCD_command(0x0F); /* turn on display, cursor blinking */

    LCD_command(0x83);
        LCD_data('H');
        LCD_data('o');
        LCD_data('l');
        LCD_data('d');

```

```

        LCD_data(' ');
        LCD_data('#');
        LCD_data(' ');
        LCD_data('T');
        LCD_data('o');
        LCD_command(0xC2);
        LCD_data('S');
        LCD_data('e');
        LCD_data('l');
        LCD_data('e');
        LCD_data('c');
        LCD_data('t');
        LCD_data(' ');
        LCD_data('M');
        LCD_data('o');
        LCD_data('d');
        LCD_data('e');
    }

void LCD_nibble_write(unsigned char data, unsigned char control)
{
    data &= 0xF0;    /* clear lower nibble for control */
    control &= 0x0F; /* clear upper nibble for data */
    PTC->PDOR = ((data | control | EN) & 0xF0) << 6 | ((data|control|EN) & 0x04) << 7 | ((data|control|EN) & 0x01)
<< 8;    /* RS = 0, R/W = 0 */
    PTC->PDOR = data | control | EN; /* pulse E */
    delayMs(0);
    PTC->PDOR = (data & 0xF0) << 6;
    PTC->PDOR = 0;
}

void LCD_command(unsigned char command)
{
    LCD_nibble_write(command & 0xF0, 0); /* upper nibble first */
    LCD_nibble_write(command << 4, 0); /* then lower nibble */

    if (command < 4)
        delayMs(4);    /* commands 1 and 2 need up to 1.64ms */
    else
        delayMs(1);    /* all others 40 us */
}

void LCD_data(unsigned char data)
{
    LCD_nibble_write(data & 0xF0, RS); /* upper nibble first */
    LCD_nibble_write(data << 4, RS); /* then lower nibble */

    delayMs(1);

```

```

}

//clock speed 48 MHZ and 115200 baud
void UART0_init(void) {
    SIM->SCGC4 |= SIM_SCGC4_UART0(1);    //0x0400; /* enable clock for UART0 */
    SIM->SOPT2 |= SIM_SOPT2_UART0SRC(1);    //0x04000000 /* use FLL output for UART Baud rate
generator */
    UART0->C2 = 0;                        //0x00 /* turn off UART0 while changing configurations */
    UART0->BDH = UART0_BDH_SBR(0);        //0x00; /* SBR12:SBR8 = 0b000000 - 115200 Baud
    UART0->C4 = UART0_C4_OSR(15);        //0x0F; /* Over Sampling Ratio (15+1) */
//    UART0->BDL = UART0_BDL_SBR(137);    //0x89; /* SBR7:SBR0 = 0b10001001 - 9600
Baud - Clock = 20.97 MHz*/
//    UART0->BDL = UART0_BDL_SBR(12);    //0x0C; /* SBR7:SBR0 = 0b00001100 - 115200
Baud - Clock = 20.97 MHz*/
    UART0->BDL = UART0_BDL_SBR(26);        //0x1A; /* SBR7:SBR0 = 0b00011010 -
115200 Baud - Clock = 48.00 MHz */
    UART0->C1 = UART0_C1_M(0);            //0x00; /* 8-bit data */
    UART0->C2 = UART0_C2_TE(1)|UART0_C2_RE(1); //0x0C; /* enable transmit & Receive*/
    NVIC->ISER[0] &= ~0x00001000;        //disable interrupt
    SIM->SCGC5 |= SIM_SCGC5_PORTA(1);    //0x0200; /* enable clock for PORTA */
    PORTA->PCR[2] = PORT_PCR_MUX(2);    //0x0200; /* make PTA2 UART0_Tx pin */
    PORTA->PCR[1] = PORT_PCR_MUX(2);    //0x0200; /* make PTA1 UART0_Rx pin */

    SIM->SCGC6 |= 0x01000000;            /* enable clock to TPM0 */
    SIM->SOPT2 |= 0x01000000;            /* use MCGFLLCLK as timer counter clock */
    TPM0->SC = 0;
    TPM0->CONTROLS[3].CnSC |= TPM_CnSC_MSB_MASK | TPM_CnSC_ELSB_MASK;
    TPM0->CONTROLS[3].CnV = 0;
    NVIC_DisableIRQ(15);
    NVIC_DisableIRQ(17);
}

void ADC0_init(void)
{
    uint16_t calibration;

    SIM->SCGC5 |= 0x2000; /* clock to PORTE */
    PORTE->PCR[20] = 0; /* PTE20 analog input */
    PORTE->PCR[21] = 0; /*PTE21 analog input
    SIM->SCGC6 |= 0x8000000; /* clock to ADC0 */
    ADC0->SC2 &= ~0x40; /* software trigger */
    ADC0->SC1[0] |= 0x40;

    /* clock div by 4, long sample time, single ended 12 bit, bus clock */
    ADC0->CFG1 = 0x40 | 0x10 | 0x04 | 0x00;

    //Start Calibration
    ADC0->SC3 |= ADC_SC3_CAL_MASK;

```

```

while (ADC0->SC3 & ADC_SC3_CAL_MASK) {
// Wait for calibration to complete
}
// Finish off the calibration
// Initialize a 16-bit variable in RAM
calibration = 0x0;
// Add the plus-side calibration results to the variable
calibration += ADC0->CLP0;
calibration += ADC0->CLP1;
calibration += ADC0->CLP2;
calibration += ADC0->CLP3;
calibration += ADC0->CLP4;
calibration += ADC0->CLPS;
// Divide by two
calibration /= 2;
// Set the MSB of the variable
calibration |= 0x8000;
// Store the value in the plus-side gain calibration register
ADC0->PG = calibration;
// Repeat the procedure for the minus-side calibration value
calibration = 0x0000;
calibration += ADC0->CLM0;
calibration += ADC0->CLM1;
calibration += ADC0->CLM2;
calibration += ADC0->CLM3;
calibration += ADC0->CLM4;
calibration += ADC0->CLMS;
calibration /= 2;
calibration |= 0x8000;
ADC0->MG = calibration;
//Done Calibration

/* Reconfigure ADC0*/
/* clock div by 4, long sample time, single ended 12 bit, bus clock */
ADC0->CFG1 = 0x40 | 0x10 | 0x04 | 0x00;
NVIC_EnableIRQ(15);          //Enable IRQ 15
}

void UART0Mode34_init(void) {
SIM->SCGC4 |= 0x0400; /* enable clock for UART0 */
SIM->SOPT2 |= 0x04000000; /* use FLL output for UART Baud rate generator */
UART0->C2 = 0; /* turn off UART0 while changing configurations */
UART0->BDH = 0x00;
UART0->BDL = UART0_BDL_SBR(26); /* 115200 Baud - Using 48 MHz clock*/
UART0->C4 = 0x0F; /* Over Sampling Ratio 16 */
UART0->C1 = 0x00; /* 8-bit data */
UART0->C2 = 0x2C; /* enable receive, transmit and receive interrupt*/
NVIC->ISER[0] |= 0x00001000; /* enable INT12 (bit 12 of ISER[0]) */
}

```



```

SIM->SCGC5 |= 0x0200; /* enable clock for PORTA */
PORTA->PCR[1] = 0x0200; /* make PTA1 UART0_Rx pin */
PORTA->PCR[2] = 0x0200; /* make PTA1 UART0_Tx pin */
}

void LED_init(void) {
    SIM->SCGC5 |= 0x400; /* enable clock to Port B */
    SIM->SCGC5 |= 0x1000; /* enable clock to Port D */
    PORTB->PCR[18] = 0x100; /* make PTB18 pin as GPIO */
    PTB->PDDR |= 0x40000; /* make PTB18 as output pin */
    PORTD->PCR[1] = 0x100; /* make PTD1 pin as GPIO */
    PTD->PDDR |= 0x02; /* make PTD1 as output pin */
}

void PWM_init25(void)
{
    SIM->SCGC5 |= 0x800; /* enable clock to Port C */
    PORTC->PCR[4] = 0x0400; /* PTC4 used by TPM0 */
    SIM->SCGC6 |= 0x01000000; /* enable clock to TPM0 */
    SIM->SOPT2 |= 0x01000000; /* use MCGFLLCLK as timer counter clock */

    TPM0->SC = 0; /* disable timer */
    TPM0->CONTROLS[3].CnSC |= TPM_CnSC_MSB_MASK | TPM_CnSC_ELSB_MASK; //Enable TPM0_CH1
as edge-aligned PWM
    TPM0->MOD = 60000; /* Set up modulo register for 50 Hz - 48.00 MHz */
    TPM0->CONTROLS[3].CnV = 1500; /* Set up channel value for 2.5% duty-cycle */
    TPM0->SC |= 0x0C; /* enable TPM0 with pre-scaler /16 */

    if(mode == 2)
        NVIC_EnableIRQ(17); /*Enable IRQ 17
}

void Trigger_Timer2_init(void)
{
    SIM->SCGC5 |= 0x02000; /*enable clock to Port E
    SIM->SCGC6 |= 0x04000000; /*enable clock to TPM2
    SIM->SOPT2 |= 0x01000000; /*use MCGFLLCLK as timer counter clock
    //PORTC->PCR[2] = 0x0400; /* PTC2 used by TPM0_CH1
    PORTE->PCR[22] = 0x0300; /*PTE22 used by TPM2_CH0

    TPM2->SC = 0; /*disable timer
    TPM2->CONTROLS[0].CnSC = 0x80; /*clear CHF for Channel 2
    TPM2->CONTROLS[0].CnSC |= 0x20|0x08; //edge-aligned, pulse high MSB:MSA=10, ELSB:ELSA=10
    TPM2->CONTROLS[0].CnV = 8; /*Set up channel value for >10 us
    TPM2->SC |= 0x06; /*set timer with prescaler /64
    TPM2->MOD = mod; /*Set up modulo register = (44),14999
//*****PRE-Scaler settings *****
    TPM2->SC |= 0x08; /*enable timer

```

```

/*****
*/
}

void Capture_Timer2_init(void) // Also enables the TPM2_CH1 interrupt
{
    SIM->SCGC5 |= 0x2000;    /* enable clock to Port E */
    SIM->SCGC6 |= 0x04000000; /* enable clock to TPM2 */
    PORTE->PCR[23] = 0x0300; /* PTE23 used by TPM2_CH1 */
    SIM->SOPT2 |= 0x01000000; /* use MCGFLLCLK as timer counter clock */
    TPM2->SC = 0;            /* disable timer */

    TPM2->CONTROLS[1].CnSC = 0x80; /* clear CHF for Channel 1 */
// MSB:MSA=00, ELSB:ELSA=11 and set interrupt*/
/* capture on both edges, MSB:MSA=00, ELSB:ELSA=11 and set interrupt*/
    TPM2->CONTROLS[1].CnSC |=
TPM_CnSC_CHIE_MASK|TPM_CnSC_ELSB_MASK|TPM_CnSC_ELSA_MASK;
    TPM2->MOD = mod;          /* Set up modulo register = 44999 */
    TPM2->CONTROLS[1].CnV = (mod+1)/2 -1; /* Set up 50% dutycycle */

    TPM2->SC |= 0x80;         /* clear TOF */
    TPM2->SC |= 0x06;         /* enable timer with prescaler /2^6 = 64 */
/*****PRE-Scaler settings *****/
    TPM2->SC |= 0x08;         /* enable timer */
/*****
NVIC_EnableIRQ(TPM2_IRQn); /* enable Timer2 interrupt in NVIC */
}

void UART0_Transmit_Poll(uint8_t data) {
    while (!(UART0->S1 & UART_S1_TDRE_MASK)); // wait until transmit data register is empty
    TDRE_Mask is 0x80
    UART0->D = data;
}

uint8_t UART0_Receive_Poll(void) {
    while (!(UART0->S1 & UART_S1_RDRF_MASK)); // wait until receive data register is full RDRF_Mask
    is 0x20
    return UART0->D;
}

void UART0Tx(char c) {
    while(!(UART0->S1 & 0x80)) {
    } /* wait for transmit buffer empty */
    UART0->D = c; /* send a char */
}

void UART0_puts(char* s) {
    while (*s != 0) /* if not end of string */
        UART0Tx(*s++); /* send the character through UART0 */
}

```

```

void delayMs(int n) {
    int i, j;
    for(i = 0 ; i < n; i++)
        for (j = 0; j < 3500; j++) {}
}

```

```

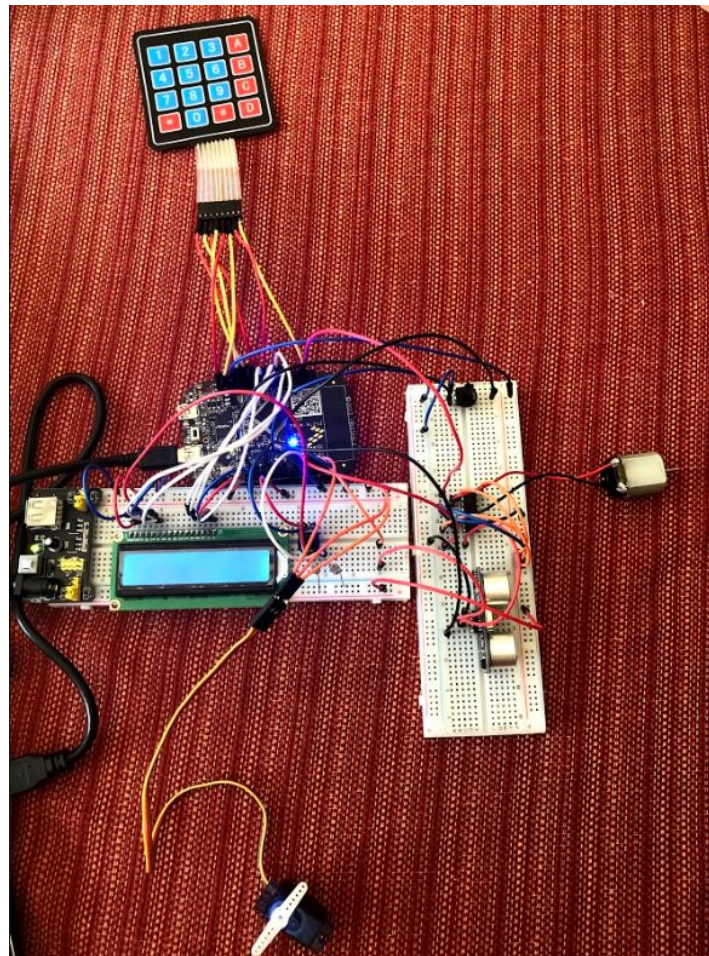
void delayUs(int n){
    int i; int j;
    for(i = 0 ; i < n; i++) {
        for(j = 0; j < 5; j++) ;
    }
}

```

## **Pictures**

Picture 1 shows the entire configuration of the board and peripherals used.

Picture 1



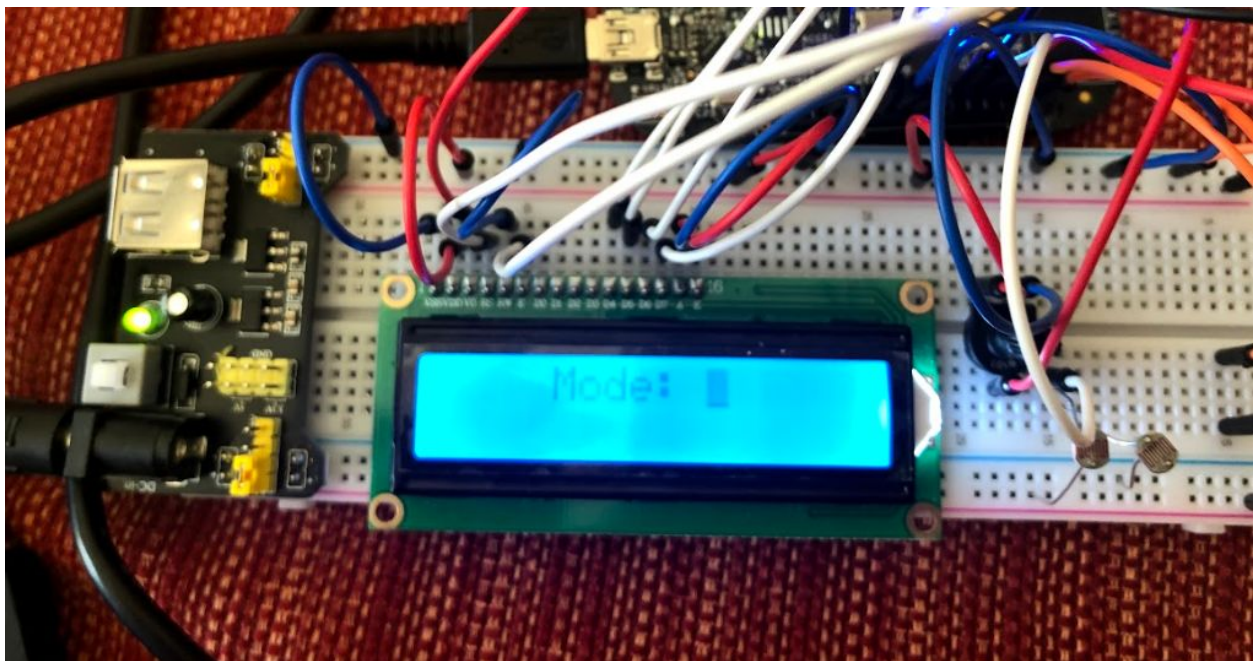
Picture 2 shows the LCD display before any mode is selected. The LCD instructs the user which button allows for mode selection.

Picture 2



Picture 3 shows the LCD when '#' has been pressed but no mode has been selected.

Picture 3



Picture 4 shows the PuTTY output when the key '#' has been pushed but no mode has been selected.



Picture 4



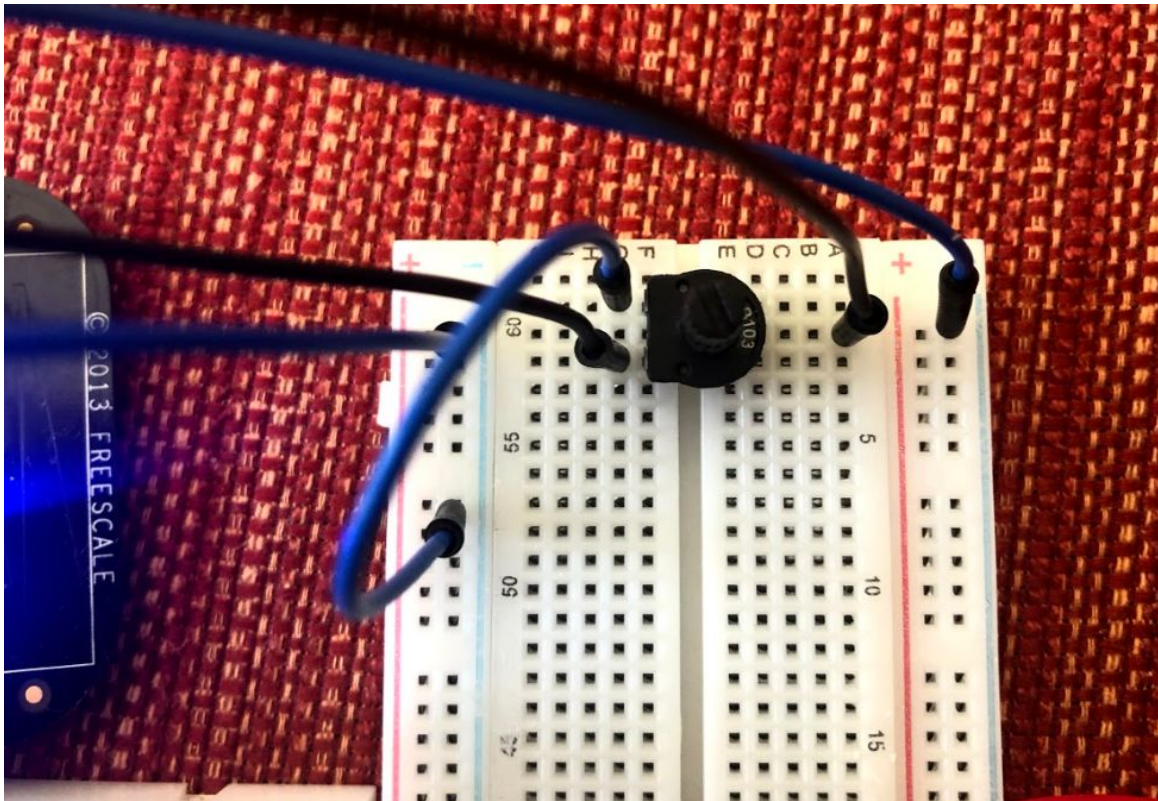
Picture 5 shows the LCD display when mode 1 was selected.

Picture 5



Picture 6 shows the potentiometer setup used in modes 1 and 2 where the inputs are ground on the top and 3.3V on the bottom and the output leads to pin E20 for Analog to Digital Conversion.

Picture 6



Picture 7 shows the PuTTY output when mode 1 is selected.

Picture 7

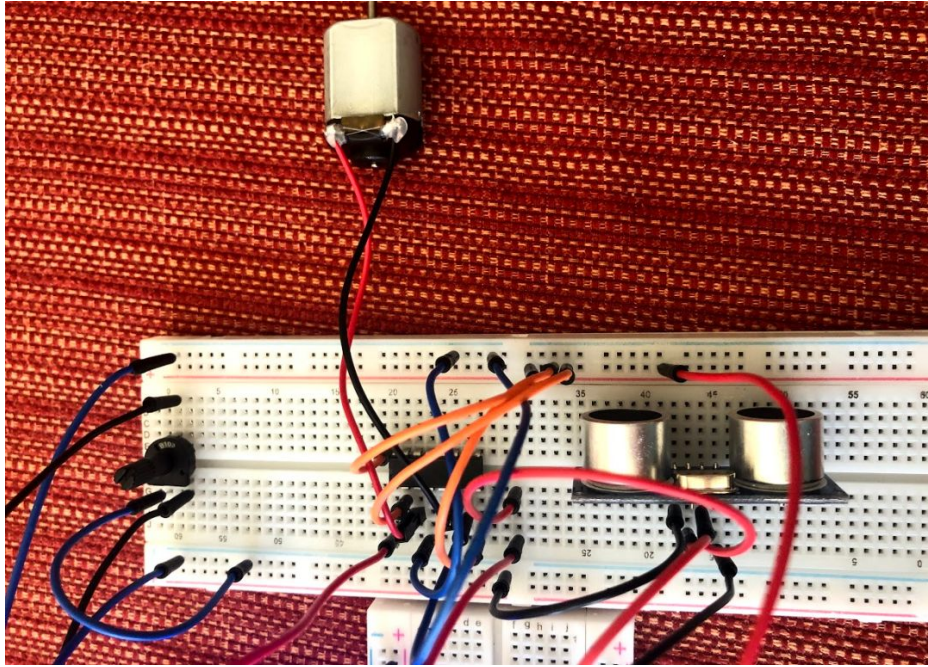


Picture 8 shows the potentiometer, L293D Chip, DC motor, and Ultrasonic Sensor setup used in modes 2 and 5. The input for the L293D Chip is from the PWM setup on pin C4. The motor has



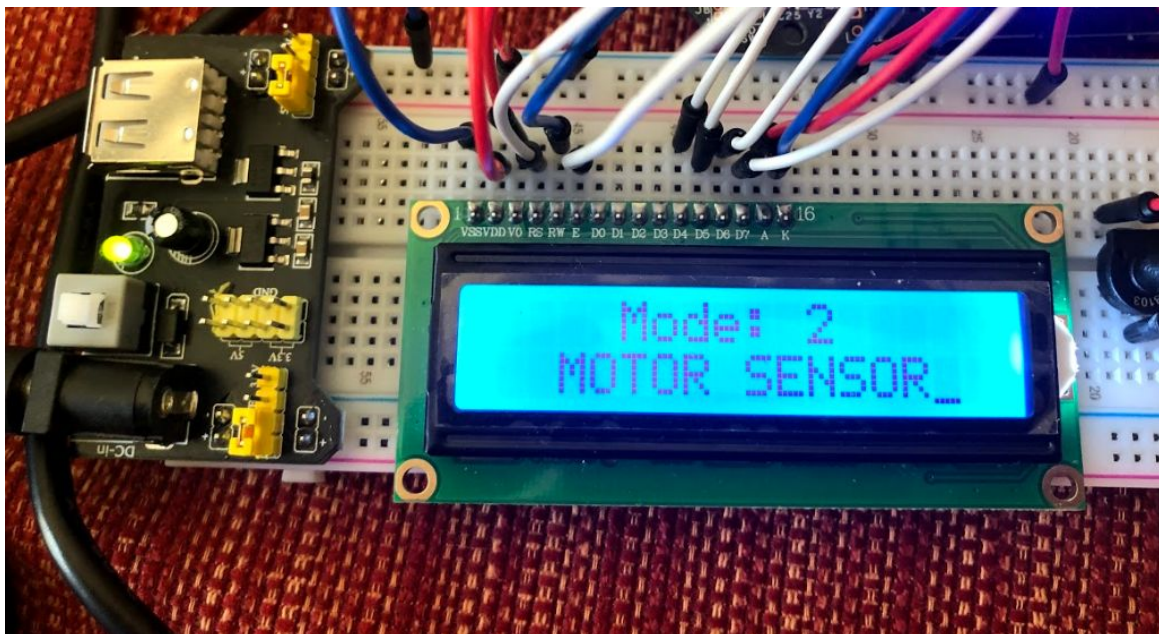
its positive terminal on pin 3 and the negative terminal on pin 6 of the L293D Chip. This is for the motor to spin clockwise. The ADC was taken from the potentiometer as seen in mode 1. The Ultrasonic Sensor has the TRIG Pin connected to pin E22 and the ECHO pin connected to pin E23 on the FRDM board.

Picture 8



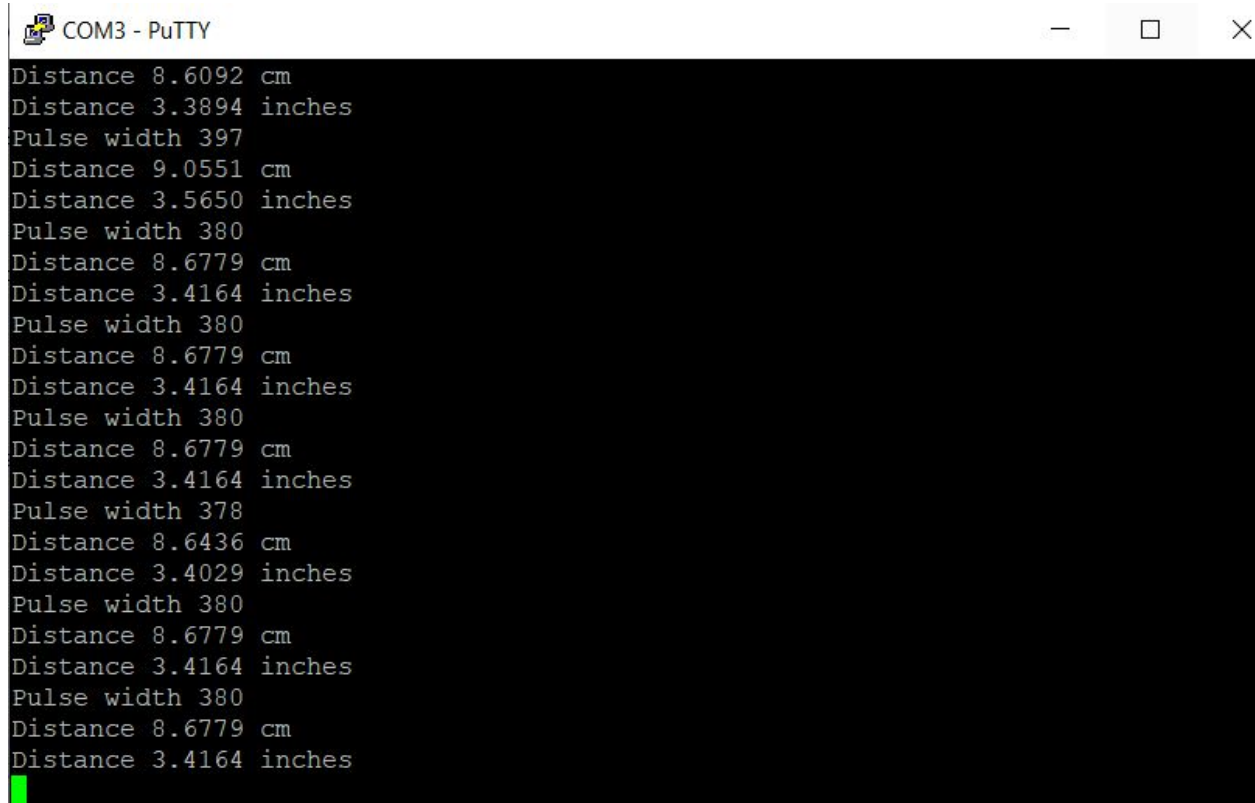
Picture 9 shows the LCD display when mode 2 is selected.

Picture 9



Picture 10 shows the output from mode 2 on PuTTY. The outputs are the values of the calculated pulse width and distance in both centimeters and inches.

Picture 10

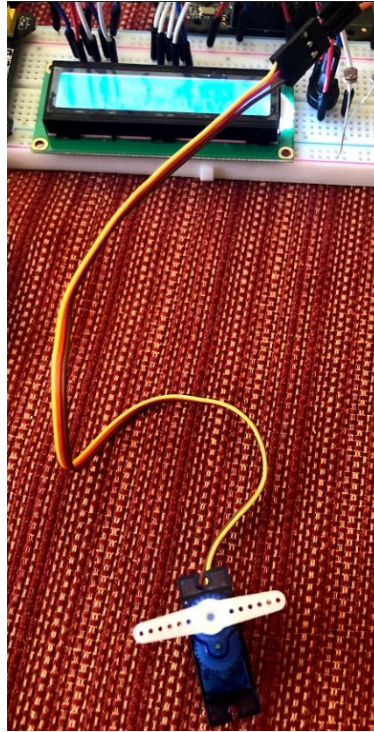


```
COM3 - PuTTY
Distance 8.6092 cm
Distance 3.3894 inches
Pulse width 397
Distance 9.0551 cm
Distance 3.5650 inches
Pulse width 380
Distance 8.6779 cm
Distance 3.4164 inches
Pulse width 380
Distance 8.6779 cm
Distance 3.4164 inches
Pulse width 380
Distance 8.6779 cm
Distance 3.4164 inches
Pulse width 378
Distance 8.6436 cm
Distance 3.4029 inches
Pulse width 380
Distance 8.6779 cm
Distance 3.4164 inches
Pulse width 380
Distance 8.6779 cm
Distance 3.4164 inches
```



Picture 11 shows a close view of the servo motor used in modes 1,3 and 4. The yellow connection is the PWM input from pin C2 and the red and brown connections are 5V and ground respectively.

Picture 11



Picture 12 shows the LCD display when mode 3 is selected.

Picture 12



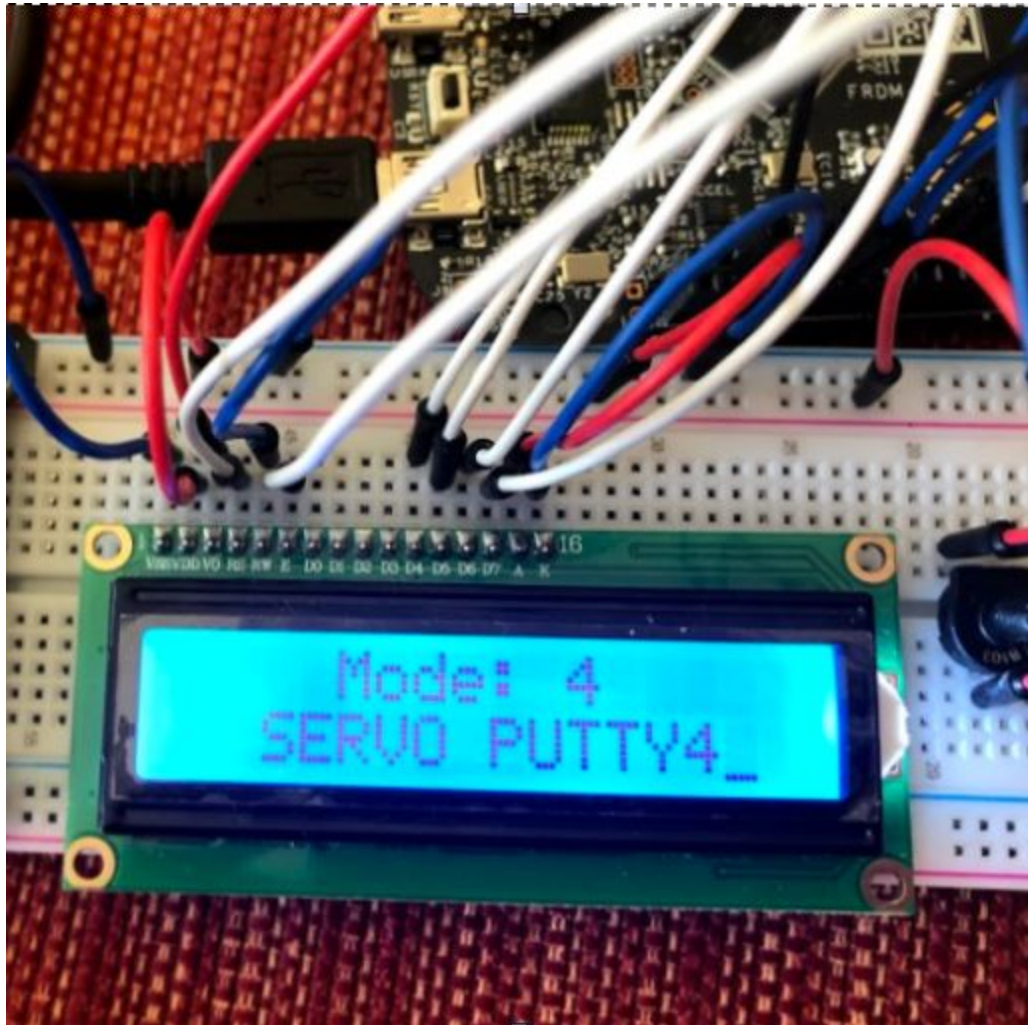
Picture 13 shows the PuTTY display when mode 3 is selected. PuTTY allows the user to enter a speed as input and that speed is shown and converted to a PWM signal for the servo motor.

Picture 13



Picture 14 shows the LCD display when mode 4 is selected.

Picture 14



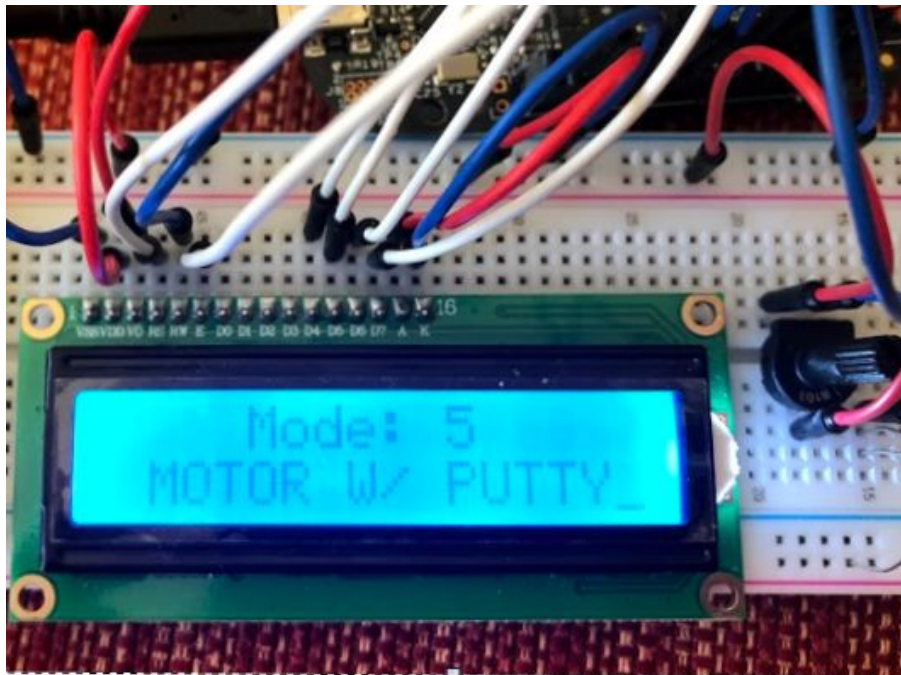
Picture 15 shows the PuTTY display when mode 4 is selected. Mode 4 allows the user to enter a 4 digit number and converts that input to the angle the servo motor is turned to. The input 1800 is the angle 180.0°.

Picture 15



Picture 16 shows the LCD display when mode 5 is selected.

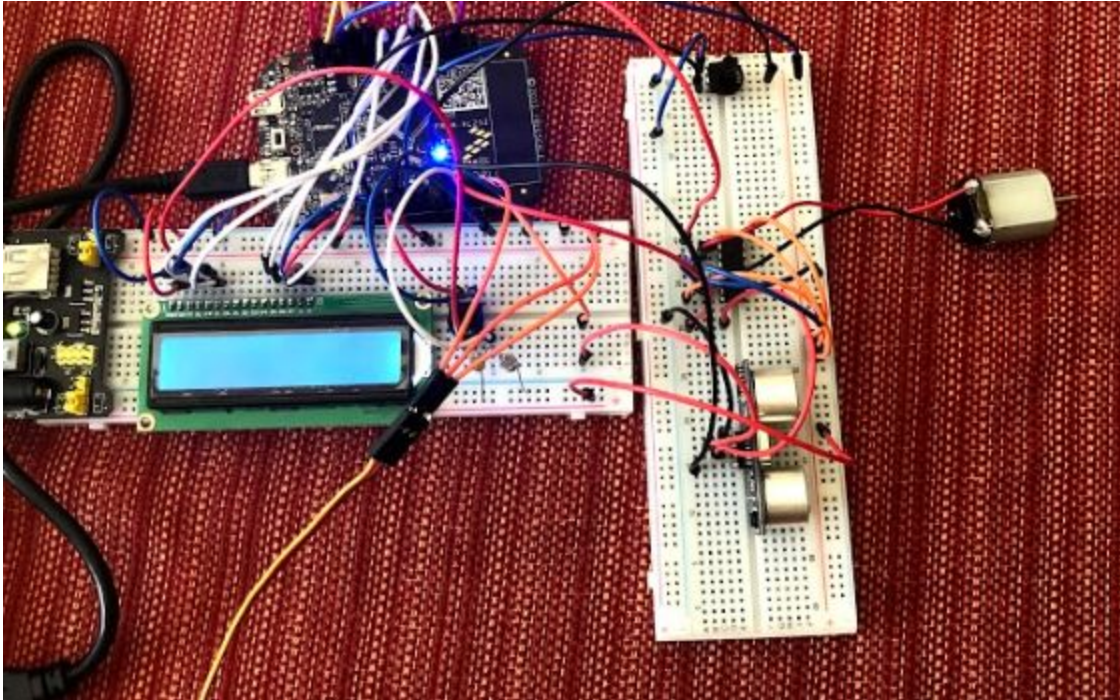
Picture 16





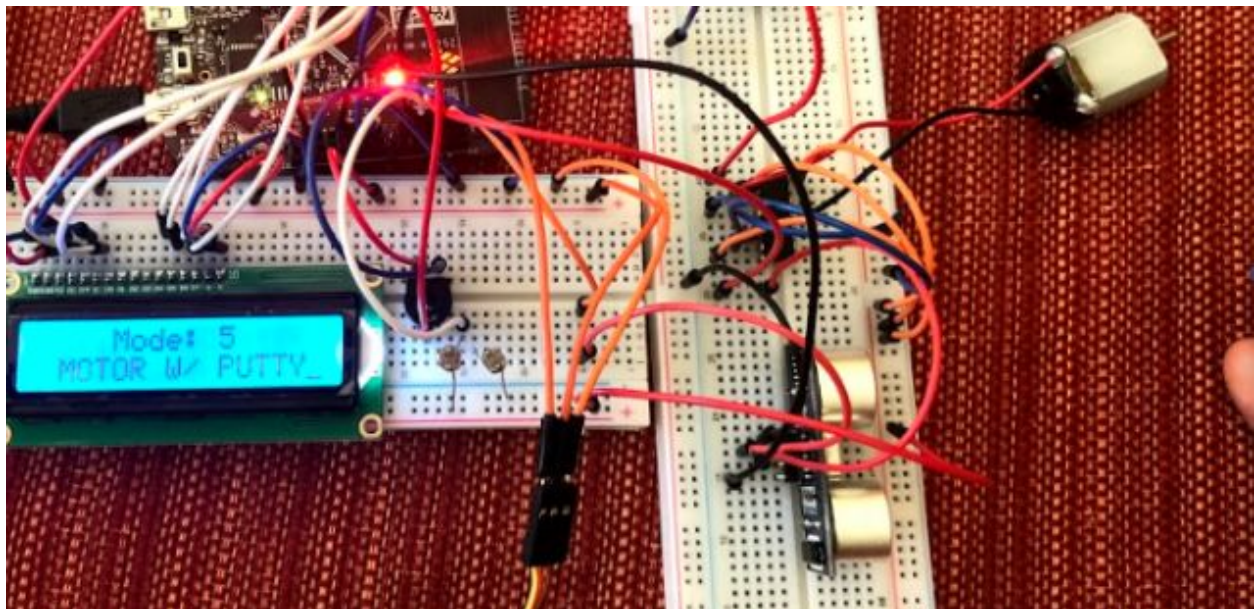
Picture 17 shows the blue LED engaged when an object is within 2 feet of the sensor when used in modes 2 and 5.

Picture 17



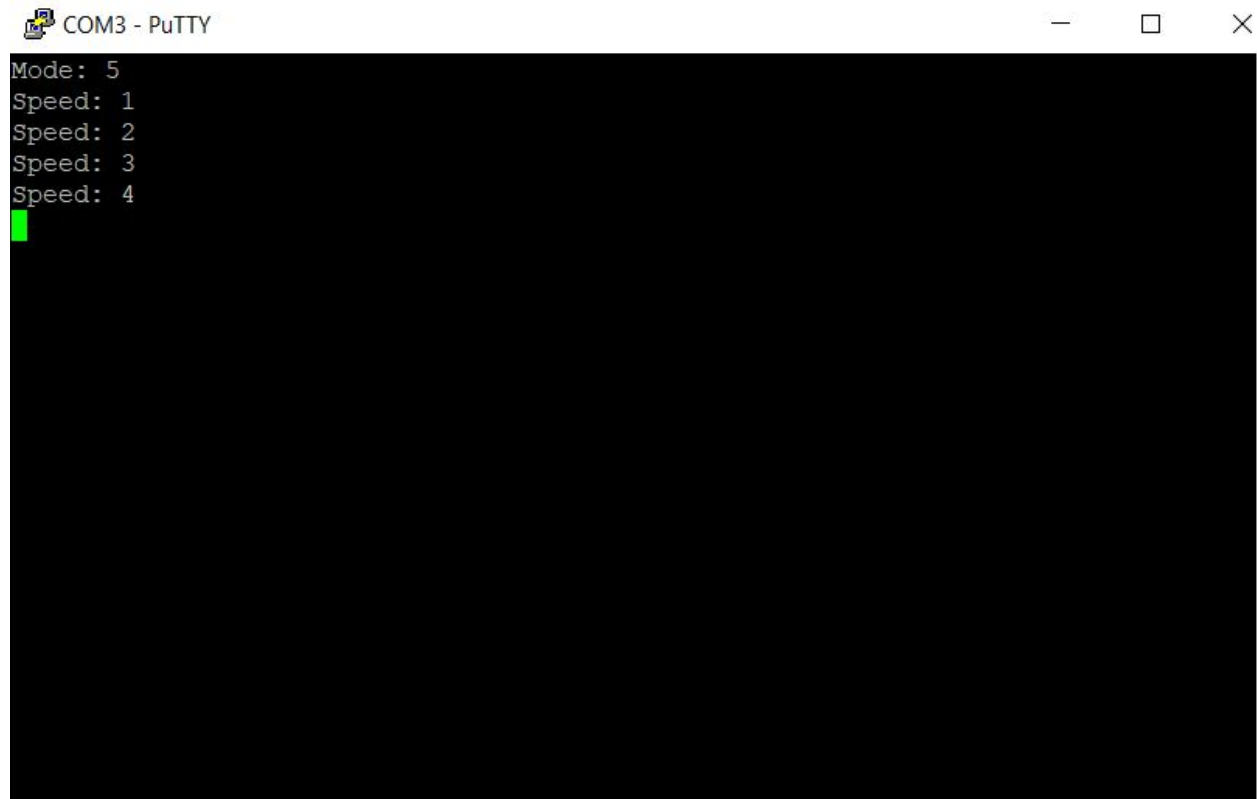
Picture 18 shows the red LED engaged when an object is within 1 foot of the sensor used in modes 2 and 5.

Picture 18



Picture 19 shows the PuTTY display when mode 5 is selected. The user selects the DC motor speed similar to mode 3.

Picture 19



```
COM3 - PuTTY
Mode: 5
Speed: 1
Speed: 2
Speed: 3
Speed: 4
█
```

## **Conclusion**

This report has discussed our design of an interrupt driven system based on the project guidelines given. The objectives of the project were to use various peripherals such as the LCD, DC Motor, Ultrasonic sensor, Servo motor, and passive buzzer and to control these peripherals using both hardware and software. Analog to digital conversion (A/D) and serial communication were also used in the project. Most of the criteria in the requirement checklist were satisfied with the exception being the use of the passive buzzer in modes two and five. More importantly, this comprehensive final project led us to realize the importance of the design process.