University of Nevada Las Vegas. Department of Electrical and Computer Engineering Laboratories.

Class:	CPE301L – 1001	Semester:	Spring 2023
Points		Document author:	Dylan Flores, Andy Lee
		Author's email:	flored16@unlv.nevada.edu leea76@unlv.nevada.edu
		Document topic:	Final Project
Instructor's comments:			

1. Introduction / Theory of Operation

This project implements a gate where the LCD interacts with the user through its encrypted messaging and the keypad which the user will only interact with physically. The AVR microcontroller will communicate with the PCB board to open the gate (move the servo motor from an angle of 0 degrees horizontally to the wires to a 90 degree angle indicating that the "gate is opened", illustrated by the servo motor. The proximity sensor will dictate whether the gate should open after the correct passcode from the user is entered and will NOT open until whatever is in front of the gate that is too close to it moves out of the way. Proximity sensor will detect if any object is near the gate so it does not close when it is open and the keypad on the PCB board will take inputs from the user; also, the LCD connected through SPI connection will display the following sequence of numbers. When the following sequence of numbers is entered correctly, the motor will turn on through a specific frequency and it will move for a period of time until the "gate is open", and then the proximity sensor will continuously detect to see if anything is in between the gate so it does not close onto the object itself. Afterwards, a timer will count until a number is hit and then the gate will close back down with the same resulting PWM frequency.

Components:

Proximity Distance Sensor - Measure object in way of motor gate

LCD display - Display ASCII characters for the passcode entry to see if passcode is correct

Servo-Motor - Actuator used to manually move the gate open

Keypad - Input for user to enter password

ATMega328P uC - Microcontroller to directly code all of these sensors to work together correctly

2. Final Project C Code

```
#define F_CPU 8000000UL
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <avr/io.h>
```

```
#include <util/delay.h>
#include <avr/interrupt.h>
#include <string.h>

#define SERVO_PIN PB3 // define the pin the servo is connected
to
#define TRIGGER_PIN  PB2
#define ECHO_PIN    PB0
#define rs PB5
#define rw PB6
#define en PB7
#define data PORTD
#define keypad PORTC
#define SOUND_SPEED 343 // speed of sound in m/s
#define empty 255
#define gate_closed -78
#define gate_opened -67
//functions necessary
void double_to_string(double, char *);
void gate_init(void);
void lcd_init(); //to initialize lcd
void lcd_cmd(char cmd_out); //to send command to lcd
void lcd_data(char data_out); //to send data to lcd
void lcd_str(const char *str); //to send string,, basically
stripping each character and sending
void opened();
uint8_t motor(double);
void motor_init(void);
double sensor();
int TimerOverflow = 0;
uint8_t check_col1();
uint8_t check_col2();
uint8_t check_col3();
void set_servo_angle(int);
uint8_t i = 0;
uint8_t num = 255;
long count;
uint16_t passcode[4];

// Converting numerical value into ASCII to print onto LCD
void double_to_string(double num, char *str) {
     // Check if the number is negative
     if (num < 0) {
          num = -num;
          *str++ = '-';
     }
     // Extract the integer part
     int int_part = (int)num;
     int digits = 1;
     while (int_part / digits > 9) {
          digits *= 10;
     }
     while (digits > 0) {
          *str++ = '0' + (int_part / digits);
          int_part %= digits;
          digits /= 10;
     }

     // Add the decimal point
     *str++ = '.';

     // Extract the fractional part
     double frac_part = num - (double)(int)num;
     for (int i = 0; i < 5; i++) { // Output up to 5 decimal
places
          frac_part *= 10;
          int digit = (int)frac_part;
          *str++ = '0' + digit;
          frac_part -= (double)digit;
     }

     // Terminate the string
     *str = '\0';
}

void gate_init(void){
     // passcode data init
     passcode[0] = 255;
     passcode[1] = 255;
     passcode[2] = 255;
     passcode[3] = 255;

     // Keypad init
     DDRC = 0b00000111;
     keypad = 0xFF;
     lcd_init();
     _delay_ms(100);

     lcd_cmd(0x01); // Clear display
     lcd_cmd(0x02); // Reset cursor at start of line
     motor_init();
}
// Main program
```

```
int main(){
    gate_init();
    while(1){
        // Get inputs from keypad
        passcode[i] = check_col1();
        passcode[i] = check_col2();
        passcode[i] = check_col3();
        if (passcode[0] != empty && passcode[1] == empty &&
passcode[2] == empty && passcode[3] == empty && i == 0){
            num = empty;
            i++;
        }
        else if (passcode[0] != empty && passcode[1] != empty
&& passcode[2] == empty && passcode[3] == empty && i == 1){
            i++;
            num = empty;
        }
        else if (passcode[0] != empty && passcode[1] != empty
&& passcode[2] != empty && passcode[3] == empty && i == 2){
            i++;
            num = empty;
        }
        else if (passcode[0] != empty && passcode[1] != empty
&& passcode[2] != empty && passcode[3] != empty && i == 3){
            i++;
            num = empty;
        }
        if(i == 4 && (passcode[0] == 7 && passcode[1] == 6 &&
passcode[2] == 9 && passcode[3] == 1))
        {
            lcd_cmd(0x02); // Reset cursor at start of line
            lcd_str("Opened");
            _delay_ms(100);
            opened();
            num = empty;
            passcode[0] = empty;
            passcode[1] = empty;
            passcode[2] = empty;
            passcode[3] = empty;
            i = 0;
            lcd_cmd(0x01);
            lcd_cmd(0x02);
            lcd_str("Proceed through!");
            _delay_ms(1000);
            lcd_cmd(0xC0);
            lcd_str("Gate now closing...");
            set_servo_angle(gate_closed);
```

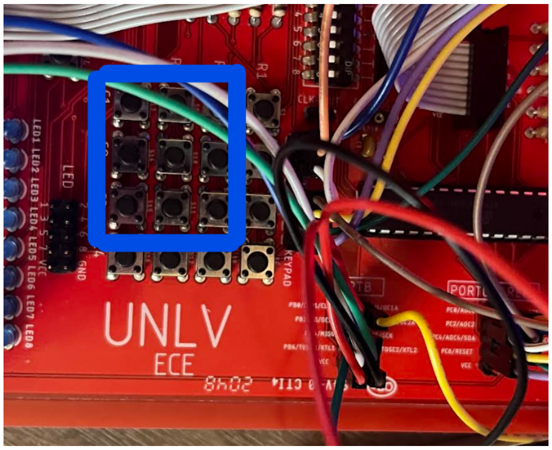
```
               _delay_ms(500);
               lcd_cmd(0x01);
          }
          else if( (i == 4 && (passcode[0] != 7 || passcode[1]
!= 6 || passcode[2] != 9 || passcode[3] != 1)) )
          {
               lcd_cmd(0xC0);
               lcd_str("Incorrect");
               _delay_ms(500);
               lcd_cmd(0x01); // Clear display
               num = empty;
               passcode[0] = empty;
               passcode[1] = empty;
               passcode[2] = empty;
               passcode[3] = empty;
               i = 0;
               lcd_cmd(0x01);
          }
     }
}

// Correct passcode triggers this event
void opened(){
     char printDistance[8];
     double distance;
     uint8_t done = 0;
     // Keep the gate closed if an obstacle is too close to the
sensor/gate and continue checking until the gate finally opens.
     while(done != 1){
          distance = sensor();
          lcd_cmd(0x01);
          lcd_cmd(0x02);
          double_to_string(distance, printDistance);
          lcd_str(printDistance);
          _delay_ms(100);
          done = motor(distance);
     }
}

// Setup wirings for servo-motor
void motor_init() {
     // Set timer mode to Fast PWM with TOP = ICR1
     TCCR1A |= (1 << WGM11) | (1 << COM1A1);
     TCCR1B |= (1 << WGM13) | (1 << WGM12) | (1 << CS11);
     ICR1 = 40000; // TOP value for 50Hz PWM frequency
     DDRB |= (1 << 1); // Set PB1 as output
     set_servo_angle(gate_closed);
```

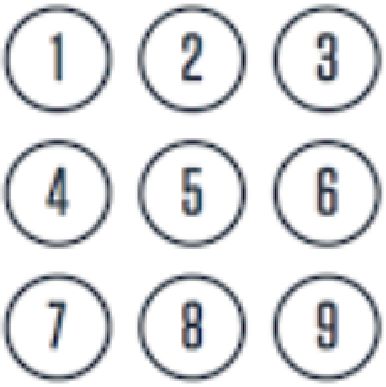
```
}

// Motor functions that determine a specific distance at 2
inches is too close to the gate, keep the gate closed so the
gate does not hit the obstacle or open otherwise.
uint8_t motor(double distance){
     uint8_t done = 0;
     // Gate stays closed
     if(distance < 2) {
          _delay_ms(10);
          set_servo_angle(gate_closed);      //      call      the
set_servo_angle function with the desired angle
          done = 0;
          lcd_cmd(0xC0);
          lcd_str("Gate too close!");
     }
     // Gate opens because gate's obstacles are clear
     else{
          set_servo_angle(gate_opened);      //      call      the
set_servo_angle function with the desired angle
          lcd_cmd(0xC0);
          lcd_str("Gate opening!");
          _delay_ms(10);
          done = 1;
     }
     _delay_ms(100);
     return done;
}

// Proximity sensor that detects the distance of an obstacle to
be used for the gate's functionality.
double sensor(void){
     DDRB |= (1 << TRIGGER_PIN); // set the TRIGGER_PIN as an
output
     DDRB &= ~(1 << ECHO_PIN); // set the ECHO_PIN as an input
     PORTB &= ~(1 << TRIGGER_PIN); // set the TRIGGER_PIN low
     double distance;

     // send a 10us pulse to the trigger pin
     PORTB |= (1 << TRIGGER_PIN);
     _delay_us(10);
     PORTB &= ~(1 << TRIGGER_PIN);

     // measure the duration of the pulse on the echo pin
     unsigned long duration = 0;
     while ((PINB & (1 << ECHO_PIN)) == 0);
     while ((PINB & (1 << ECHO_PIN)) != 0)
     {
          duration++;
          _delay_us(1);
     }

     // convert the duration to distance
     distance = duration * SOUND_SPEED / 20000.0;
     return distance;
}

// Function to move the servo-motor
void set_servo_angle(int angle) {
     OCR1A = (angle * 11.11) + 1000; // calculate and set the
duty cycle based on the desired angle
     _delay_ms(10); // wait for the servo to move to the desired
angle
}

// Initializing all lcd functions
void lcd_init(void){
     //set DDR LCD init
     DDRD = 0xFF;
     DDRB  |= (1 << rs) | (1 << en) | (1 << rw);
     _delay_ms(1);
     lcd_cmd(0x30);
     _delay_ms(1);
     lcd_cmd(0x30);
     _delay_ms(1);
     lcd_cmd(0x30);
     // Initializing to 2 lines & 5x7 font
     _delay_ms(1);
     lcd_cmd(0x38);
     _delay_ms(1);

     // Display on, cursor on
     lcd_cmd(0x0E);
     _delay_ms(1);
     // Clear LCD
     lcd_cmd(0x01);
     _delay_ms(1);
     // Set cursor position to top row 0x80
     lcd_cmd(0x80);
     _delay_ms(1);
}

// LCD command function
void lcd_cmd(char cmd_out){
     //send the cmd_out to data
     data = cmd_out;
     //set rs = 0 ,rw=0 and en =1
     PORTB &= ~(1 << rs);
     PORTB &= ~(1 << rw);
     PORTB |= (1 << en);

     //wait for small delay 1ms
     _delay_ms(1);
     //set rs = 0 ,rw=0 and en =0
     PORTB &= ~(1 << rs);
     PORTB &= ~(1 << en);
     PORTB &= ~(1 << rw);

     //wait for small delay 1ms
     _delay_ms(1);
}

// Printing out chars in the LCD
void lcd_data(char data_out) {
     //send the data_out to data
     data= data_out;
     //set rs = ? ,rw=? and en =?
     PORTB |= (1 << rs);
     PORTB |= (1 << en);
     PORTB &= ~(1 << rw);

     //wait for small delay 1ms
     _delay_ms(1);

     //set rs = ? ,rw=? and en =?
     PORTB &= ~(1 << en);
     //wait for small delay 1ms
     _delay_ms(1);

}

// Simplified version to print a string than separate chars
void lcd_str(const char *str){
     unsigned int i=0;
     while(str[i]!='\0'){
          lcd_data(str[i]);
          i++;
     }
}

/*
```

```
Key pad formulas using a 3x3 basic keypad from 0-2, 3-5, 6-9
from top to bottom
*/
uint8_t check_col1() {
     uint16_t mask = 0b01111001;
     keypad = mask; // Column
     _delay_ms(10); // Wait for I/O state to change
     // For each keypad press in target column, check which row
was pressed
     if((PINC & mask) == 0b00001001)
     {
          lcd_str("1");
          num = 1;
     }
     else if((PINC & mask) == 0b00010001)
     {
          lcd_str("4");
          num = 4;
     }
     else if((PINC & mask) == 0b00100001)
     {
          lcd_str("7");
          num = 7;
     }
     else if((PINC & mask) == 0b01000001)
     {
          lcd_str("*");
     }
     return num;
}

uint8_t check_col2()
{
     uint16_t mask = 0b01111010;
     keypad = mask; // Column
     _delay_ms(10); // Wait for I/O state to change
     // For each keypad press in target column, check which row
was pressed
     if((PINC & mask) == 0b00001010)
     {
          lcd_str("2");
          num = 2;
     }
     else if((PINC & mask) == 0b00010010)
     {
          lcd_str("5");
          num = 5;
```

```
    }
    else if((PINC & mask) == 0b00100010)
    {
        lcd_str("8");
        num = 8;
    }
    else if((PINC & mask) == 0b01000010)
    {
        lcd_str("0");
        num = 0;
    }
    return num;
}
uint8_t check_col3()
{
    uint16_t mask = 0b1111100;
    keypad = mask; // Column
    _delay_ms(10); // Wait for I/O state to change
    // For each keypad press in target column, check which row was pressed
    if((PINC & mask) == 0b0001100)
    {
        lcd_str("3");
        num = 3;
    }
    else if((PINC & mask) == 0b00010100)
    {
        lcd_str("6");
        num = 6;
    }
    else if((PINC & mask) == 0b00100100)
    {
        lcd_str("9");
        num = 9;
    }
    else if((PINC & mask) == 0b01000100)
    {
        lcd_cmd(0x01); // Clear display;
    }
    return num;
}
```

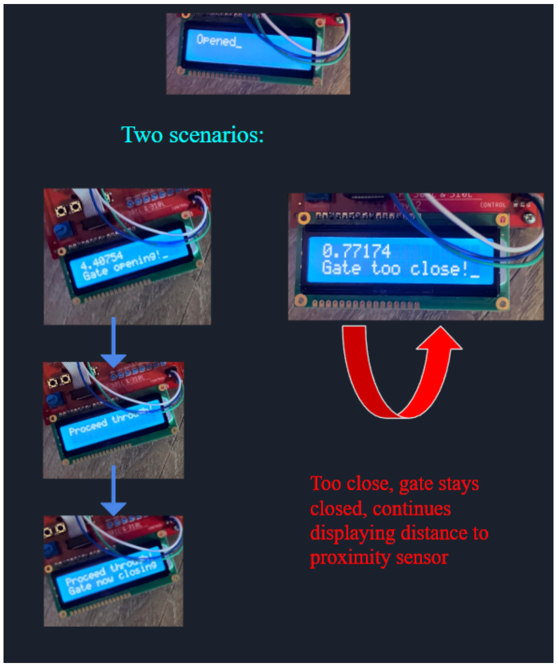
Keypad:





User interfaces a 3x3 basic keypad code to enter into the program and the columns and rows will be compared to the bit masking set on PORTC.

LCD:

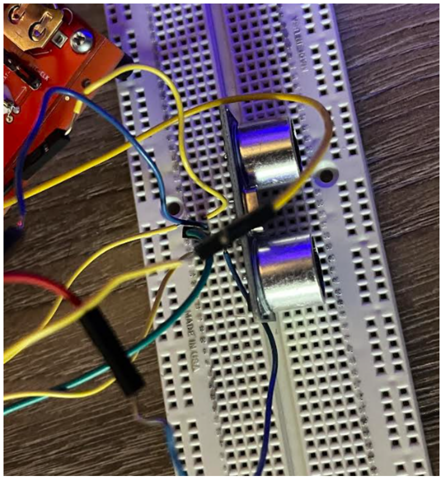


Two scenarios:

Too close, gate stays closed, continues displaying distance to proximity sensor

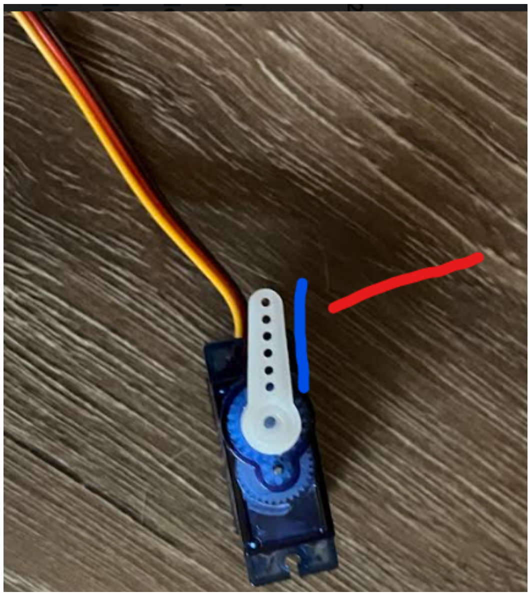
Data configurations are wired through an SPI connection where the commands of the LCD will configure the operations and the data transmitted will be displayed through PORTD.

Proximity Sensor:



Proximity sensor will transmit a trigger response when the correct passcode is entered every 10us turning the pin connected to the trigger on and off with that 10us interval. The response from the echo will help determine the calculated distance using the sound of speed as 343 m/s into a formula converted that will be displayed onto the LCD. Using this number, the servo-motor or gate will either remain closed until the number is smaller than 2 which was what we set the boundary as or open for 1000ms/1 second and then close again.

SG51R Micro-Servo Motor:



This motor is not the occasional 360 degree motor but only turns at around 90 degrees only and at -78 degrees set, we designated the gate to be closed and at -67 degrees we designated the gate to be displayed as opened.

3. Conclusions & Summary

We had an issue with implementing the proximity sensor and the motor because we forgot to change the pins for ICP1 which was an interrupt into another variable and the motor used OCR1A and as such, the signals intertwined and it was not fairly smooth causing the motor to sometimes dip into a random angle due to this interrupt. The issue was resolved after moving the proximity sensor's program out of the interrupt and onto another pin and a similar issue occurred with the keypad because we ran out of pins to use due to the LCD and sensor taking up the pins, VREF took up an extra pin and we tried implementing a different circuit for each problem and we had to manually change the keypad from using all 4x4 buttons into only 3x3 for the basis of this demo.