CPE301 – SPRING 2025

Design Assignment 6

Student Name: Carlos Funes

Student #: 5007995832

Student Email: [funesc2@unlv.nevada.edu](mailto:funesc2@unlv.nevada.edu)

Primary Github address: <https://github.com/CaFu0320>

Directory: <https://github.com/CaFu0320/submission_da/tree/main/DesignAssignments/DA6>

Video Playlist:

<https://youtube.com/playlist?list=PLa3xS6s509hhSrDtIiRlyynXMlhipu13Z&si=vMDsBcMr7TgJm_Hq>

Submit the following for all Labs:

1. In the document, for each task submit the modified or included code (only) with highlights and justifications of the modifications. Also, include the comments.
2. Use the previously create a Github repository with a random name (no CPE/301, Lastname, Firstname). Place all labs under the root folder ESD301/DA, sub-folder named LABXX, with one document and one video link file for each lab, place modified asm/c files named as LabXX-TYY.asm/c.
3. If multiple asm/c files or other libraries are used, create a folder LabXX-TYY and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) and other include files, c) text file with youtube video links (see template).

1. **COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS**

**A diagram of a computer

AI-generated content may be incorrect.**

* ATMEGA328PB MINI BOARD
* UART
* SERVO MOTOR
* HC-SR04 ULTRASONIC SENSOR
* SEVEN SEG DISPLAY

1. **C CODE TASKS**

/\*

\* DA6

\*

\* Created: 4/28/2025 11:50:07 PM

\* Author: Carlos Funes

\*/

#define *F\_CPU* 16000000UL //defining CPU frequency

#include <avr/io.h> //including AVR I/O register definitions

#include <stdint.h> //integer types

#include <avr/interrupt.h> //including interrupt handling functions

#include <util/delay.h> //for delay\_ms()

#include <stdio.h> //I/O functions

#include <stdlib.h> //for functions

#include <string.h> //string handling

//UART config

#define BAUDRATE 9600 //baud rate

#define BAUD\_PRESCALLER (((*F\_CPU* / (BAUDRATE \* 16UL))) - 1) // Prescaler calculation

char uart\_buffer[64]; // Buffer for UART messages

//function prototypes

void USART\_init(void); //UART

void Servo\_init(void); //servo motor

void Ultrasonic\_init(void); // Initialize ultrasonic sensor

void SPI\_init(void); // Initialize SPI for 7-segment

void USART\_putstring(const char\* str); //sending string via UART

*uint16\_t* angle\_to\_OCR(*uint16\_t* angle); //converting angle to PWM value

double read\_distance\_mm(void); //reading distance from sensor

void display\_number\_on\_7seg(*uint16\_t* val); //displaying number on 7-segment

void display\_number\_once(*uint16\_t* val); //single display update

//UART initialization

void USART\_init(void) {

UBRR0H = (*uint8\_t*)(BAUD\_PRESCALLER >> 8); //high byte of baud rate

UBRR0L = (*uint8\_t*)(BAUD\_PRESCALLER); //low byte

UCSR0B = (1 << RXEN0) | (1 << TXEN0); //enabling receiver and transmitter

UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); //8-bit data, 1 stop bit, no parity

}

//sending single character via UART

void USART\_send(unsigned char data) {

while (!(UCSR0A & (1 << UDRE0))); //waiting until buffer is empty

UDR0 = data; //loading data into transmit register

}

//sending string via UART

void USART\_putstring(const char\* str) {

while (\*str) { //looping until end of string

USART\_send(\*str++); //sending each character

}

}

//servo motor initialization

void Servo\_init(void) {

DDRB |= (1 << PB1); //setting PB1 as output (OC1A)

//configuring Timer1 for fast PWM mode

TCCR1A = (1 << COM1A1) | (1 << WGM11); //non-inverting, fast PWM

TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS11) | (1 << CS10); //prescaler 64

ICR1 = 4999; //PWM period for 50Hz (20ms)

}

//angle (0-180) to OCR1A value (105-610)

*uint16\_t* angle\_to\_OCR(*uint16\_t* angle) {

return (*uint16\_t*)(105 + (angle \* 505.0 / 180.0));

}

//ultrasonic sensor initialization (PD5 trigger, PE2 echo)

volatile *uint16\_t* TimerOverflow = 0; //tracking Timer3 overflows

ISR(TIMER3\_OVF\_vect) { //timer3 overflow interrupt

TimerOverflow++; //increment overflow count

}

void Ultrasonic\_init(void) {

DDRD |= (1 << PD5); //PD5 as trigger output

DDRE &= ~(1 << PE2); //PE2 (ICP3) as input

TCCR3A = 0; //normal timer3 operation

TIMSK3 |= (1 << TOIE3); //enabling timer3 overflow interrupt

}

//reading distance in mm

double read\_distance\_mm(void) {

PORTD |= (1 << PD5); //sending 10us trigger pulse

*\_delay\_us*(10);

PORTD &= ~(1 << PD5);

TCCR3B = (1 << ICES3) | (1 << CS10); //capturing rising edge, no prescaler

TIFR3 = (1 << ICF3) | (1 << TOV3); //clearing flags

TimerOverflow = 0; //resetting overflow counter

while (!(TIFR3 & (1 << ICF3))); //waiting for rising edge

TCNT3 = 0; //resetting timer

TCCR3B &= ~(1 << ICES3); //capturing falling edge

TIFR3 = (1 << ICF3) | (1 << TOV3); //clearing flags

TimerOverflow = 0;

while (!(TIFR3 & (1 << ICF3))); //waiting for falling edge

*uint32\_t* pulse = (*uint32\_t*)ICR3 + (65536 \* TimerOverflow);

return (pulse / 58.0 / 16.0) \* 10.0; //converting to mm

}

//SPI and 7 segment display PB2-PB5

#define DATA (1<<PB3) //data pin (PB3)

#define LATCH (1<<PB2) //latch (PB2)

#define CLOCK (1<<PB5) //clock (PB5)

//segment patterns for 0-9

const *uint8\_t* SEGMENT\_MAP[10] = {

0xC0, 0xF9, 0xA4, 0xB0, 0x99, // 0-4

0x92, 0x82, 0xF8, 0x80, 0x90 // 5-9

};

//digit enable patterns for 4-digit display

const *uint8\_t* DIGIT\_SELECT[4] = {0xF1, 0xF2, 0xF4, 0xF8};

void SPI\_init(void) {

DDRB |= (DATA | CLOCK | LATCH); //setting data/latch/clock pins as outputs

PORTB &= ~(DATA | CLOCK | LATCH); //initialize low

SPCR0 = (1 << SPE) | (1 << MSTR); //enabling SPI communication, master mode

}

//sending number pattern via SPI

void SPI\_send(*uint8\_t* byte) {

SPDR0 = byte; //loading data

while (!(SPSR0 & (1 << SPIF))); //waiting until data is fully sent

}

//displaying 4-digit number with multiplexing

void display\_number\_on\_7seg(*uint16\_t* val) {

*uint8\_t* digits[4] = {

(*uint8\_t*)(val % 10), //ones

(*uint8\_t*)((val / 10) % 10), //tens

(*uint8\_t*)((val / 100) % 10), //hundreds

(*uint8\_t*)(val / 1000) //thousands

};

for (*int8\_t* i = 3; i >= 0; i--) { //cycling through each digit position

PORTB &= ~LATCH; //preparing to send data

SPI\_send(SEGMENT\_MAP[digits[i]]); //sending segment pattern

SPI\_send(DIGIT\_SELECT[i]); //selecting digit that lights up

PORTB |= LATCH; //latching high to update

*\_delay\_ms*(2); //display each digit for 2ms

}

}

//single display update

void display\_number\_once(*uint16\_t* val) {

// Similar to above but without delay

*uint8\_t* digits[4] = {val % 10, (val/10)%10, (val/100)%10, val/1000};

for (*int8\_t* i = 3; i >= 0; i--) {

PORTB &= ~LATCH;

SPI\_send(SEGMENT\_MAP[digits[i]]);

SPI\_send(DIGIT\_SELECT[i]);

PORTB |= LATCH;

}

}

//main program

int main(void) {

*uint16\_t* angle; //current servo angle

double distance, min\_distance; //distance measurements

*uint16\_t* display\_min; //minimum distance to display

char angleStr[8], distStr[8]; //UART buffers

USART\_init(); //starting serial communication

Servo\_init(); //preparing servo motor

Ultrasonic\_init(); //preparing distance sensor

SPI\_init(); //preparing display

*sei*(); //enabling interrupts

USART\_putstring("Angle (Degrees), Distance (mm)\n"); //printing

while (1) {

min\_distance = 9999.0; //starting with a high minimum distance

//sweeping servo 0 to 180

for (angle = 0; angle <= 180; angle += 2) {

OCR1A = angle\_to\_OCR(angle); // Moving servo to current angle

distance = read\_distance\_mm(); //measuring distance

if (distance < min\_distance) min\_distance = distance; //tracking closest object

//formating and sending data via UART

*dtostrf*((double)angle, 3, 0, angleStr); //converting angle to text

*dtostrf*(distance, 6, 2, distStr); //converting distance to text

*snprintf*(uart\_buffer, sizeof(uart\_buffer),

"Angle: %s° Distance: %s mm\n", angleStr, distStr); //sending this to the terminal

USART\_putstring(uart\_buffer);

display\_min = (*uint16\_t*)min\_distance; //updating display with the closes distance

display\_number\_on\_7seg(display\_min);

*\_delay\_ms*(100); //waiting between steps

}

//servo 180 to 0

for (angle = 180; angle >= 0; angle -= 2) { //this is the same process from above but in reverse

OCR1A = angle\_to\_OCR(angle);

distance = read\_distance\_mm();

*dtostrf*((double)angle, 3, 0, angleStr);

*dtostrf*(distance, 6, 2, distStr);

*snprintf*(uart\_buffer, sizeof(uart\_buffer),

"Angle: %s° Distance: %s mm\n", angleStr, distStr);

USART\_putstring(uart\_buffer);

display\_number\_on\_7seg((*uint16\_t*)distance);

*\_delay\_ms*(100);

}

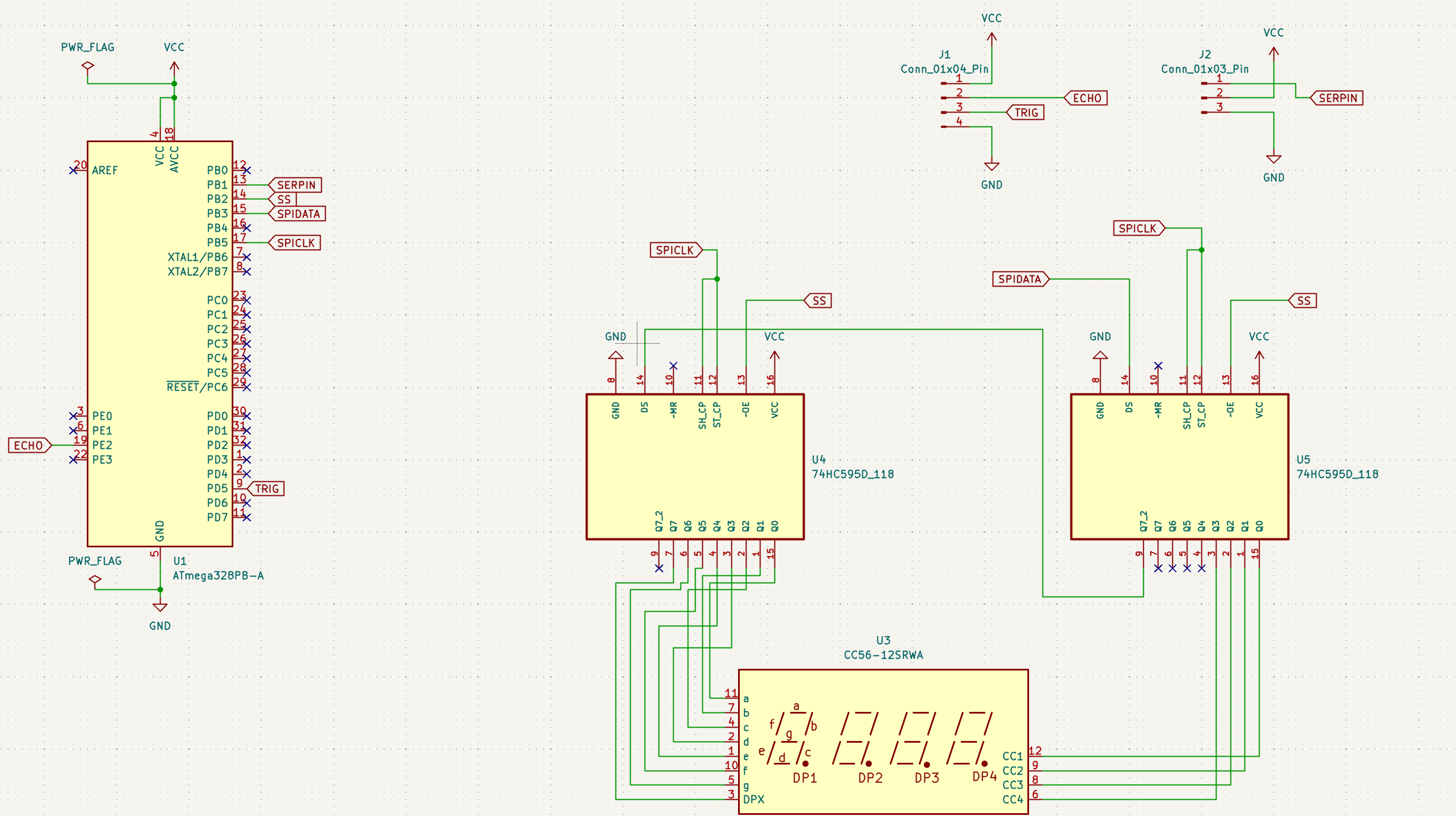
}

}

1. **SCHEMATIC**

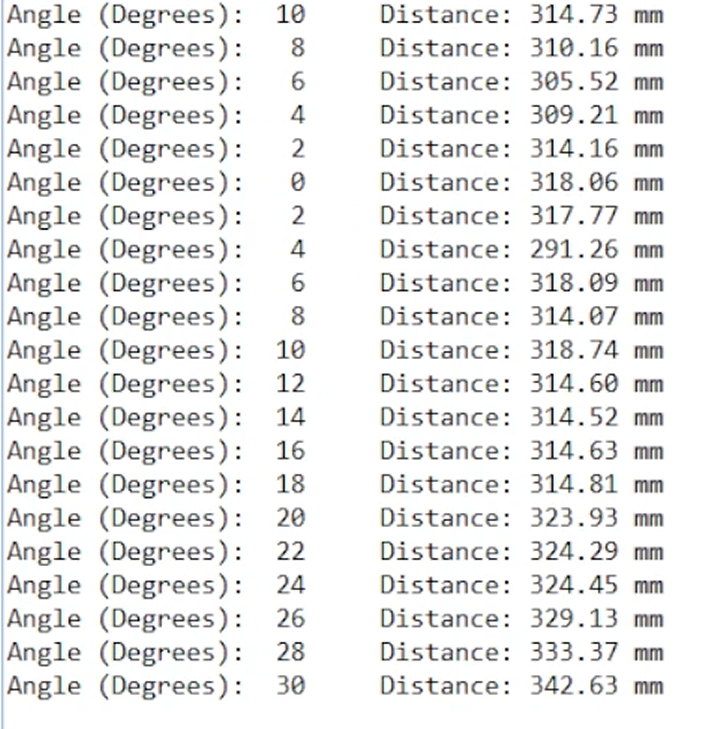
**A computer screen shot of a computer

AI-generated content may be incorrect.**

****

1. **SCREENSHOTS OF EACH TASK OUTPUT (**

**TASK 1**

****

**TASK 2**

**Found on YouTube, but for demonstration purposes:**

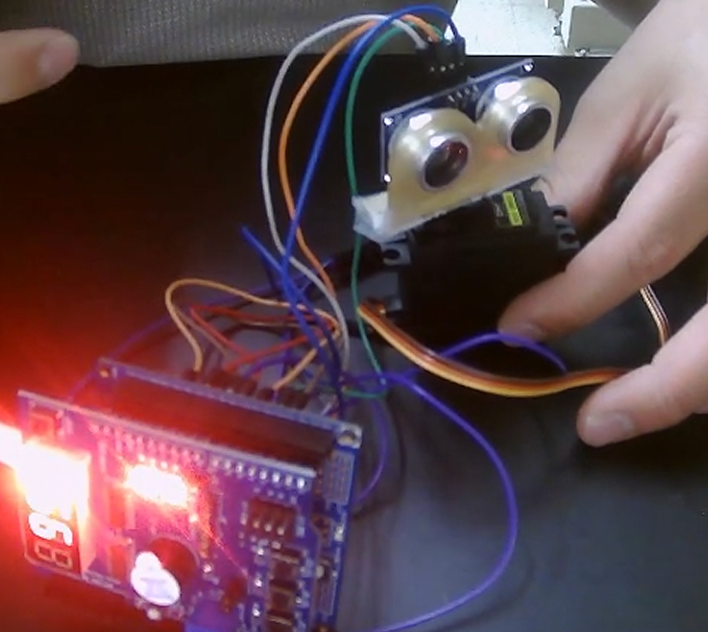
**Clockwise:**

[**https://youtube.com/shorts/q9tgixeA7Qo?si=ZEo5pHIBrCJsBaI4**](https://youtube.com/shorts/q9tgixeA7Qo?si=ZEo5pHIBrCJsBaI4)

**Counterclockwise:**

[**https://youtube.com/shorts/H0FlKaygN3o**](https://youtube.com/shorts/H0FlKaygN3o)

1. **SCREENSHOT OF DEMO**

****

1. **VIDEO LINK FOR DEMO**

Task: <https://youtu.be/keCZ4SwhddY>

1. **GITHUB LINK OF THIS DA**

Tasks c code: <https://github.com/CaFu0320/submission_da/tree/main/DesignAssignments/DA5>

**Student Academic Misconduct Policy**

<http://studentconduct.unlv.edu/misconduct/policy.html>

“This assignment submission is my own, original work”.

Carlos Funes