CPE301 – SPRING 2025

Design Assignment 7

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/DA7>

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 blue rectangle with white text

AI-generated content may be incorrect.**

* ATMEGA328PB MINI BOARD
* UART
* SERIAL PLOTTER
* BMI160 6-DOF IMU SENSOR
* I2C OLED Display
* 2 BREADBOARDS

1. **C CODE TASKS**

/\*

\* DA7\_7v2

\* Author: Carlos Funes

\*/

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

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

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

#include <stdlib.h> //for functions

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

#include <math.h> //include this to calculate angles

#include <avr/pgmspace.h> //storing data into flash memory

#define DISPLAY\_WIDTH 128 //oled width in pixels

#define DISPLAY\_HEIGHT 64 //height in pixels

#define LCD\_I2C\_ADR 0x3C //i2c address

#define ACCEL\_SCALE 16384.0f //accelerometer sensitivity

#define GYRO\_SCALE 16.4f //gyroscope sensitivity

#define FONT ssd1306oled\_font //font data being stored in PROGMEM

const *uint8\_t* special\_char[][2] = { {0,0}, {0xFF,0xFF} }; //placeholder for graphics

void oled\_init(void); //turning on oled screen

void oled\_clear(void); //clearing the screen

void oled\_draw\_string(*uint8\_t* x, *uint8\_t* page, const char \*str); //drawing text at specific positions

//UART

void USART\_Init(unsigned long BAUDRATE) { //setting up the serial plotter

*uint16\_t* ubrr\_value = (*F\_CPU* / (16UL \* BAUDRATE)) - 1;

UCSR0B = (1 << RXEN0) | (1 << TXEN0);

UCSR0C = (1 << UCSZ00) | (1 << UCSZ01);

UBRR0L = ubrr\_value;

UBRR0H = (ubrr\_value >> 8);

}

void USART\_TxChar(char data) { //sending single characters to serial plotter

UDR0 = data;

while (!(UCSR0A & (1<<UDRE0)));

}

void USART\_SendString(char \*str) { //sending a full string over to serial

int i = 0;

while (str[i] != 0) {

USART\_TxChar(str[i]);

i++;

}

}

//I2C

void i2c\_init(void) { //preparing the i2c pins

TWSR0 = 0x00;

TWBR0 = 0x48; //100khz

TWCR0 = (1 << TWEN);

}

void i2c\_start(void) { //starting conversations with device

TWCR0 = (1 << TWSTA) | (1 << TWEN) | (1 << TWINT);

while (!(TWCR0 & (1 << TWINT)));

}

void i2c\_stop(void) { //ending conversations

TWCR0 = (1 << TWSTO) | (1 << TWEN) | (1 << TWINT);

*\_delay\_us*(10);

}

void i2c\_write(*uint8\_t* data) { //sending byte of data to i2c

TWDR0 = data;

TWCR0 = (1 << TWEN) | (1 << TWINT);

while (!(TWCR0 & (1 << TWINT)));

}

*uint8\_t* i2c\_read\_ack(void) { //reading byte and acknowledges it

TWCR0 = (1 << TWEN) | (1 << TWINT) | (1 << TWEA);

while (!(TWCR0 & (1 << TWINT)));

return TWDR0;

}

*uint8\_t* i2c\_read\_nack(void) { //reading byte without acknowledging it

TWCR0 = (1 << TWEN) | (1 << TWINT);

while (!(TWCR0 & (1 << TWINT)));

return TWDR0;

}

//SSD1306 (OLED) functions

#define OLED\_ADDR 0x3C

static void oled\_cmd(*uint8\_t* c){ //sending command to screen

i2c\_start();

i2c\_write(OLED\_ADDR<<1);

i2c\_write(0x00);

i2c\_write(c); i2c\_stop();

}

static void oled\_data(*uint8\_t* d){ //sending pixel to screen

i2c\_start();

i2c\_write(OLED\_ADDR<<1);

i2c\_write(0x40);

i2c\_write(d);

i2c\_stop();

}

static void oled\_setpos(*uint8\_t* col,*uint8\_t* page){ //moving cursor to specific pixel location

oled\_cmd(0xB0|page);

oled\_cmd(0x00|(col&0x0F));

oled\_cmd(0x10|(col>>4));

}

const char ssd1306oled\_font[][6] PROGMEM = { //data font for letters

{0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, // sp

{0x00, 0x00, 0x00, 0x2f, 0x00, 0x00}, // !

{0x00, 0x00, 0x07, 0x00, 0x07, 0x00}, // "

{0x00, 0x14, 0x7f, 0x14, 0x7f, 0x14}, // #

{0x00, 0x24, 0x2a, 0x7f, 0x2a, 0x12}, // $

{0x00, 0x62, 0x64, 0x08, 0x13, 0x23}, // %

{0x00, 0x36, 0x49, 0x55, 0x22, 0x50}, // &

{0x00, 0x00, 0x05, 0x03, 0x00, 0x00}, // '

{0x00, 0x00, 0x1c, 0x22, 0x41, 0x00}, // (

{0x00, 0x00, 0x41, 0x22, 0x1c, 0x00}, // )

{0x00, 0x14, 0x08, 0x3E, 0x08, 0x14}, // \*

{0x00, 0x08, 0x08, 0x3E, 0x08, 0x08}, // +

{0x00, 0x00, 0x00, 0xA0, 0x60, 0x00}, // ,

{0x00, 0x08, 0x08, 0x08, 0x08, 0x08}, // -

{0x00, 0x00, 0x60, 0x60, 0x00, 0x00}, // .

{0x00, 0x20, 0x10, 0x08, 0x04, 0x02}, // /

{0x00, 0x3E, 0x51, 0x49, 0x45, 0x3E}, // 0

{0x00, 0x00, 0x42, 0x7F, 0x40, 0x00}, // 1

{0x00, 0x42, 0x61, 0x51, 0x49, 0x46}, // 2

{0x00, 0x21, 0x41, 0x45, 0x4B, 0x31}, // 3

{0x00, 0x18, 0x14, 0x12, 0x7F, 0x10}, // 4

{0x00, 0x27, 0x45, 0x45, 0x45, 0x39}, // 5

{0x00, 0x3C, 0x4A, 0x49, 0x49, 0x30}, // 6

{0x00, 0x01, 0x71, 0x09, 0x05, 0x03}, // 7

{0x00, 0x36, 0x49, 0x49, 0x49, 0x36}, // 8

{0x00, 0x06, 0x49, 0x49, 0x29, 0x1E}, // 9

{0x00, 0x00, 0x36, 0x36, 0x00, 0x00}, // :

{0x00, 0x00, 0x56, 0x36, 0x00, 0x00}, // ;

{0x00, 0x08, 0x14, 0x22, 0x41, 0x00}, // <

{0x00, 0x14, 0x14, 0x14, 0x14, 0x14}, // =

{0x00, 0x00, 0x41, 0x22, 0x14, 0x08}, // >

{0x00, 0x02, 0x01, 0x51, 0x09, 0x06}, // ?

{0x00, 0x32, 0x49, 0x59, 0x51, 0x3E}, // @

{0x00, 0x7C, 0x12, 0x11, 0x12, 0x7C}, // A

{0x00, 0x7F, 0x49, 0x49, 0x49, 0x36}, // B

{0x00, 0x3E, 0x41, 0x41, 0x41, 0x22}, // C

{0x00, 0x7F, 0x41, 0x41, 0x22, 0x1C}, // D

{0x00, 0x7F, 0x49, 0x49, 0x49, 0x41}, // E

{0x00, 0x7F, 0x09, 0x09, 0x09, 0x01}, // F

{0x00, 0x3E, 0x41, 0x49, 0x49, 0x7A}, // G

{0x00, 0x7F, 0x08, 0x08, 0x08, 0x7F}, // H

{0x00, 0x00, 0x41, 0x7F, 0x41, 0x00}, // I

{0x00, 0x20, 0x40, 0x41, 0x3F, 0x01}, // J

{0x00, 0x7F, 0x08, 0x14, 0x22, 0x41}, // K

{0x00, 0x7F, 0x40, 0x40, 0x40, 0x40}, // L

{0x00, 0x7F, 0x02, 0x0C, 0x02, 0x7F}, // M

{0x00, 0x7F, 0x04, 0x08, 0x10, 0x7F}, // N

{0x00, 0x3E, 0x41, 0x41, 0x41, 0x3E}, // O

{0x00, 0x7F, 0x09, 0x09, 0x09, 0x06}, // P

{0x00, 0x3E, 0x41, 0x51, 0x21, 0x5E}, // Q

{0x00, 0x7F, 0x09, 0x19, 0x29, 0x46}, // R

{0x00, 0x46, 0x49, 0x49, 0x49, 0x31}, // S

{0x00, 0x01, 0x01, 0x7F, 0x01, 0x01}, // T

{0x00, 0x3F, 0x40, 0x40, 0x40, 0x3F}, // U

{0x00, 0x1F, 0x20, 0x40, 0x20, 0x1F}, // V

{0x00, 0x3F, 0x40, 0x38, 0x40, 0x3F}, // W

{0x00, 0x63, 0x14, 0x08, 0x14, 0x63}, // X

{0x00, 0x07, 0x08, 0x70, 0x08, 0x07}, // Y

{0x00, 0x61, 0x51, 0x49, 0x45, 0x43}, // Z

{0x00, 0x00, 0x7F, 0x41, 0x41, 0x00}, // [

{0x00, 0x55, 0x2A, 0x55, 0x2A, 0x55}, // backslash

{0x00, 0x00, 0x41, 0x41, 0x7F, 0x00}, // ]

{0x00, 0x04, 0x02, 0x01, 0x02, 0x04}, // ^

{0x00, 0x40, 0x40, 0x40, 0x40, 0x40}, // \_

{0x00, 0x00, 0x01, 0x02, 0x04, 0x00}, // '

{0x00, 0x20, 0x54, 0x54, 0x54, 0x78}, // a

{0x00, 0x7F, 0x48, 0x44, 0x44, 0x38}, // b

{0x00, 0x38, 0x44, 0x44, 0x44, 0x20}, // c

{0x00, 0x38, 0x44, 0x44, 0x48, 0x7F}, // d

{0x00, 0x38, 0x54, 0x54, 0x54, 0x18}, // e

{0x00, 0x08, 0x7E, 0x09, 0x01, 0x02}, // f

{0x00, 0x18, 0xA4, 0xA4, 0xA4, 0x7C}, // g

{0x00, 0x7F, 0x08, 0x04, 0x04, 0x78}, // h

{0x00, 0x00, 0x44, 0x7D, 0x40, 0x00}, // i

{0x00, 0x40, 0x80, 0x84, 0x7D, 0x00}, // j

{0x00, 0x7F, 0x10, 0x28, 0x44, 0x00}, // k

{0x00, 0x00, 0x41, 0x7F, 0x40, 0x00}, // l

{0x00, 0x7C, 0x04, 0x18, 0x04, 0x78}, // m

{0x00, 0x7C, 0x08, 0x04, 0x04, 0x78}, // n

{0x00, 0x38, 0x44, 0x44, 0x44, 0x38}, // o

{0x00, 0xFC, 0x24, 0x24, 0x24, 0x18}, // p

{0x00, 0x18, 0x24, 0x24, 0x18, 0xFC}, // q

{0x00, 0x7C, 0x08, 0x04, 0x04, 0x08}, // r

{0x00, 0x48, 0x54, 0x54, 0x54, 0x20}, // s

{0x00, 0x04, 0x3F, 0x44, 0x40, 0x20}, // t

{0x00, 0x3C, 0x40, 0x40, 0x20, 0x7C}, // u

{0x00, 0x1C, 0x20, 0x40, 0x20, 0x1C}, // v

{0x00, 0x3C, 0x40, 0x30, 0x40, 0x3C}, // w

{0x00, 0x44, 0x28, 0x10, 0x28, 0x44}, // x

{0x00, 0x1C, 0xA0, 0xA0, 0xA0, 0x7C}, // y

{0x00, 0x44, 0x64, 0x54, 0x4C, 0x44}, // z

{0x00, 0x00, 0x08, 0x77, 0x41, 0x00}, // {

{0x00, 0x00, 0x00, 0x63, 0x00, 0x00}, // Â¦

{0x00, 0x00, 0x41, 0x77, 0x08, 0x00}, // }

{0x00, 0x08, 0x04, 0x08, 0x08, 0x04} // ~

};

static void oled\_char(char c) { //drawing single character on the screen

if (c < 0x20 || c > 0x7E) { //valid ASCII range

c = '?'; //replacING invalid with '?'

}

*uint8\_t* idx = c - 0x20; //index into font array

for (*uint8\_t* i = 0; i < 6; i++) {

oled\_data(pgm\_read\_byte(&ssd1306oled\_font[idx][i])); //send each column

}

oled\_data(0x00); //spacing column after character

}

static void oled\_string(const char\* s){ //drawing full string of characters

while(\*s) oled\_char(\*s++);

}

void oled\_clear(void){ //filling screen with black pixels

for(*uint8\_t* p=0;p<8;p++){

oled\_setpos(0,p);

for(*uint8\_t* c=0;c<128;c++)

oled\_data(0);

}

}

void oled\_init(void){ //initializing screen with startup commands

const *uint8\_t* seq[] = {

0xAE,

0xD5,

0x80,

0xA8,

0x3F,

0xD3,

0x00,

0x40,

0x8D,

0x14,

0x20,

0x00,

0xA1,

0xC8,

0xDA,

0x12,

0x81,

0xCF,

0xD9,

0xF1,

0xDB,

0x40,

0xA4,

0xA6,

0xAF

};

*\_delay\_ms*(100);

for(*uint8\_t* i=0;i<sizeof(seq);i++)

oled\_cmd(seq[i]);

oled\_clear();

}

//OLED helpers

#define VAL\_COL 64

//BMI160

#define BMI160\_ADDR 0x69 //connected to vcc

*int16\_t* accData[3], gyrData[3]; //storing accelerometer array

float pitch = 0.0f, roll = 0.0f, yaw = 0.0f, dt = 0.01; //holding calculated angles

void bmi160\_write(*uint8\_t* reg, *uint8\_t* data) { //sending commands to sensor

i2c\_start();

i2c\_write((BMI160\_ADDR << 1));

i2c\_write(reg);

i2c\_write(data);

i2c\_stop();

}

void bmi160\_read\_bytes(*uint8\_t* reg, *uint8\_t* \*buf, *uint8\_t* len) { //reading data from sensor

i2c\_start();

i2c\_write((BMI160\_ADDR << 1));

i2c\_write(reg);

i2c\_start();

i2c\_write((BMI160\_ADDR << 1) | 1);

for (*uint8\_t* i = 0; i < len; i++) {

buf[i] = (i == len - 1) ? i2c\_read\_nack() : i2c\_read\_ack();

}

i2c\_stop();

}

void bmi160\_init(void) { //configuring the sensor

//reset to ensure clean configuration

bmi160\_write(0x7E, 0xB6);

*\_delay\_ms*(50);

//configuring accelerometer (100hz ODR, +-2g)

bmi160\_write(0x40, 0x2A); //ODR 100Hz, normal mode

bmi160\_write(0x41, 0x03); //+-2g range

//configuring gyroscope (100hz ODR, +-2000dps)

bmi160\_write(0x42, 0x1A); // ODR 100Hz, Normal mode

bmi160\_write(0x43, 0x00); //+-2000dps range

//powering on sensors

bmi160\_write(0x7E, 0x11); //accel normal mode

*\_delay\_ms*(50);

bmi160\_write(0x7E, 0x15); //gyro normal mode

*\_delay\_ms*(50);

}

void bmi160\_read\_raw(void) { //reading accelerometer and gyro data to accData/gyrData

*uint8\_t* data[12];

bmi160\_read\_bytes(0x0C, data, 12);

for (int i = 0; i < 3; i++) {

gyrData[i] = (*int16\_t*)((data[i \* 2 + 1] << 8) | data[i \* 2]);

accData[i] = (*int16\_t*)((data[i \* 2 + 7] << 8) | data[i \* 2 + 6]);

}

}

void complementary\_filter(void) { //converting raw accelometer data to g forces

//converting raw accelerometer data to g's

float ax = accData[0] / ACCEL\_SCALE;

float ay = accData[1] / ACCEL\_SCALE;

float az = accData[2] / ACCEL\_SCALE;

//calculating angles from accelerometer

float accPitch = *atan2f*(ay, az) \* 180 / *M\_PI*; //pitch angle from gravity

float accRoll = *atan2f*(ax, az) \* 180 / *M\_PI*; //roll angle from gravity

//gyro data to dps and integrating

pitch += (gyrData[0] / GYRO\_SCALE) \* dt; //updating pitch using gyro

roll -= (gyrData[1] / GYRO\_SCALE) \* dt; //same for roll

yaw += (gyrData[2] / GYRO\_SCALE) \* dt; //same for yaw

//mixing accelerometer and gyro data

pitch = pitch \* 0.98f + accPitch \* 0.02f; //98% for gyro, 2% for acceleormeter

roll = roll \* 0.98f + accRoll \* 0.02f; //same for roll

}

//MAIN

int main(void) {

char buffer[32]; //temporary string to hold numbers

i2c\_init(); //i2c communication

USART\_Init(9600);

oled\_init(); //turning on screen

oled\_clear(); //clearing the screen

//static labels once during setup

oled\_setpos(0, 0); //moving cursor to top left

oled\_string("Pitch:"); //writing pitch on the screen

oled\_setpos(0, 2); //moving cursor to middle left

oled\_string("Roll:"); //writing roll

oled\_setpos(0, 4); //moving cursor to bottom left

oled\_string("Yaw:"); //writing yaw

*\_delay\_ms*(500); //waiting for stability

bmi160\_init(); //configuring the motion sensor

while (1) {

bmi160\_read\_raw(); //looping forever

complementary\_filter(); //calculating pitch/roll/yaw

//UART output for pitch, roll and yaw

USART\_SendString("Pitch: ");

*dtostrf*(pitch, 6, 2, buffer); //converting pitch into a string using 6 characters and 2 decimals

USART\_SendString(buffer); //sending pitch value using a buffer

USART\_SendString("\t\t");

USART\_SendString("Roll: ");

*dtostrf*(roll, 6, 2, buffer); //same logic for the rest of these

USART\_SendString(buffer);

USART\_SendString("\t\t");

USART\_SendString("Yaw: ");

*dtostrf*(yaw, 6, 2, buffer);

USART\_SendString(buffer);

USART\_SendString("\r\n");

//sending accelerometer data

USART\_SendString("AccX: ");

*dtostrf*(accData[0]/ACCEL\_SCALE, 6, 2, buffer);

USART\_SendString(buffer);

USART\_SendString("\t\t");

USART\_SendString("AccY: ");

*dtostrf*(accData[1]/ACCEL\_SCALE, 6, 2, buffer);

USART\_SendString(buffer);

USART\_SendString("\t\t");

USART\_SendString("AccZ: ");

*dtostrf*(accData[2]/ACCEL\_SCALE, 6, 2, buffer);

USART\_SendString(buffer);

USART\_SendString("\r\n");

//sending Gyro data

USART\_SendString("GyroX: ");

*dtostrf*(gyrData[0]/GYRO\_SCALE, 6, 2, buffer);

USART\_SendString(buffer);

USART\_SendString("\t\t");

USART\_SendString("GyroY: ");

*dtostrf*(gyrData[1]/GYRO\_SCALE, 6, 2, buffer);

USART\_SendString(buffer);

USART\_SendString("\t\t");

USART\_SendString("GyroZ: ");

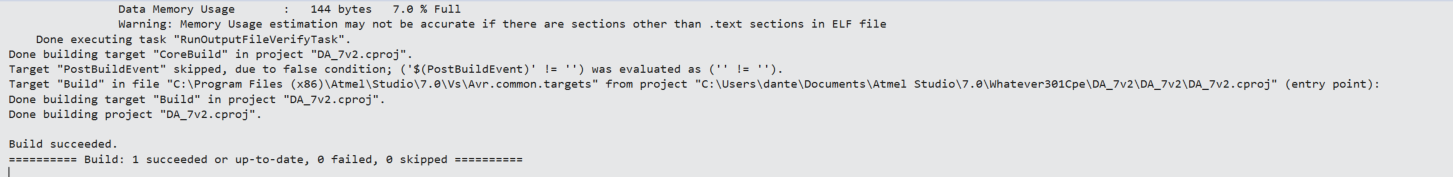
*dtostrf*(gyrData[2]/GYRO\_SCALE, 6, 2, buffer);

USART\_SendString(buffer);

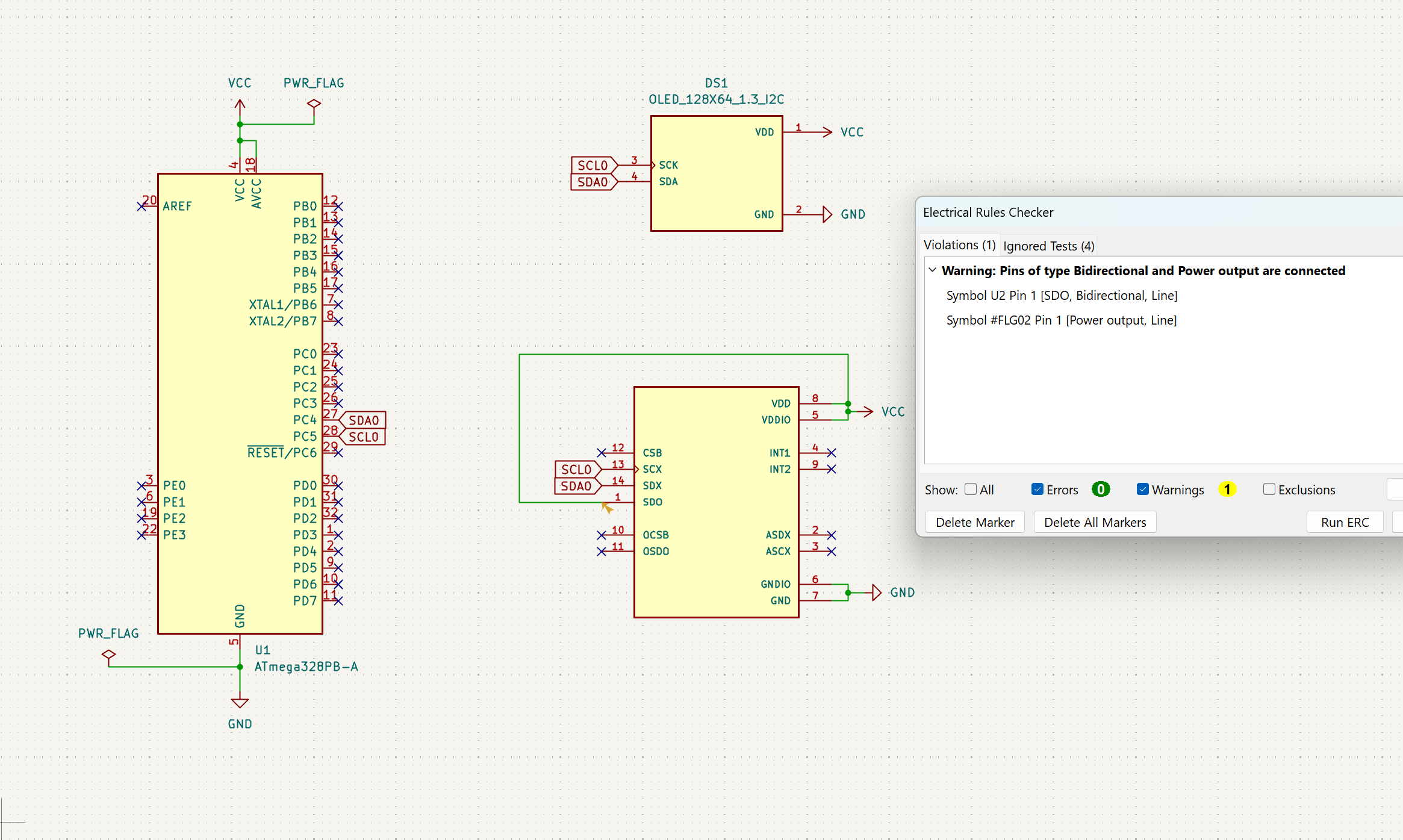
USART\_SendString("\r\n");

*\_delay\_ms*(100);

}

}

1. **SCHEMATIC**



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

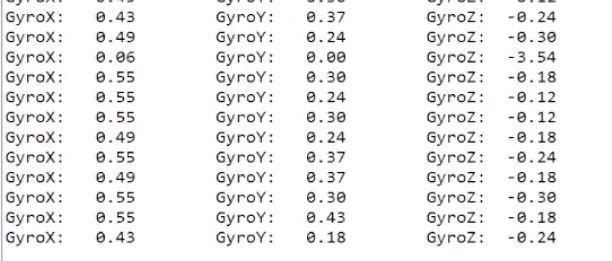
**TASK 1**

**Accelerometer on terminal window:**

**A screenshot of a computer code

AI-generated content may be incorrect.**

**Gyro:**

****

**Pitch, roll, yaw:**

**A table with numbers and letters

AI-generated content may be incorrect.**

**TASK 2**

**Accelerometer on serial plotter:**

**A graph with lines and numbers

AI-generated content may be incorrect.**

**Gyro:**

**A computer screen with a computer screen and a computer screen

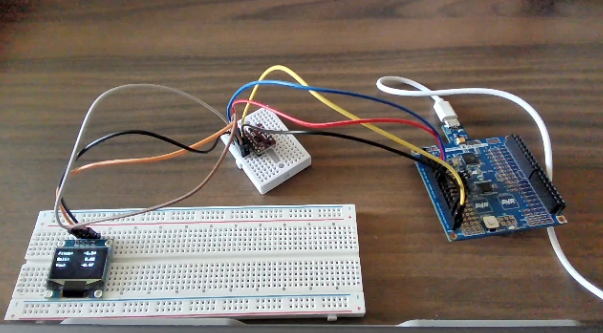
AI-generated content may be incorrect.**

**Pitch, roll, yaw:**

**A graph with lines and text

AI-generated content may be incorrect.**

1. **SCREENSHOT OF DEMO**

****

1. **VIDEO LINK FOR DEMOS**

Task (serial plotter):

<https://youtu.be/q7OJiuuBne4>

Task (terminal):

<https://youtu.be/JJvWWR_0mi0>

1. **GITHUB LINK OF THIS DA**

Tasks c code: <https://github.com/CaFu0320/submission_da/tree/main/DesignAssignments/DA7/DA_7v2/DA_7v2>

**Student Academic Misconduct Policy**

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

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

Carlos Funes