CPE 301 - 1001

DESIGN ASSIGNMENT 7

The goal of the assignment is to develop the above code to do the following:

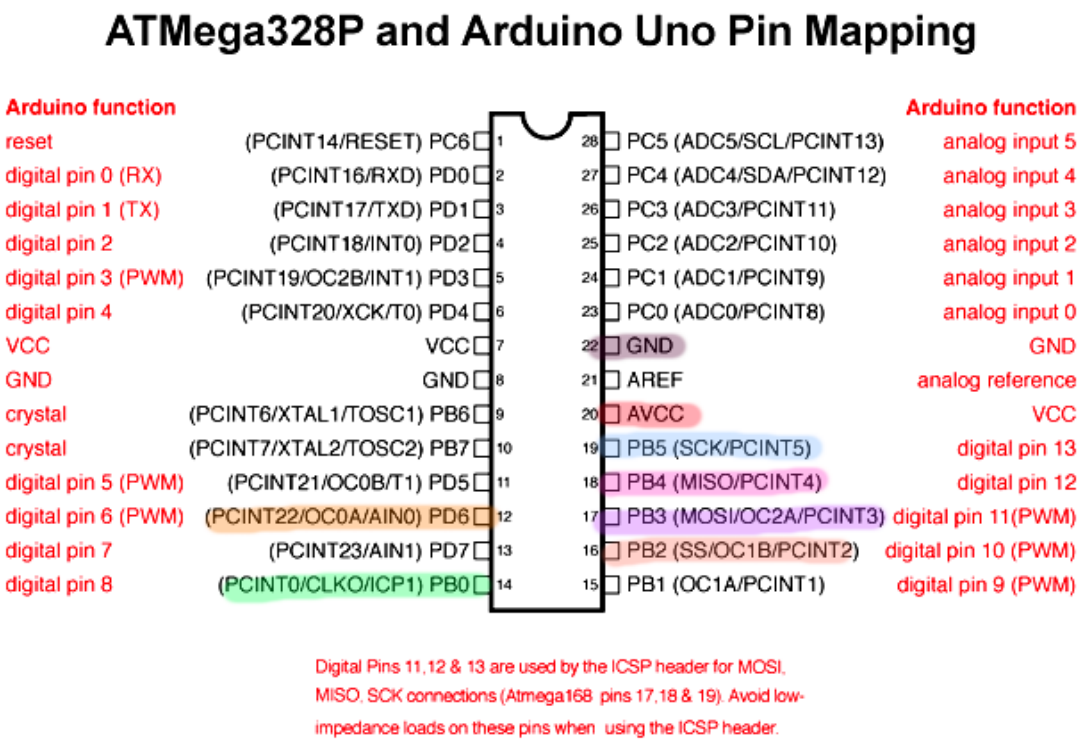
1. Interface the provided BMI160 6-DOF IMU Sensor to the ATmega328pb using the I2C interface. Using the earlier developed code for UART, display the accelerometer and gyro data to the UART Terminal and Serial Plotter application.

2. Apply Complementary to the accelerometer and gyro data to determine the roll, pitch, and yaw of the sensor orientation. Plot the above six values as graphs.

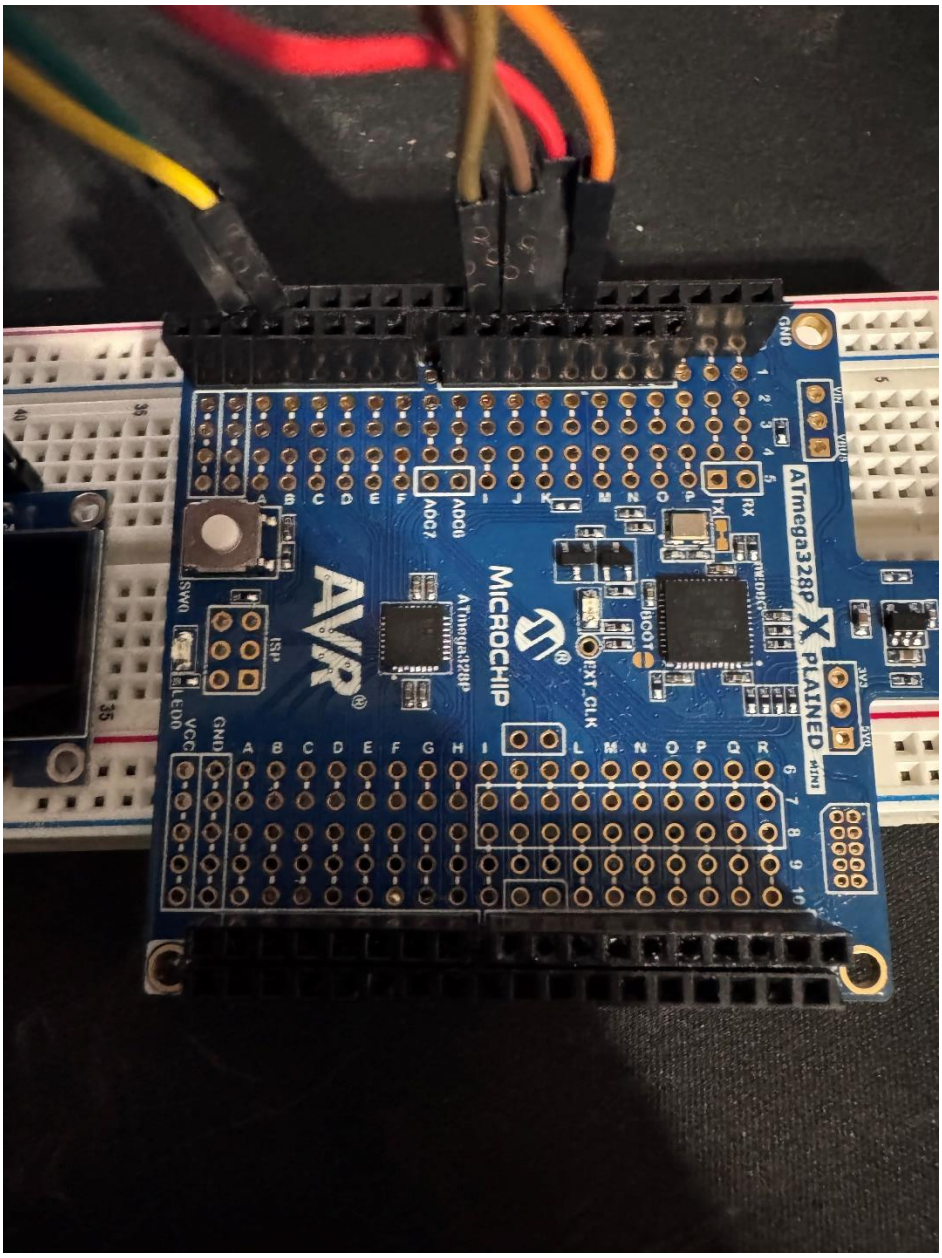
3. Display the roll, pitch, and yaw values in I2C OLED Display.

4. Provide schematics for both devices connected to the microcontroller, accelerometer values should be displayed in a single graph, and gyro values displayed in a single graph, two graphs to show the accelerometer and gyro data. Another graph to show the roll, pitch, and yaw values/data.

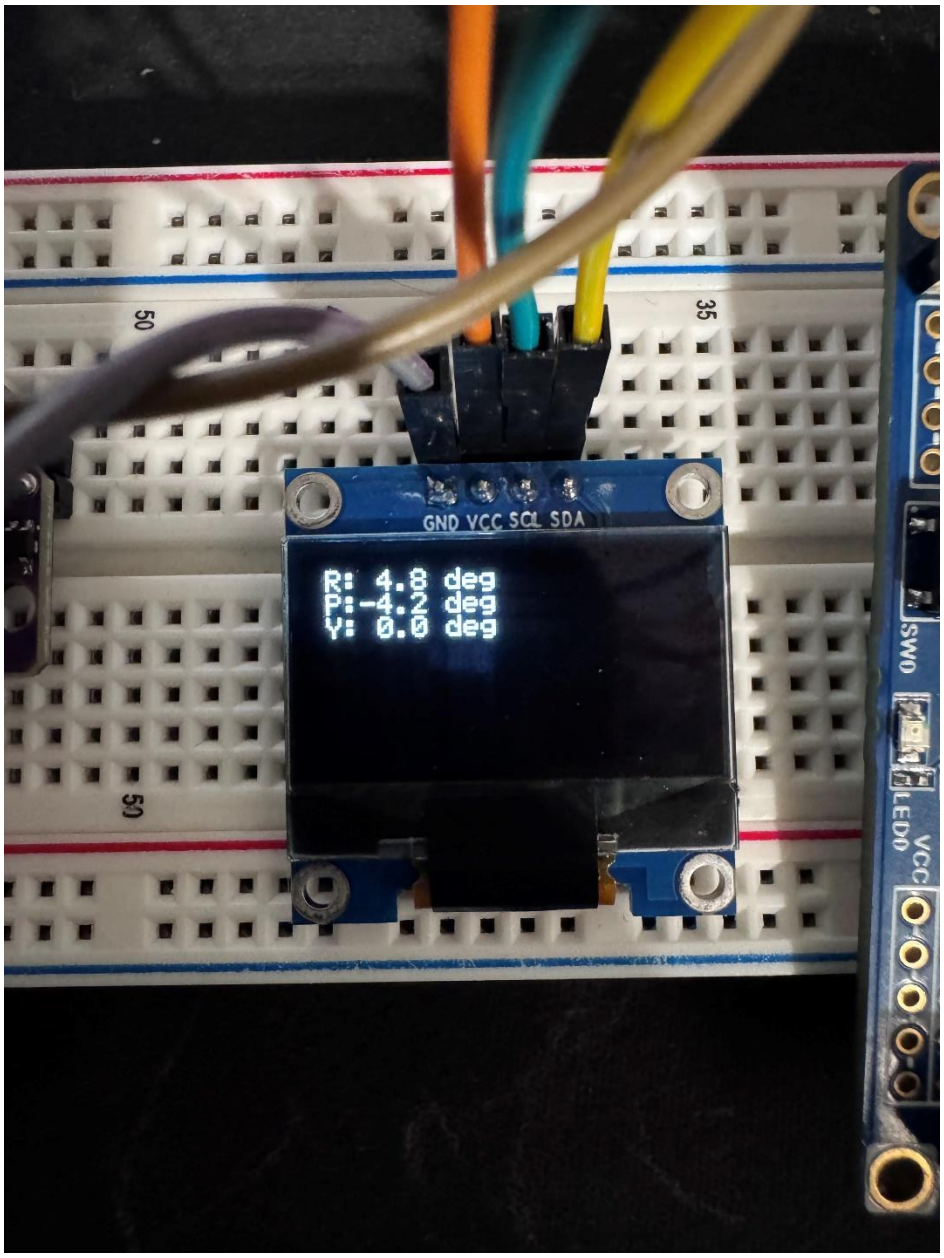
Components Used/Connected



DA7



Atmega328p wiring

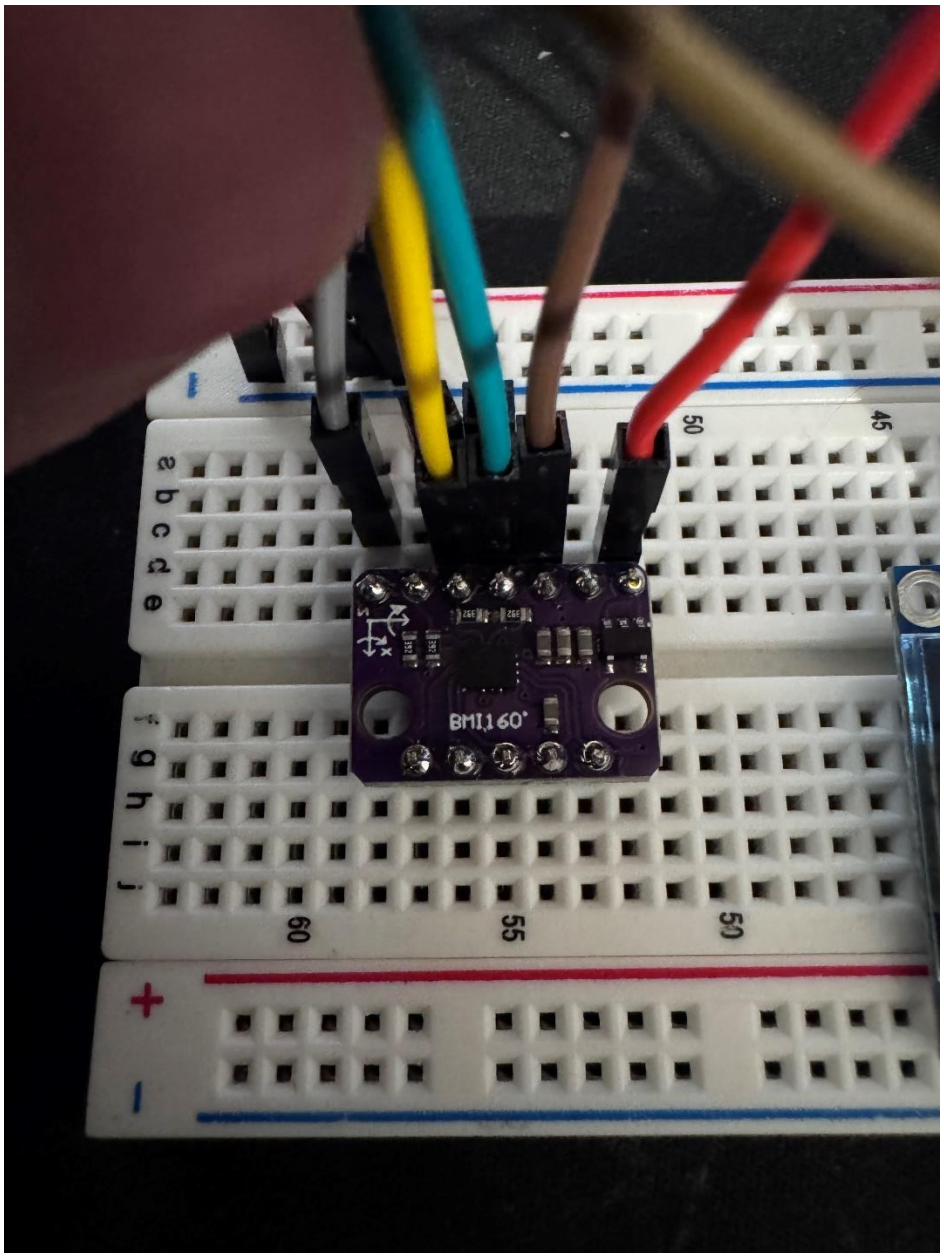


SSD1306-oled-I2C Setup

SCL to PC5

SDA to PC4

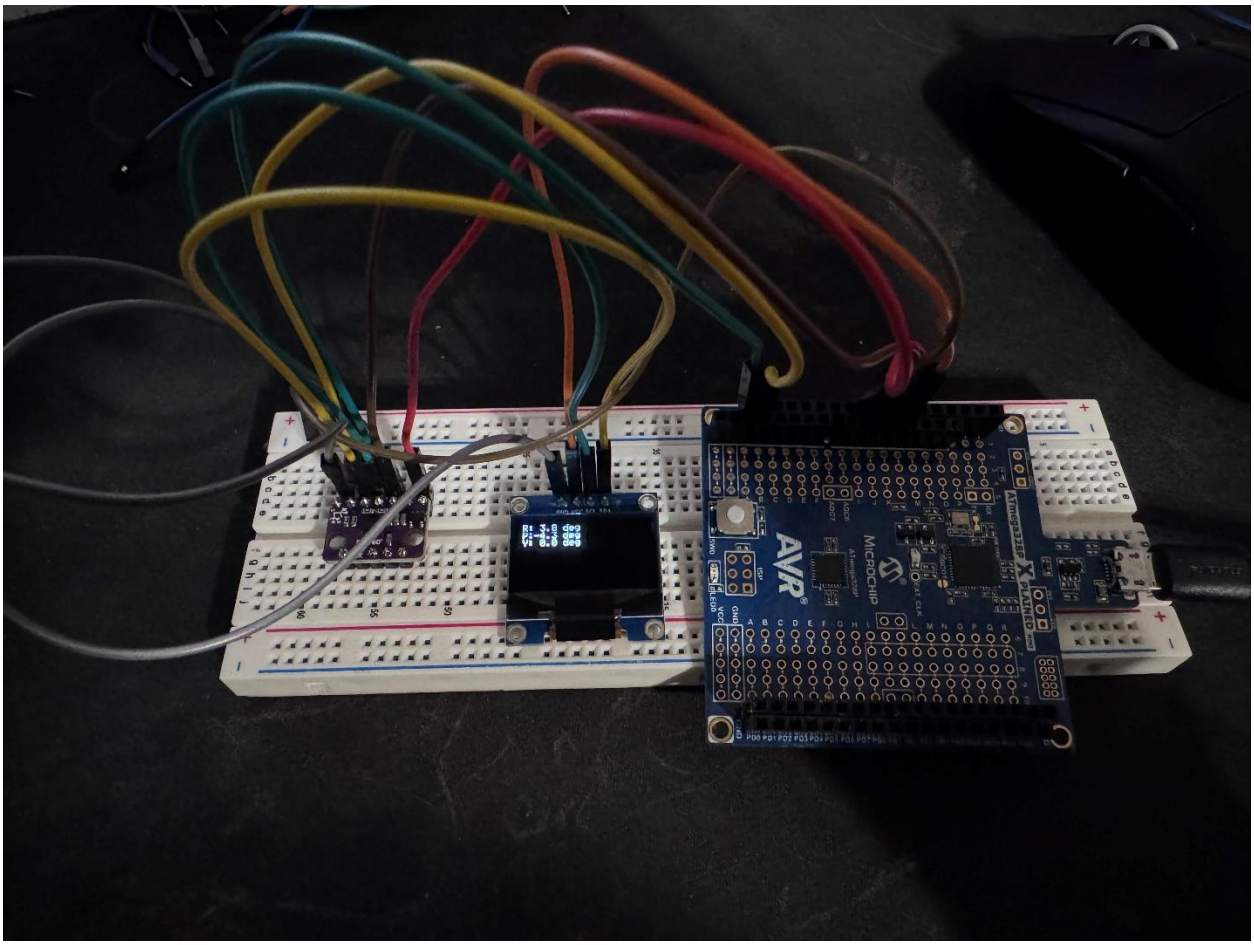
DA7



BMI160 setup

SCL to PC5

SDA to PC4



Whole setup.

AVR C Code

```c
/*
 * DA7.c
 *
 * Created: 5/11/2025 2:47:03 PM
 * Author : enriq
 */

#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <stdint.h>

#include "i2c.h"
#include "lcd.h"
#include "font.h"

// BMI160 I2C address and registers
#define BMI160_ADDR              0x68
#define CMD_REG                  0x7E
#define ACC_CONF_REG             0x40
#define GYR_CONF_REG             0x42
#define DATA_REG                 0x12

// Sensitivities and timing
#define ACCELEROMETER_SENSITIVITY 16384.0f   // LSB/g
#define GYROSCOPE_SENSITIVITY     16.4f      // LSB/(°/s)
#define DT                        0.01f

static int16_t accData[3], gyrData[3];
static float roll = 0.0f, pitch = 0.0f, yaw = 0.0f;

// Initialize UART
static void uart_init(void) {
        UBRR0 = 103;
        UCSR0B = (1<<TXEN0);
        UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
}

static void uart_tx(char c) {
        while (!(UCSR0A & (1<<UDRE0)));
        UDR0 = c;
}

static void uart_print_str(const char *s) {
        while (*s) uart_tx(*s++);
}
```

```
// Initialize BMI160 sensor
static void bmi160_init(void) {
        // Soft reset
        i2c_start((BMI160_ADDR<<1)|0);
        i2c_write(CMD_REG);
        i2c_write(0xB6);
        i2c_stop();
        _delay_ms(100);

        // Accel: 2g, 100Hz
        i2c_start((BMI160_ADDR<<1)|0);
        i2c_write(ACC_CONF_REG);
        i2c_write(0x28);
        i2c_stop();

        // Gyro: 250°/s, 100Hz
        i2c_start((BMI160_ADDR<<1)|0);
        i2c_write(GYR_CONF_REG);
        i2c_write(0x28);
        i2c_stop();

        // Normal modes
        i2c_start((BMI160_ADDR<<1)|0);
        i2c_write(CMD_REG);
        i2c_write(0x15);
        i2c_stop();
        _delay_ms(50);
        i2c_start((BMI160_ADDR<<1)|0);
        i2c_write(CMD_REG);
        i2c_write(0x11);
        i2c_stop();
        _delay_ms(50);
}

// Read 6-axis data
static void bmi160_read(void) {
        i2c_start((BMI160_ADDR<<1)|0);
        i2c_write(DATA_REG);
        i2c_start((BMI160_ADDR<<1)|1);

        // Accel XYZ
        for (int i = 0; i < 3; i++) {
                uint8_t lo = i2c_read_ack();
                uint8_t hi = i2c_read_ack();
                accData[i] = (int16_t)(hi<<8 | lo);
        }
        // Gyro XYZ
        for (int i = 0; i < 2; i++) {
                uint8_t lo = i2c_read_ack();
                uint8_t hi = i2c_read_ack();
                gyrData[i] = (int16_t)(hi<<8 | lo);
        }
        uint8_t lo = i2c_read_ack();
        uint8_t hi = i2c_read_nack();
```

```
        gyrData[2] = (int16_t)(hi<<8 | lo);
        i2c_stop();
}

// Complementary filter
static void ComplementaryFilter(void) {
        float pitchAcc = atan2f((float)accData[1], (float)accData[2]) * 180.0f /
M_PI;
        float rollAcc  = atan2f((float)accData[0], (float)accData[2]) * 180.0f /
M_PI;
        pitch = pitch*0.98f + pitchAcc*0.02f;
        roll  = roll*0.98f  + rollAcc*0.02f;
        yaw  += ((float)gyrData[2] / GYROSCOPE_SENSITIVITY) * DT;
}

int main(void) {
        char uart_vals1[128];
        char uart_vals2[64];
        char sabuf[8], sbbuf[8], scbuf[8];
        char gxbuf[8], gybuf[8], gzbuf[8];
        char rbuf[8], pbuf[8], ybuf[8];

        // Init peripherals
        uart_init();
        i2c_init();
        bmi160_init();
        lcd_init(0xAF);
        lcd_clrscr();
        lcd_display();
        sei();

        while (1) {
                // Read and compute
                bmi160_read();
                ComplementaryFilter();
                float ax = accData[0] / ACCELEROMETER_SENSITIVITY;
                float ay = accData[1] / ACCELEROMETER_SENSITIVITY;
                float az = accData[2] / ACCELEROMETER_SENSITIVITY;
                float gx = gyrData[0] / GYROSCOPE_SENSITIVITY;
                float gy = gyrData[1] / GYROSCOPE_SENSITIVITY;
                float gz = gyrData[2] / GYROSCOPE_SENSITIVITY;
                dtostrf(ax, 4, 2, sabuf);
                dtostrf(ay, 4, 2, sbbuf);
                dtostrf(az, 4, 2, scbuf);
                dtostrf(gx, 4, 1, gxbuf);
                dtostrf(gy, 4, 1, gybuf);
                dtostrf(gz, 4, 1, gzbuf);
                dtostrf(roll, 4, 1, rbuf);
                dtostrf(pitch,4, 1, pbuf);
                dtostrf(yaw,  4, 1, ybuf);

                // UART output
                snprintf(uart_vals1, sizeof(uart_vals1),
                "AX = %sg  AY = %sg  AZ = %sg\n"
                "GX = %s  GY = %s  GZ = %s\n",
```

```
                sabuf, sbbuf, scbuf,
                gxbuf, gybuf, gzbuf
                );
                snprintf(uart_vals2, sizeof(uart_vals2),
                "R = [%s] P = [%s] Y = [%s]\n",
                rbuf, pbuf, ybuf
                );
                uart_print_str(uart_vals1);
                uart_print_str(uart_vals2);
                uart_print_str("\n\n");

                // OLED update
                lcd_clrscr();
                lcd_puts_fb(0, 0, "R:");   lcd_puts_fb(2, 0, rbuf);   lcd_puts_fb(6,
0, " deg");
                lcd_puts_fb(0, 1, "P:");   lcd_puts_fb(2, 1, pbuf);   lcd_puts_fb(6,
1, " deg");
                lcd_puts_fb(0, 2, "Y:");   lcd_puts_fb(2, 2, ybuf);   lcd_puts_fb(6,
2, " deg");
                lcd_display();

                _delay_ms(100);
        }
        return 0;
}
```

```c
// i2c.c
#define F_CPU 16000000UL

#include <avr/io.h>
#include "i2c.h"

void i2c_init(void) {
        TWSR = 0;  // prescaler = 1
        TWBR = (uint8_t)(((F_CPU/100000UL) - 16UL) / 2UL);
        TWCR = (1<<TWEN);
        // enable internal pull-ups on PC4/PC5
        DDRC  &= ~((1<<PC4)|(1<<PC5));
        PORTC |=  (1<<PC4)|(1<<PC5);
}

uint8_t i2c_start(uint8_t addr_rw) {
        TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);
        while (!(TWCR & (1<<TWINT)));
        TWDR = addr_rw;
        TWCR = (1<<TWINT)|(1<<TWEN);
        while (!(TWCR & (1<<TWINT)));
        return (TWSR & 0xF8);
}

uint8_t i2c_write(uint8_t data) {
        TWDR = data;
        TWCR = (1<<TWINT)|(1<<TWEN);
        while (!(TWCR & (1<<TWINT)));
        return (TWSR & 0xF8);
}

void i2c_stop(void) {
        TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);
        while (TWCR & (1<<TWSTO));
}

uint8_t i2c_read_ack(void) {
        TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWEA);
        while (!(TWCR & (1<<TWINT)));
        return TWDR;
}

uint8_t i2c_read_nack(void) {
        TWCR = (1<<TWINT)|(1<<TWEN);
        while (!(TWCR & (1<<TWINT)));
        return TWDR;
}
```

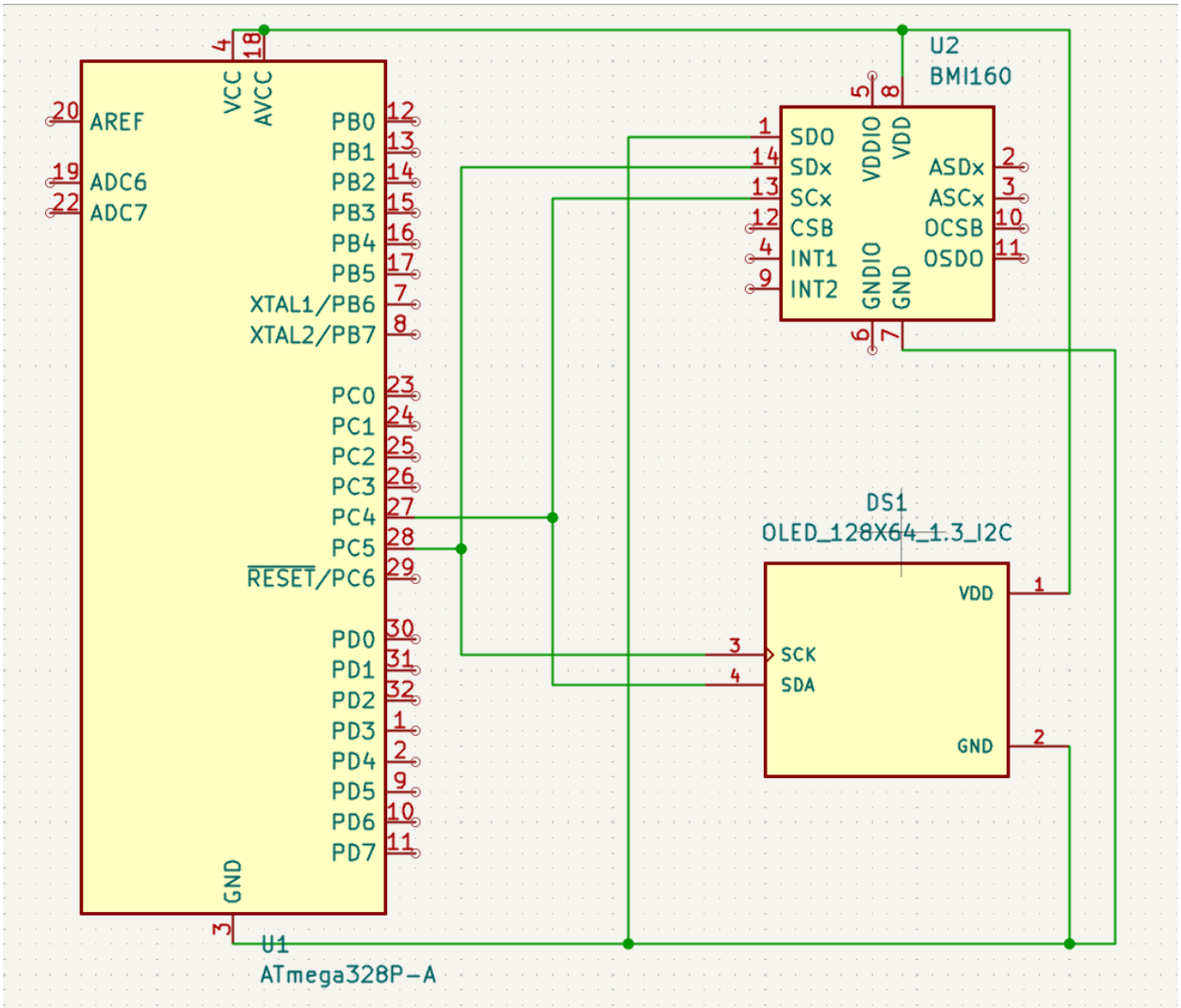
```c
/* lcd.c */
#define F_CPU 16000000UL

#include <avr/io.h>
#include <util/delay.h>
#include <string.h>
#include <avr/pgmspace.h>
#include "i2c.h"
#include "lcd.h"
#include "font.h"

#define SSD1306_ADDR 0x3C
#define SSD1306_128_64
static uint8_t framebuf[128*8];

static void ssd1306_command(uint8_t cmd) {
        i2c_start((SSD1306_ADDR<<1));
        i2c_write(0x00);
        i2c_write(cmd);
        i2c_stop();
}

void lcd_init(uint8_t disp_on_cmd) {
        // hardware reset on PD2
        DDRD |= (1<<PD2);
        PORTD &= ~(1<<PD2); _delay_ms(10);
        PORTD |=  (1<<PD2); _delay_ms(10);

        // initialization sequence
        ssd1306_command(0xAE);              // display off
        ssd1306_command(0xD5); ssd1306_command(0x80);
        ssd1306_command(0xA8); ssd1306_command(0x3F);
        ssd1306_command(0xD3); ssd1306_command(0x00);
        ssd1306_command(0x20); ssd1306_command(0x02);
        ssd1306_command(0x8D); ssd1306_command(0x14);
        ssd1306_command(0xA1);
        ssd1306_command(0xC8);
        ssd1306_command(0xDA); ssd1306_command(0x12);
        ssd1306_command(0x81); ssd1306_command(0xFF);
        ssd1306_command(0xD9); ssd1306_command(0xF1);
        ssd1306_command(0xDB); ssd1306_command(0x40);
        ssd1306_command(0xA4);
        ssd1306_command(0xA6);
        ssd1306_command(disp_on_cmd);      // display on/off
}

void lcd_clrscr(void) {
        memset(framebuf, 0, sizeof(framebuf));
}

void lcd_gotoxy(uint8_t x, uint8_t y) {
        uint8_t col = x * 6;
        ssd1306_command(0xB0 + y);
        ssd1306_command(0x00 + (col & 0x0F));
        ssd1306_command(0x10 + (col >> 4));
```

```
}

void lcd_puts(const char *s) {
        while (*s) {
                char c = *s++;
                if (c < 32 || c > 127) c = '?';
                uint8_t ci = c < 32 || c > 127 ? 0 : (c - 32);
                i2c_start((SSD1306_ADDR<<1));
                i2c_write(0x40);              // data stream
                for (uint8_t i = 0; i < 6; i++) {
                        uint8_t b = pgm_read_byte(&ssd1306oled_font[ci][i]);
                        i2c_write(b);
                }
                i2c_stop();
        }
}

void lcd_putchar_fb(uint8_t x, uint8_t y, char c) {
        if (c < 32 || c > 127) c = '?';
        uint8_t ci = c - 32;
        uint16_t base = y*128 + x*6;
        for (uint8_t i = 0; i < 6; i++) {
                framebuf[base + i] = pgm_read_byte(&ssd1306oled_font[ci][i]);
        }
}

void lcd_puts_fb(uint8_t x, uint8_t y, const char* s) {
        while (*s) {
                lcd_putchar_fb(x++, y, *s++);
        }
}

void lcd_display(void) {
        for (uint8_t page = 0; page < 8; page++) {
                ssd1306_command(0xB0 + page);
                ssd1306_command(0x00);
                ssd1306_command(0x10);
                i2c_start((SSD1306_ADDR<<1));
                i2c_write(0x40);
                for (uint8_t col = 0; col < 128; col++) {
                        i2c_write(framebuf[page*128 + col]);
                }
                i2c_stop();
        }
}
```

```
/*
 *  font.c
 *  i2c
 *
 *  Created by Michael Köhler on 16.09.18.
 *  Copyright 2018 Skie-Systems. All rights reserved.
 *
 */
#include "font.h"

const char ssd1306oled_font[][6] PROGMEM = {
{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}, // |
{0x00, 0x00, 0x41, 0x77, 0x08, 0x00}, // }
{0x00, 0x08, 0x04, 0x08, 0x08, 0x04}, // ~
/* end of normal char-set */
```

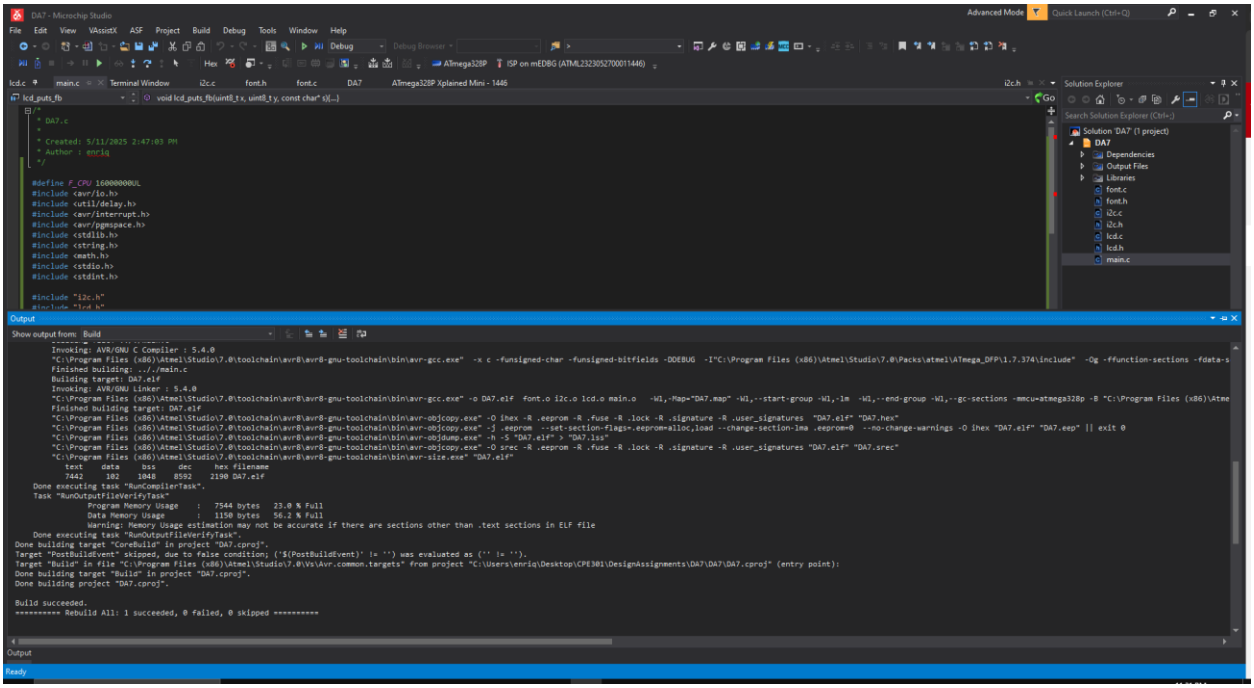
```
/* put your own signs/chars here, edit special_char too */
/* be sure that your first special char stand here */
{0x00, 0x3A, 0x40, 0x40, 0x20, 0x7A}, // ü, !!! Important: this must be
special_char[0] !!!
{0x00, 0x3D, 0x40, 0x40, 0x40, 0x3D}, // Ü
{0x00, 0x21, 0x54, 0x54, 0x54, 0x79}, // ä
{0x00, 0x7D, 0x12, 0x11, 0x12, 0x7D}, // Ä
{0x00, 0x39, 0x44, 0x44, 0x44, 0x39}, // ö
{0x00, 0x3D, 0x42, 0x42, 0x42, 0x3D}, // Ö
{0x00, 0x02, 0x05, 0x02, 0x00, 0x00}, // °
{0x00, 0x7E, 0x01, 0x49, 0x55, 0x73}, // ß
{0x00, 0x7C, 0x10, 0x10, 0x08, 0x1C} // µ
};
```



Schematic showing how the OLED and BMI160 sensor were connected to the atmega. The SCL from both OLED and BMI160 were wired to PC5.

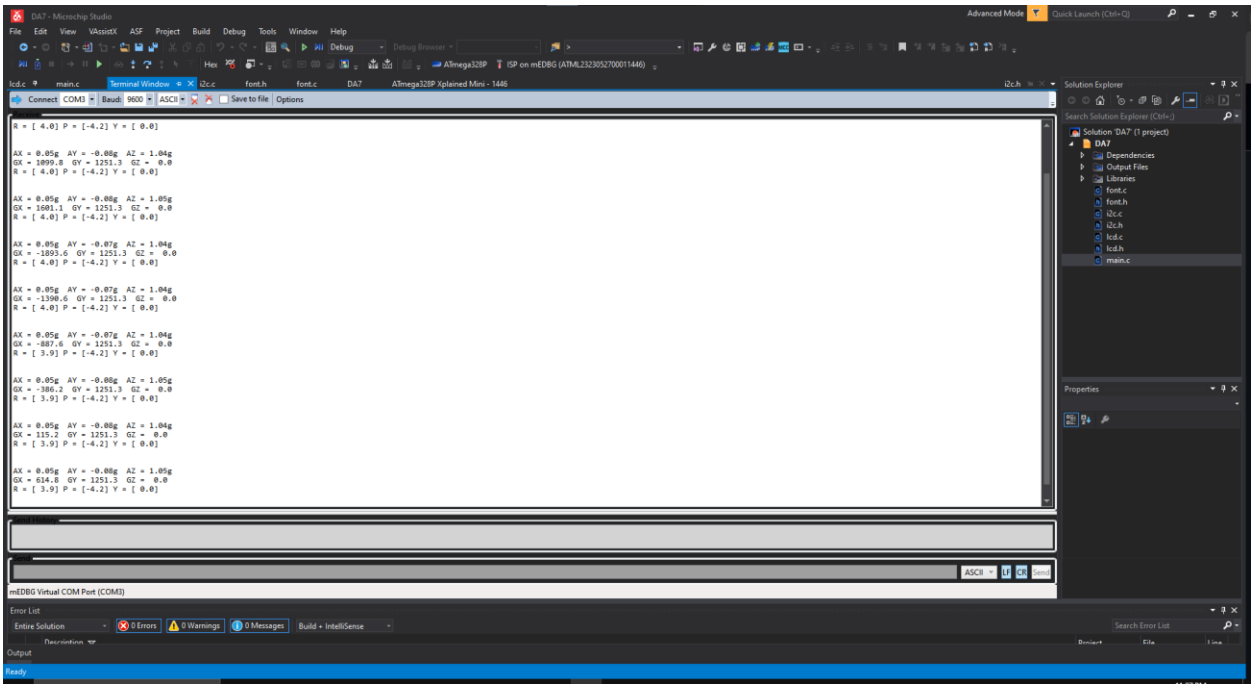
SDA from both OLED and BMI160 were wired to PC4.

DA7



Successful Compilation

DA7



Successfully reading values from BMI160.

A[X, Y, Z] – Accelerometer values

G[X, Y, Z] – Gyroscope values

R – Roll

P – Pitch

Y – Yaw