STONY BROOK UNIVERSITY
DEPARTMENT OF COMPUTER AND ELECTRICAL ENGINEERING

ESE 381.L02

Lab 11: Air Quality System II - Extension of the Features of the Air Quality System I

Name: Jason Tan
SBU ID #: 112319102
Due Date: April 28, 2022 by 9PM

DESIGN TASK 1

```c
/*
* lcd_dog_AVR128_driver.h
*
* Created: 3/19/2022 8:09:13 PM
* Author : jason
*/

#include <avr/io.h>
#include <stdio.h>

/*
Is what is responsible for transmitting the command for the LCD
*/
void lcd_spi_transmit_CMD (unsigned char cmd);

/*
Is what is responsible for transmitting the data for the LCD into what is to be
  displayed
*/
void lcd_spi_transmit_DATA (unsigned char cmd);

/*
Initialize the SPI LCD to being blank
*/
void init_spi_lcd (void);

/*
Initialize the LCD Dog to being blank
*/
void init_lcd_dog (void);

/*
Function prototype for 40 ms
*/
void delay_40mS(void);

/*
Function prototype for 40 microsecond
*/
void delay_30uS(void);

/*
To update on the LCD for something to be displayed
*/
void update_lcd_dog(void);


// Display buffer for DOG LCD using sprintf()
char dsp_buff1[17];
char dsp_buff2[17];
char dsp_buff3[17];
```

```
void lcd_spi_transmit_CMD (unsigned char cmd) {
    //Poll until ready to send the command
    //while(!(SPI0_INTFLAGS & SPI_IF_bm)){}
    PORTC_OUT &= ~PIN0_bm;  //Clear PC0 = RS = 0 = command
    PORTA_OUT &= ~PIN7_bm;  //clear PA7 = /SS = selected
    SPI0_DATA = cmd;
    //Poll until ready to send the command
    while(!(SPI0_INTFLAGS & SPI_IF_bm)){}
    PORTA_OUT |= PIN7_bm;   //clear PA7 = /SS = selected

}

void lcd_spi_transmit_DATA (unsigned char cmd) {
    //Poll until ready to send the command
    while(!(SPI0_INTFLAGS & SPI_IF_bm)){}
    PORTC_OUT |= PIN0_bm;   //PC0 = RS = 1 = command
    PORTA_OUT &= ~PIN7_bm;  //clear PA7 = /SS = selected
    SPI0_DATA = cmd;
    //Poll until ready to send the command
    while(!(SPI0_INTFLAGS & SPI_IF_bm)){}
    PORTA_OUT |= PIN7_bm;   //clear PA7 = /SS = selected

}

void init_spi_lcd (void) {
    PORTA_DIR |= PIN4_bm | PIN6_bm | PIN7_bm;  //Set MOSI, SCK and //SS as output
      while MISO as input

    PORTC_DIR |= PIN0_bm;  //Set RS of LCD as output

    SPI0_CTRLA |= SPI_ENABLE_bm | SPI_MASTER_bm; //Enable the SPI and make it in the
      Master Mode

    SPI0_CTRLB |= SPI_SSD_bm | SPI_MODE1_bm | SPI_MODE0_bm;   //Put the SPI with slave
       select (/SS) to be enabled and be in SPI Mode 3 (CPOL = 1 and CPHA = 1)

    //Wait to clears the IF flag in the INTFLAG meaning there no serial data yet to be
       transferred
    //while(SPI0_INTFLAGS & SPI_IF_bm){}
    PORTC_OUT &= ~PIN0_bm;  //PC0 = RS = 0 = command



}


void init_lcd_dog (void) {

    init_spi_lcd();     //Initialize mcu for LCD SPI
```

```
    //start_dly_40ms:
    delay_40mS();    //startup delay.


    //func_set1:
    lcd_spi_transmit_CMD(0x39);   // sedn function set #1
    delay_30uS();   //delay for command to be processed


    //func_set2:
    lcd_spi_transmit_CMD(0x39); //send fuction set #2
    delay_30uS();   //delay for command to be processed


    //bias_set:
    lcd_spi_transmit_CMD(0x1E); //set bias value.
    delay_30uS();   //delay for command to be processed


    //power_ctrl:
    lcd_spi_transmit_CMD(0x55); //~ 0x50 nominal for 5V
    //~ 0x55 for 3.3V (delicate adjustment).
    delay_30uS();   //delay for command to be processed


    //follower_ctrl:
    lcd_spi_transmit_CMD(0x6C); //follower mode on...
    delay_40mS();   //delay for command to be processed


    //contrast_set:
    lcd_spi_transmit_CMD(0x7F); //~ 77 for 5V, ~ 7F for 3.3V
    delay_30uS();   //delay for command to be processed


    //display_on:
    lcd_spi_transmit_CMD(0x0c); //display on, cursor off, blink off
    delay_30uS();   //delay for command to be processed


    //clr_display:
    lcd_spi_transmit_CMD(0x01); //clear display, cursor home
    delay_30uS();   //delay for command to be processed


    //entry_mode:
    lcd_spi_transmit_CMD(0x06); //clear display, cursor home
    delay_30uS();   //delay for command to be processed
}


void delay_40mS(void) {
```

```
    int i;
    for (int n = 40; n > 0; n--)
    for (i = 0; i < 800; i++)
    __asm("nop");
}

void delay_30uS(void) {
    int i;
    for (int n = 1; n > 0; n--)
    for (i = 0; i < 2; i++)
    __asm("nop");
}


// Updates the LCD display lines 1, 2, and 3, using the
// contents of dsp_buff_1, dsp_buff_2, and dsp_buff_3, respectively.
void update_lcd_dog(void) {

    init_spi_lcd();     //init SPI port for LCD.

    // send line 1 to the LCD module.
    lcd_spi_transmit_CMD(0x80); //init DDRAM addr-ctr
    delay_30uS();
    for (int i = 0; i < 16; i++) {
        lcd_spi_transmit_DATA(dsp_buff1[i]);
        delay_30uS();
    }

    // send line 2 to the LCD module.
    lcd_spi_transmit_CMD(0x90); //init DDRAM addr-ctr
    delay_30uS();
    for (int i = 0; i < 16; i++) {
        lcd_spi_transmit_DATA(dsp_buff2[i]);
        delay_30uS();
    }

    // send line 3 to the LCD module.
    lcd_spi_transmit_CMD(0xA0); //init DDRAM addr-ctr
    delay_30uS();
    for (int i = 0; i < 16; i++) {
        lcd_spi_transmit_DATA(dsp_buff3[i]);
        delay_30uS();
    }
}
```

```c
/*
 * SCD41_LCD_USART3_Lab11Task1.c
 *
 * Created: 4/26/2022 6:13:25 PM
 * Author : jason
 */

#include <avr/io.h>
#include <math.h>
#define F_CPU 4000000
#include "lcd_dog_AVR128_driver.h"
#include "SCD41_AVR128_driver.h"
#include "USART3_asynch_transmit.h"
#include <util/delay.h>

#define MAX_INPUT_DISPLAY 5


uint16_t CO2;
uint16_t Temp;
uint16_t Rh;

uint16_t baudRate = 9600;   //For the baud rate of USART3
uint8_t dataBits = USART_CHSIZE_8BIT_gc;    //For the (character size)  CHSIZE[2:0]
unsigned char parity = 0x00;    //PMODE[1:0]



int main(void)
{

    init_lcd_dog();      //Initialize the buffer of the LCD
    I2C0_SCD41_init();   //Initializes  the AVR128DB48 I2C0 to communicate with SCD41

    //Function to initialize the USART3 baud rate, data bit and parity
    USART3_init(baudRate, dataBits, parity);

    while (1)
    {

        SCD41_start_periodic_measurement(I2CSLAVE_ADDR_WRITE,
          ADDRESS_STARTPERIODIC_MSB, ADDRESS_STARTPERIODIC_LSB);

        //Keep polling until data is ready which can be measured
        while(!SCD41_get_data_ready_status(I2CSLAVE_ADDR_WRITE,
          ADDRESS_GETDATAREADY_MSB, ADDRESS_GETDATAREADY_LSB));

        SCD41_read_measurement(I2CSLAVE_ADDR_WRITE, ADDRESS_READMEASUR_MSB,
          ADDRESS_READMEASUR_LSB);

        CO2 = getParseCO2;
        Temp = -45 + ( (175 * getParseTemp) / (pow(2, 16)));
```

```
        Rh = 100 * (((float)getParseRh) / (pow(2, 16)));



        //Print the CO2 value into LCD buffer
        sprintf(dsp_buff1, "CO2: %d", CO2);
        sprintf(dsp_buff2, "Temp: %d", Temp);
        //Print the humidity value into LCD buffer
        sprintf(dsp_buff3, "Relative Hum: %d", Rh);
        //Update the 3 line messages into the LCD buffer
        update_lcd_dog();

        //Where it will transmit the entire array of strings to display in TeraTerm or
           Termite
        char * inputUSART3DataDisplay[] = {dsp_buff1, "    ", dsp_buff2, "    ",
          dsp_buff3};

        //Loop through all the strings in the array
        for(int i = 0; i < MAX_INPUT_DISPLAY; i++){
            USART3_sendString(inputUSART3DataDisplay[i]);
        }
        _delay_ms(1000);  //Transmit the readings once every 1 second
    }
}
```

```
/*
 * USART3_asynch_transmit.h
 *
 * Created: 4/26/2022 8:06:04 PM
 *  Author: jason
 */


#ifndef USART3_ASYNCH_TRANSMIT_H_
#define USART3_ASYNCH_TRANSMIT_H_

#define F_CPU 4000000
#define USART3_BAUD_RATE(BAUD_RATE) ((float)(F_CPU * 64 / (16 *(float)BAUD_RATE)))  //Calculation of baud rate from data sheet
#include <avr/io.h>
#include <util/delay.h>
#include <string.h>

//Header functions
void USART3_sendChar(char c);
void USART3_init (uint16_t, uint8_t, unsigned char);
void USART_sw_write(char);


//A simple function to configure a USART might have a single parameter that
//specifies the desired baud rate. The function that you must write for this task
//goes further than that, it allows both baud rate and the frame format to be specified
void USART3_init (uint16_t baud, uint8_t data_bits, unsigned char parity){
    PORTB_DIR |= PIN0_bm;  //To transmit the data

    //Specify the baud rate value for the USART3
    USART3.BAUD = (uint16_t)USART3_BAUD_RATE(baud);

    //Initialize the data bits and the parity bits type
    USART3_CTRLC |= data_bits | parity;
    USART3.CTRLB |= USART_TXEN_bm;  //Enable USART transmitter
}


//To be able to send the string of characters
void USART3_sendString(char* input){
    for(size_t i = 0; i < strlen(input); i++){
        USART_sw_write(input[i]);
    }
}



//Function to be able to transmit characters
//to the TX pin and display on the Tera Term
void USART_sw_write(char c)
```

```
{
    //Poll until the  transmit buffer register are empty
    //when they contain data that has not been moved to
    //transmit shift register
    while (!(USART3.STATUS & USART_DREIF_bm))
    {
        ;
    }

    //Load data to transmit shift register and
    //output each of the bits serially to the TXD pin
    USART3.TXDATAL = c;
}



#endif /* USART3_ASYNCH_TRANSMIT_H_ */
```

```
/*
 * SCD41_AVR128_driver.h
 *
 * Created: 4/18/2022 12:58:29 AM
 *  Author: jason
 */


#ifndef SCD41_AVR128_DRIVER_H_
#define SCD41_AVR128_DRIVER_H_

#include <avr/io.h>
#define F_CPU 4000000
#include <util/delay.h>

//Function Prototypes that will be used
void I2C0_SCD41_init();
void SCD41_start_periodic_measurement(uint8_t, uint8_t, uint8_t);
void SCD41_stop_periodic_measurement(uint8_t, uint8_t, uint8_t);
void SCD41_read_measurement(uint8_t, uint8_t, uint8_t);
uint8_t SCD41_get_data_ready_status(uint8_t, uint8_t, uint8_t);
uint8_t sensirion_common_generate_crc(const uint8_t*, uint16_t);

//For computing the checksum
#define CRC8_POLYNOMIAL 0x31
#define CRC8_INIT 0xFF

#define I2CSLAVE_ADDR_WRITE 0xC4      // 110 0010 0    0xC4
#define I2CSLAVE_ADDR_READ 0xC5       // 110 0010 1    0xC5

//The least significant and most significant byte address for the start periodic
  function
#define ADDRESS_STARTPERIODIC_LSB 0xB1
#define ADDRESS_STARTPERIODIC_MSB 0x21

//The least significant and most significant byte address for the stop periodic
  function
#define ADDRESS_STOPPERIODIC_LSB 0x86
#define ADDRESS_STOPPERIODIC_MSB 0x3F

//The least significant and most significant byte address for the read measurement
  periodic function
#define ADDRESS_READMEASUR_LSB 0x05
#define ADDRESS_READMEASUR_MSB 0xEC

//The least significant and most significant byte address for the get data ready
  function
#define ADDRESS_GETDATAREADY_LSB 0xB8
#define ADDRESS_GETDATAREADY_MSB 0xE4


//For the get_data_ready_status function to get the data response value
```

```
uint8_t readDataStatusMSB;
uint8_t readDataStatusLSB;
uint16_t getDataStatusReadyResponse;

//For the get_measurement function to get the CO2 value plus the CRC
uint8_t readCO2MSB;
uint8_t readCO2LSB;
uint8_t readCO2CRC;
uint16_t getParseCO2;

//For the get_measurement function to get the temperature value plus the CRC
uint8_t readTempMSB;
uint8_t readTempLSB;
uint8_t readTempCRC;
uint32_t getParseTemp;

//For the get_measurement function to get the relative humidity value plus the CRC
uint8_t readRhMSB;
uint8_t readRhLSB;
uint8_t readRhCRC;
uint16_t getParseRh;

//Get the data status CRC
uint8_t readDataStatusCRC;

//Also another way of storing the bytes by putting in an array
uint8_t storedCO2[2];
uint8_t storedTemp[2];
uint8_t storedRH[2];


//Initializes the AVR128DB48's I2C to communicate with the MCP23017.
//The bit transfer rate between the AVR128DB48 and the MCP23017 must be
//as fast as possible, but less than or equal to 100 kb/s.
void I2C0_SCD41_init()
{
    //Baud rate for the I2C which set to 15 assuming that is the fastest you can get to
    TWI0.MBAUD = 15;
    //Enable for the I2C Master
    TWI0.MCTRLA = TWI_ENABLE_bm;

    //Force the I2C to the idle state
    TWI0.MSTATUS = TWI_BUSSTATE_IDLE_gc;
}

//Starts the periodic measurement, signal update interval is 5 seconds
void SCD41_start_periodic_measurement(uint8_t SCD41_address, uint8_t SCD41_MSB,
  uint8_t SCD41_LSB){
    //To write the address of SCD41 (0x62) except also write operation so 110 0010 0
    TWI0_MADDR = SCD41_address;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));
```

```
    //To write the most significant byte
    TWI0_MDATA = SCD41_MSB;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));

    //To write the least significant byte
    TWI0_MDATA = SCD41_LSB;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));

    //Execute acknowledge action followed by issuing a stop condition
    TWI0_MCTRLB = TWI_MCMD_STOP_gc;
}


//This function is what stops the periodic measurement to change the sensor configuration or to save
//power. Note that the sensor will only respond to other commands after waiting 500 ms after issuing the
//stop_periodic_measurement command
void SCD41_stop_periodic_measurement(uint8_t SCD41_address, uint8_t SCD41_MSB, uint8_t SCD41_LSB){
    //To write the address of SCD41 (0x62) except also write operation so 110 0010 0
    TWI0_MADDR = SCD41_address;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));

    //To write the most significant byte
    TWI0_MDATA = SCD41_MSB;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));

    //To write the least significant byte
    TWI0_MDATA = SCD41_LSB;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));
    _delay_ms(500);   //Delay for 500 ms;

    //Execute acknowledge action followed by issuing a stop condition
    TWI0_MCTRLB = TWI_MCMD_STOP_gc;
}

//Function to read the measurement value for the temperature, relative humidity and CO2 of the SCD41
void SCD41_read_measurement(uint8_t SCD41_address, uint8_t SCD41_MSB, uint8_t SCD41_LSB){
    //To write the address of SCD41 (0x62) except also write operation so 110 0010 0
    TWI0_MADDR = SCD41_address;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));

    //----------------------------------------

    //To write the most significant byte command
    TWI0_MDATA = SCD41_MSB;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));
```

```
    //To write the least significant byte command
    TWI0_MDATA = SCD41_LSB;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));
    _delay_ms(1); //Delay for 1 ms

    //To write the I2C slave address which would then indicate reading from the slave
      to the master
    TWI0_MADDR = I2CSLAVE_ADDR_READ; //SCD41_address read;

    //-----------------------------------------------------------------
    //CO2
    //To start reading from the slave the Data_MSB of CO2
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readCO2MSB = TWI0_MDATA;
    storedCO2[0] = readCO2MSB;
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;

    //Poll until there's something to read from the slave: the Data_LSB of CO2
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readCO2LSB = TWI0_MDATA;
    storedCO2[1] = readCO2LSB;
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;

    //Concatenate the MSB and LSB of CO2 together for 16 bits altogether
    getParseCO2 = ((storedCO2[0] << 8) | storedCO2[1]);

    //getParseCO2 = (getParseCO2 & ~(0b11111111 << 0)) | ((readCO2LSB & 0b11111111) <<
       0);
    //getParseCO2 |= (getParseCO2 & ~(0b11111111 << 8)) | ((readCO2MSB & 0b11111111)
      << 8);


    //Poll until there's something to read from the slave: the CRC of CO2 which isn't
      necessary to read for lab 10
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readCO2CRC = TWI0_MDATA;
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;

    //-----------------------------------------------------------------------------
      -----------------

    //TEMPERATURE
    //Poll until there's something to read from the slave: the Most significant byte
      of the temperature
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readTempMSB = TWI0_MDATA;
    storedTemp[0] = readTempMSB;
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;    //Send ACK with a
      restart

    //Poll until there's something to read from the slave: the least significant byte
      of the temperature
```

```
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readTempLSB = TWI0_MDATA;
    storedTemp[1] = readTempLSB;

    //Concatenate the MSB and LSB of Temperature together for 16 bits altogether
    getParseTemp = ((uint16_t)storedTemp[0] << 8) | storedTemp[1];
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;    //Send ACK with a
      restart

    //Read modify write to parse each byte of the two bytes into the 16 bit field for
      the temperature
    //getParseTemp = (getParseTemp & ~(0b11111111 << 0)) | ((readTempLSB & 0b11111111)
       << 0);
    //getParseTemp |= (getParseTemp & ~(0b11111111 << 8)) | ((readTempMSB &
      0b11111111) << 8);

    //Poll to read the CRC of temperature which isn't necessary to read for lab 10
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readTempCRC = TWI0_MDATA;
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;    //Send ACK with a
      restart

    //-------------------------------------------------------------------------------
      -----------------
    //RELATIVE HUMIDITY
    //Poll until there's something to read from the slave: the Most significant byte
      of the relative humidity (RH)
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readRhMSB = TWI0_MDATA;         //Read the MSB of Rh
    storedRH[0] = readRhMSB;
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;    //Send ACK with a
      restart

    //Poll until there's something to read from the slave: the least significant byte
      of the relative humidity (RH)
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readRhLSB = TWI0_MDATA;         //Read the LSB of Rh
    storedTemp[1] = readRhLSB;

    //Concatenate the MSB and LSB of RH together for 16 bits altogether
    getParseRh = (storedRH[0] << 8) | storedRH[1];
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;    //Send ACK with a
      restart

    //Read modify write to parse each byte of the two bytes into the 16 bit field for
      the relative humidity (RH)
    //getParseRh = (getParseRh & ~(0b11111111 << 0)) | ((readRhLSB & 0b11111111) <<
      0);
    //getParseRh |= (getParseRh & ~(0b11111111 << 8)) | ((readRhMSB & 0b11111111) <<
      8);
```

```
    //Poll to read the CRC of relative humidity (RH) which isn't necessary to read for lab 10
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readRhCRC = TWI0_MDATA;
    //Master send to slave to stop reading data by sending a NACK response
    TWI0_MCTRLB = TWI_MCMD_STOP_gc | TWI_ACKACT_NACK_gc;

}


//Check if data is ready to be read from the SCD41
uint8_t SCD41_get_data_ready_status(uint8_t SCD41_address, uint8_t SCD41_MSB, uint8_t SCD41_LSB){
    //Poll to write the address of SCD41 (0x62) except also write operation so 110 0010 0
    TWI0_MADDR = SCD41_address;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));

    //Poll To write the most significant byte command: 0xE4
    TWI0_MDATA = SCD41_MSB;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));

    //Poll To write the least significant byte command: 0xB8
    TWI0_MDATA = SCD41_LSB;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));
    _delay_ms(1);       //Wait 1 ms after sending the command

    //To write the I2C slave address which would then indicate reading from the slave to the master
    TWI0_MADDR = I2CSLAVE_ADDR_READ; //SCD41_address;

    //Poll until there's something to read from the slave: the Data_MSB
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readDataStatusMSB = TWI0_MDATA;
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;       //Send ACK with a restart

    //Poll until there's something to read from the slave: the Data_LSB
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readDataStatusLSB = TWI0_MDATA;
    //Concatenate the MSB and LSB of data together to form 16 bits
    getDataStatusReadyResponse = (readDataStatusMSB << 8) | readDataStatusLSB;
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;   //Send ACK with a restart

    //16 bit data status result
    //getDataStatusReadyResponse = (getDataStatusReadyResponse & ~(0b11111111 << 0)) |((readDataStatusLSB & 0b11111111) << 0);
    //getDataStatusReadyResponse |= (getDataStatusReadyResponse & ~(0b11111111 << 8)) | ((readDataStatusMSB & 0b11111111) << 8);

    //Poll until there's something to read from the slave: the CRC of
```

```
      data_ready_status value which we don't need for lab 10
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    //Master send to slave to stop reading data by sending a NACK response
    readDataStatusCRC = TWI0_MDATA;

    //Stop having the Master to read the slave data
    TWI0_MCTRLB = TWI_MCMD_STOP_gc | TWI_ACKACT_NACK_gc;

    //The case if the LSB 11 bits are not all 0's meaning data is ready
    if(getDataStatusReadyResponse & 0x7FF){
        return 1;
    }

    //Else go there meaning that all the LSB 11 bits are all 0s meaning data is not
      ready
    return 0;
}



//This is what is responsible for computing the checksum
uint8_t sensirion_common_generate_crc(const uint8_t* data, uint16_t count) {
    uint16_t current_byte;
    uint8_t crc = CRC8_INIT;
    uint8_t crc_bit;
    /* calculates 8-Bit checksum with given polynomial */
    for (current_byte = 0; current_byte < count; ++current_byte) {
        crc ^= (data[current_byte]);
        for (crc_bit = 8; crc_bit > 0; --crc_bit) {
            if (crc & 0x80)
            crc = (crc << 1) ^ CRC8_POLYNOMIAL;
            else
            crc = (crc << 1);
        }
    }
    return crc;
}

#endif /* SCD41_AVR128_DRIVER_H_ */
```

DESIGN TASK 2

```
/*
* lcd_dog_AVR128_driver.h
*
* Created: 3/19/2022 8:09:13 PM
* Author : jason
*/

#include <avr/io.h>
#include <stdio.h>

/*
Is what is responsible for transmitting the command for the LCD
*/
void lcd_spi_transmit_CMD (unsigned char cmd);

/*
Is what is responsible for transmitting the data for the LCD into what is to be displayed
*/
void lcd_spi_transmit_DATA (unsigned char cmd);

/*
Initialize the SPI LCD to being blank
*/
void init_spi_lcd (void);

/*
Initialize the LCD Dog to being blank
*/
void init_lcd_dog (void);

/*
Function prototype for 40 ms
*/
void delay_40mS(void);

/*
Function prototype for 40 microsecond
*/
void delay_30uS(void);

/*
To update on the LCD for something to be displayed
*/
void update_lcd_dog(void);


// Display buffer for DOG LCD using sprintf()
char dsp_buff1[17];
char dsp_buff2[17];
char dsp_buff3[17];
```

```
void lcd_spi_transmit_CMD (unsigned char cmd) {
    //Poll until ready to send the command
    //while(!(SPI0_INTFLAGS & SPI_IF_bm)){}
    PORTC_OUT &= ~PIN0_bm;  //Clear PC0 = RS = 0 = command
    PORTA_OUT &= ~PIN7_bm;  //clear PA7 = /SS = selected
    SPI0_DATA = cmd;
    //Poll until ready to send the command
    while(!(SPI0_INTFLAGS & SPI_IF_bm)){}
    PORTA_OUT |= PIN7_bm;   //clear PA7 = /SS = selected

}

void lcd_spi_transmit_DATA (unsigned char cmd) {
    //Poll until ready to send the command
    while(!(SPI0_INTFLAGS & SPI_IF_bm)){}
    PORTC_OUT |= PIN0_bm;   //PC0 = RS = 1 = command
    PORTA_OUT &= ~PIN7_bm;  //clear PA7 = /SS = selected
    SPI0_DATA = cmd;
    //Poll until ready to send the command
    while(!(SPI0_INTFLAGS & SPI_IF_bm)){}
    PORTA_OUT |= PIN7_bm;   //clear PA7 = /SS = selected

}

void init_spi_lcd (void) {
    PORTA_DIR |= PIN4_bm | PIN6_bm | PIN7_bm;  //Set MOSI, SCK and //SS as output
      while MISO as input

    PORTC_DIR |= PIN0_bm;  //Set RS of LCD as output

    SPI0_CTRLA |= SPI_ENABLE_bm | SPI_MASTER_bm; //Enable the SPI and make it in the
      Master Mode

    SPI0_CTRLB |= SPI_SSD_bm | SPI_MODE1_bm | SPI_MODE0_bm;   //Put the SPI with slave
       select (/SS) to be enabled and be in SPI Mode 3 (CPOL = 1 and CPHA = 1)

    //Wait to clears the IF flag in the INTFLAG meaning there no serial data yet to be
       transferred
    //while(SPI0_INTFLAGS & SPI_IF_bm){}
    PORTC_OUT &= ~PIN0_bm;  //PC0 = RS = 0 = command



}


void init_lcd_dog (void) {

    init_spi_lcd();     //Initialize mcu for LCD SPI
```

```
    //start_dly_40ms:
    delay_40mS();    //startup delay.


    //func_set1:
    lcd_spi_transmit_CMD(0x39);   // sedn function set #1
    delay_30uS();   //delay for command to be processed


    //func_set2:
    lcd_spi_transmit_CMD(0x39); //send fuction set #2
    delay_30uS();   //delay for command to be processed


    //bias_set:
    lcd_spi_transmit_CMD(0x1E); //set bias value.
    delay_30uS();   //delay for command to be processed


    //power_ctrl:
    lcd_spi_transmit_CMD(0x55); //~ 0x50 nominal for 5V
    //~ 0x55 for 3.3V (delicate adjustment).
    delay_30uS();   //delay for command to be processed


    //follower_ctrl:
    lcd_spi_transmit_CMD(0x6C); //follower mode on...
    delay_40mS();   //delay for command to be processed


    //contrast_set:
    lcd_spi_transmit_CMD(0x7F); //~ 77 for 5V, ~ 7F for 3.3V
    delay_30uS();   //delay for command to be processed


    //display_on:
    lcd_spi_transmit_CMD(0x0c); //display on, cursor off, blink off
    delay_30uS();   //delay for command to be processed


    //clr_display:
    lcd_spi_transmit_CMD(0x01); //clear display, cursor home
    delay_30uS();   //delay for command to be processed


    //entry_mode:
    lcd_spi_transmit_CMD(0x06); //clear display, cursor home
    delay_30uS();   //delay for command to be processed
}


void delay_40mS(void) {
```

```
    int i;
    for (int n = 40; n > 0; n--)
    for (i = 0; i < 800; i++)
    __asm("nop");
}

void delay_30uS(void) {
    int i;
    for (int n = 1; n > 0; n--)
    for (i = 0; i < 2; i++)
    __asm("nop");
}


// Updates the LCD display lines 1, 2, and 3, using the
// contents of dsp_buff_1, dsp_buff_2, and dsp_buff_3, respectively.
void update_lcd_dog(void) {

    init_spi_lcd();     //init SPI port for LCD.

    // send line 1 to the LCD module.
    lcd_spi_transmit_CMD(0x80); //init DDRAM addr-ctr
    delay_30uS();
    for (int i = 0; i < 16; i++) {
        lcd_spi_transmit_DATA(dsp_buff1[i]);
        delay_30uS();
    }

    // send line 2 to the LCD module.
    lcd_spi_transmit_CMD(0x90); //init DDRAM addr-ctr
    delay_30uS();
    for (int i = 0; i < 16; i++) {
        lcd_spi_transmit_DATA(dsp_buff2[i]);
        delay_30uS();
    }

    // send line 3 to the LCD module.
    lcd_spi_transmit_CMD(0xA0); //init DDRAM addr-ctr
    delay_30uS();
    for (int i = 0; i < 16; i++) {
        lcd_spi_transmit_DATA(dsp_buff3[i]);
        delay_30uS();
    }
}
```

```c
/*
 * SCD41_Multimodule_LED_CO2_Level_Lab11Task2.c
 *
 * Created: 4/26/2022 9:15:52 PM
 * Author : jason
 */

#include <avr/io.h>
#include <math.h>
#define F_CPU 4000000
#include "lcd_dog_AVR128_driver.h"
#include "SCD41_AVR128_driver.h"
#include "USART3_asynch_transmit.h"
#include "MCP23017_CO2_level_LED_simple_display.h"
#include <util/delay.h>

#define MAX_INPUT_DISPLAY 5


uint16_t CO2;
uint16_t Temp;
uint16_t Rh;

uint16_t baudRate = 9600;   //For the baud rate of USART3
uint8_t dataBits = USART_CHSIZE_8BIT_gc;    //For the (character size)  CHSIZE[2:0]
unsigned char parity = 0x00;    //PMODE[1:0]


int main(void)
{
    init_lcd_dog();      //Initialize the buffer of the LCD

    //Function to initialize the USART3 baud rate, data bit and parity
    USART3_init(baudRate, dataBits, parity);

    while (1)
    {
        I2C0_SCD41_init();   //Initializes  the AVR128DB48 I2C0 to communicate with
          SCD41 to change its BAUD RATE for SCD31
        SCD41_start_periodic_measurement(I2CSLAVE_ADDR_WRITE,
          ADDRESS_STARTPERIODIC_MSB, ADDRESS_STARTPERIODIC_LSB);

        //Keep polling until data is ready which can be measured
        while(!SCD41_get_data_ready_status(I2CSLAVE_ADDR_WRITE,
          ADDRESS_GETDATAREADY_MSB, ADDRESS_GETDATAREADY_LSB));

        SCD41_read_measurement(I2CSLAVE_ADDR_WRITE, ADDRESS_READMEASUR_MSB,
          ADDRESS_READMEASUR_LSB);

        CO2 = getParseCO2;
        Temp = -45 + ( (175 * getParseTemp) / (pow(2, 16)));
        Rh = 100 * (((float)getParseRh) / (pow(2, 16)));
```

```
        //Print the CO2 value into LCD buffer
        sprintf(dsp_buff1, "CO2: %d", CO2);
        sprintf(dsp_buff2, "Temp: %d", Temp);
        //Print the humidity value into LCD buffer
        sprintf(dsp_buff3, "Relative Hum: %d", Rh);
        //Update the 3 line messages into the LCD buffer
        update_lcd_dog();

        I2C0_MCP23017_init(); //Initializes  the AVR128DB48 I2C0 to communicate with
          MCP23017 to change its BAUD RATE for MCP23017
        MCP23017_I2C_init();  //Initializes  GPIO (GPB) as outputs for outputting to
          the LEDS

        if(CO2 >= 400 && CO2 <= 499){
            MCP23017_I2C_write(WRITE_opcode, OLATBaddr_b1, 0x7F); //0111 1111 GPB7-
              GPB0
        }
        else if(CO2 >= 500 && CO2 <= 599){
            MCP23017_I2C_write(WRITE_opcode, OLATBaddr_b1, 0x3F); //0011 1111 GPB7-
              GPB0
        }
        else if(CO2 >= 600 && CO2 <= 699){
            MCP23017_I2C_write(WRITE_opcode, OLATBaddr_b1, 0x1F); //0001 1111 GPB7-
              GPB0
        }
        else if(CO2 >= 700 && CO2 <= 799){
            MCP23017_I2C_write(WRITE_opcode, OLATBaddr_b1, 0x0F); //0000 1111 GPB7-
              GPB0
        }
        else if(CO2 >= 800 && CO2 <= 899){
            MCP23017_I2C_write(WRITE_opcode, OLATBaddr_b1, 0x07); //0000 0111 GPB7-
              GPB0
        }
        else if(CO2 >= 900 && CO2 <= 999){
            MCP23017_I2C_write(WRITE_opcode, OLATBaddr_b1, 0x03); //0000 0011 GPB7-
              GPB0
        }
        else if(CO2 >= 1000 && CO2 <= 1099){
            MCP23017_I2C_write(WRITE_opcode, OLATBaddr_b1, 0x01); //0000 0001 GPB7-
              GPB0
        }
        else if(CO2 >= 1100 && CO2 <= 1199){
            MCP23017_I2C_write(WRITE_opcode, OLATBaddr_b1, 0x00); //0000 0000 GPB7-
              GPB0
        }

        //Where it will transmit the entire array of strings to display in TeraTerm or
           Termite
        char *inputUSART3DataDisplay[] = {dsp_buff1, "    ", dsp_buff2, "    ",
```

```
          dsp_buff3};

        //Loop through all the strings in the array
        for(int i = 0; i < MAX_INPUT_DISPLAY; i++){
            USART3_sendString(inputUSART3DataDisplay[i]);
        }
        USART3_sendString("\r\n");  //For new line of the Teraterm or Termite
        _delay_ms(1000);  //Transmit the readings once every 1 second
    }
}
```

```c
/*
 * MCP23017_CO2_level_LED_simple_display.h
 *
 * Created: 4/26/2022 9:25:26 PM
 *  Author: jason
 */


#ifndef MCP23017_CO2_LEVEL_LED_SIMPLE_DISPLAY_H_
#define MCP23017_CO2_LEVEL_LED_SIMPLE_DISPLAY_H_


#include <avr/io.h>
#define F_CPU 4000000     // 4MHz default clock
#include <util/delay.h>

//Defines for GIO, b1 means when 8-bit mode, BANK = 1
#define IOCONaddr_b0 0x0A   //address at reset, default 16-bit mode
#define IOCONaddr_b1 0x05   //Control registers
#define IODIRAaddr_b1 0x00  //Directional register for PORTA
#define IODIRBaddr_b1 0x10  //Directional register for PORTB
#define GPPUAaddr_b1 0x06
#define GPIOAaddr_b1 0x09   //GPIOA I/O PORT register
#define OLATBaddr_b1 0x1A   //GPIO output registers
#define WRITE_opcode 0x40
#define READ_opcode 0x41

//Function Prototypes that will be used
void I2C0_MCP23017_init();
void MCP23017_I2C_init();
void MCP23017_I2C_write(uint8_t, uint8_t, uint8_t);



//Initializes the AVR128DB48's I2C0 to commmunicate with the MCP23017.
//The bit transfer rate between the AVR128DB48 and the MCP23017 must be
//as fast as possible, but less than or equal to 400 kb/s.
void I2C0_MCP23017_init()
{
    //Baud rate for the I2C which set to 0 assuming that is the fastest you can get to
    TWI0.MBAUD = 0;


    //Enable for the I2C Master
    TWI0.MCTRLA = TWI_ENABLE_bm;

    //Force the I2C to the idle state
    TWI0.MSTATUS = TWI_BUSSTATE_IDLE_gc;
}

//This function initializes the MCP23017. Port A of the GPIO (GPA) must be
//configured as all inputs with pull ups enabled. GPB must be
```

```
//configured as all outputs.
void MCP23017_I2C_init(){

    MCP23017_I2C_write(WRITE_opcode, IOCONaddr_b0, 0xA0);

    //Configure PORTB of the GPIO MCP23017 as the output
    MCP23017_I2C_write(WRITE_opcode, IODIRBaddr_b1, 0x00);
}


//This function is what write to the actual GPIO expander MCP23017
//to access the registers and modifying those bit fields
void MCP23017_I2C_write(uint8_t opcode, uint8_t address, uint8_t data){
    TWI0_MADDR = opcode;   //Pass the opcode to the address

    //Poll until master transmit address of byte write operation is complete ↵
      regardless
    while(!(TWI0.MSTATUS & TWI_WIF_bm));

    TWI0_MDATA = address; //Pass the address to master data

    //Poll until master transmit address of byte write operation is complete ↵
      regardless
    while(!(TWI0.MSTATUS & TWI_WIF_bm));

    //Pass the data to master data
    TWI0_MDATA = data;

    //Poll until master transmit address of byte write operation is complete ↵
      regardless
    while(!(TWI0.MSTATUS & TWI_WIF_bm));


    //Execute acknowledge action followed by issuing a stop condition
    TWI0_MCTRLB = TWI_MCMD_STOP_gc;
}



#endif /* MCP23017_CO2_LEVEL_LED_SIMPLE_DISPLAY_H_ */
```

```
/*
 * SCD41_AVR128_driver.h
 *
 * Created: 4/18/2022 12:58:29 AM
 *  Author: jason
 */


#ifndef SCD41_AVR128_DRIVER_H_
#define SCD41_AVR128_DRIVER_H_

#include <avr/io.h>
#define F_CPU 4000000
#include <util/delay.h>

//Function Prototypes that will be used
void I2C0_SCD41_init();
void SCD41_start_periodic_measurement(uint8_t, uint8_t, uint8_t);
void SCD41_stop_periodic_measurement(uint8_t, uint8_t, uint8_t);
void SCD41_read_measurement(uint8_t, uint8_t, uint8_t);
uint8_t SCD41_get_data_ready_status(uint8_t, uint8_t, uint8_t);
uint8_t sensirion_common_generate_crc(const uint8_t*, uint16_t);

//For computing the checksum
#define CRC8_POLYNOMIAL 0x31
#define CRC8_INIT 0xFF

#define I2CSLAVE_ADDR_WRITE 0xC4      // 110 0010 0    0xC4
#define I2CSLAVE_ADDR_READ 0xC5       // 110 0010 1    0xC5

//The least significant and most significant byte address for the start periodic
  function
#define ADDRESS_STARTPERIODIC_LSB 0xB1
#define ADDRESS_STARTPERIODIC_MSB 0x21

//The least significant and most significant byte address for the stop periodic
  function
#define ADDRESS_STOPPERIODIC_LSB 0x86
#define ADDRESS_STOPPERIODIC_MSB 0x3F

//The least significant and most significant byte address for the read measurement
  periodic function
#define ADDRESS_READMEASUR_LSB 0x05
#define ADDRESS_READMEASUR_MSB 0xEC

//The least significant and most significant byte address for the get data ready
  function
#define ADDRESS_GETDATAREADY_LSB 0xB8
#define ADDRESS_GETDATAREADY_MSB 0xE4


//For the get_data_ready_status function to get the data response value
```

```
uint8_t readDataStatusMSB;
uint8_t readDataStatusLSB;
uint16_t getDataStatusReadyResponse;

//For the get_measurement function to get the CO2 value plus the CRC
uint8_t readCO2MSB;
uint8_t readCO2LSB;
uint8_t readCO2CRC;
uint16_t getParseCO2;

//For the get_measurement function to get the temperature value plus the CRC
uint8_t readTempMSB;
uint8_t readTempLSB;
uint8_t readTempCRC;
uint32_t getParseTemp;

//For the get_measurement function to get the relative humidity value plus the CRC
uint8_t readRhMSB;
uint8_t readRhLSB;
uint8_t readRhCRC;
uint16_t getParseRh;

//Get the data status CRC
uint8_t readDataStatusCRC;

//Also another way of storing the bytes by putting in an array
uint8_t storedCO2[2];
uint8_t storedTemp[2];
uint8_t storedRH[2];


//Initializes the AVR128DB48's I2C to communicate with the MCP23017.
//The bit transfer rate between the AVR128DB48 and the MCP23017 must be
//as fast as possible, but less than or equal to 100 kb/s.
void I2C0_SCD41_init()
{
    //Baud rate for the I2C which set to 15 assuming that is the fastest you can get to
    TWI0.MBAUD = 15;
    //Enable for the I2C Master
    TWI0.MCTRLA = TWI_ENABLE_bm;

    //Force the I2C to the idle state
    TWI0.MSTATUS = TWI_BUSSTATE_IDLE_gc;
}

//Starts the periodic measurement, signal update interval is 5 seconds
void SCD41_start_periodic_measurement(uint8_t SCD41_address, uint8_t SCD41_MSB,
  uint8_t SCD41_LSB){
    //To write the address of SCD41 (0x62) except also write operation so 110 0010 0
    TWI0_MADDR = SCD41_address;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));
```

```
    //To write the most significant byte
    TWI0_MDATA = SCD41_MSB;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));

    //To write the least significant byte
    TWI0_MDATA = SCD41_LSB;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));

    //Execute acknowledge action followed by issuing a stop condition
    TWI0_MCTRLB = TWI_MCMD_STOP_gc;
}


//This function is what stops the periodic measurement to change the sensor configuration or to save
//power. Note that the sensor will only respond to other commands after waiting 500 ms after issuing the
//stop_periodic_measurement command
void SCD41_stop_periodic_measurement(uint8_t SCD41_address, uint8_t SCD41_MSB, uint8_t SCD41_LSB){
    //To write the address of SCD41 (0x62) except also write operation so 110 0010 0
    TWI0_MADDR = SCD41_address;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));

    //To write the most significant byte
    TWI0_MDATA = SCD41_MSB;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));

    //To write the least significant byte
    TWI0_MDATA = SCD41_LSB;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));
    _delay_ms(500);   //Delay for 500 ms;

    //Execute acknowledge action followed by issuing a stop condition
    TWI0_MCTRLB = TWI_MCMD_STOP_gc;
}

//Function to read the measurement value for the temperature, relative humidity and CO2 of the SCD41
void SCD41_read_measurement(uint8_t SCD41_address, uint8_t SCD41_MSB, uint8_t SCD41_LSB){
    //To write the address of SCD41 (0x62) except also write operation so 110 0010 0
    TWI0_MADDR = SCD41_address;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));

    //----------------------------------------

    //To write the most significant byte command
    TWI0_MDATA = SCD41_MSB;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));
```

```
    //To write the least significant byte command
    TWI0_MDATA = SCD41_LSB;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));
    _delay_ms(1); //Delay for 1 ms

    //To write the I2C slave address which would then indicate reading from the slave
      to the master
    TWI0_MADDR = I2CSLAVE_ADDR_READ; //SCD41_address read;

    //-----------------------------------------------------------------
    //CO2
    //To start reading from the slave the Data_MSB of CO2
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readCO2MSB = TWI0_MDATA;
    storedCO2[0] = readCO2MSB;
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;

    //Poll until there's something to read from the slave: the Data_LSB of CO2
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readCO2LSB = TWI0_MDATA;
    storedCO2[1] = readCO2LSB;
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;

    //Concatenate the MSB and LSB of CO2 together for 16 bits altogether
    getParseCO2 = ((storedCO2[0] << 8) | storedCO2[1]);

    //getParseCO2 = (getParseCO2 & ~(0b11111111 << 0)) | ((readCO2LSB & 0b11111111) <<
       0);
    //getParseCO2 |= (getParseCO2 & ~(0b11111111 << 8)) | ((readCO2MSB & 0b11111111)
      << 8);


    //Poll until there's something to read from the slave: the CRC of CO2 which isn't
      necessary to read for lab 10
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readCO2CRC = TWI0_MDATA;
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;

    //-----------------------------------------------------------------------------
      -----------------

    //TEMPERATURE
    //Poll until there's something to read from the slave: the Most significant byte
      of the temperature
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readTempMSB = TWI0_MDATA;
    storedTemp[0] = readTempMSB;
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;    //Send ACK with a
      restart

    //Poll until there's something to read from the slave: the least significant byte
      of the temperature
```

```
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readTempLSB = TWI0_MDATA;
    storedTemp[1] = readTempLSB;

    //Concatenate the MSB and LSB of Temperature together for 16 bits altogether
    getParseTemp = ((uint16_t)storedTemp[0] << 8) | storedTemp[1];
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;    //Send ACK with a
      restart

    //Read modify write to parse each byte of the two bytes into the 16 bit field for
      the temperature
    //getParseTemp = (getParseTemp & ~(0b11111111 << 0)) | ((readTempLSB & 0b11111111)
       << 0);
    //getParseTemp |= (getParseTemp & ~(0b11111111 << 8)) | ((readTempMSB &
      0b11111111) << 8);

    //Poll to read the CRC of temperature which isn't necessary to read for lab 10
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readTempCRC = TWI0_MDATA;
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;    //Send ACK with a
      restart

    //-------------------------------------------------------------------------------
      -----------------
    //RELATIVE HUMIDITY
    //Poll until there's something to read from the slave: the Most significant byte
      of the relative humidity (RH)
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readRhMSB = TWI0_MDATA;         //Read the MSB of Rh
    storedRH[0] = readRhMSB;
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;    //Send ACK with a
      restart

    //Poll until there's something to read from the slave: the least significant byte
      of the relative humidity (RH)
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readRhLSB = TWI0_MDATA;         //Read the LSB of Rh
    storedTemp[1] = readRhLSB;

    //Concatenate the MSB and LSB of RH together for 16 bits altogether
    getParseRh = (storedRH[0] << 8) | storedRH[1];
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;    //Send ACK with a
      restart

    //Read modify write to parse each byte of the two bytes into the 16 bit field for
      the relative humidity (RH)
    //getParseRh = (getParseRh & ~(0b11111111 << 0)) | ((readRhLSB & 0b11111111) <<
      0);
    //getParseRh |= (getParseRh & ~(0b11111111 << 8)) | ((readRhMSB & 0b11111111) <<
      8);
```

```
    //Poll to read the CRC of relative humidity (RH) which isn't necessary to read for lab 10
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readRhCRC = TWI0_MDATA;
    //Master send to slave to stop reading data by sending a NACK response
    TWI0_MCTRLB = TWI_MCMD_STOP_gc | TWI_ACKACT_NACK_gc;

}


//Check if data is ready to be read from the SCD41
uint8_t SCD41_get_data_ready_status(uint8_t SCD41_address, uint8_t SCD41_MSB, uint8_t SCD41_LSB){
    //Poll to write the address of SCD41 (0x62) except also write operation so 110 0010 0
    TWI0_MADDR = SCD41_address;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));

    //Poll To write the most significant byte command: 0xE4
    TWI0_MDATA = SCD41_MSB;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));

    //Poll To write the least significant byte command: 0xB8
    TWI0_MDATA = SCD41_LSB;
    while(!(TWI0_MSTATUS & TWI_WIF_bm));
    _delay_ms(1);       //Wait 1 ms after sending the command

    //To write the I2C slave address which would then indicate reading from the slave to the master
    TWI0_MADDR = I2CSLAVE_ADDR_READ; //SCD41_address;

    //Poll until there's something to read from the slave: the Data_MSB
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readDataStatusMSB = TWI0_MDATA;
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;       //Send ACK with a restart

    //Poll until there's something to read from the slave: the Data_LSB
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    readDataStatusLSB = TWI0_MDATA;
    //Concatenate the MSB and LSB of data together to form 16 bits
    getDataStatusReadyResponse = (readDataStatusMSB << 8) | readDataStatusLSB;
    TWI0_MCTRLB =  TWI_ACKACT_ACK_gc | TWI_MCMD_RECVTRANS_gc;   //Send ACK with a restart

    //16 bit data status result
    //getDataStatusReadyResponse = (getDataStatusReadyResponse & ~(0b11111111 << 0)) | ((readDataStatusLSB & 0b11111111) << 0);
    //getDataStatusReadyResponse |= (getDataStatusReadyResponse & ~(0b11111111 << 8)) | ((readDataStatusMSB & 0b11111111) << 8);

    //Poll until there's something to read from the slave: the CRC of
```

```
      data_ready_status value which we don't need for lab 10
    while(!(TWI0_MSTATUS & TWI_RIF_bm));
    //Master send to slave to stop reading data by sending a NACK response
    readDataStatusCRC = TWI0_MDATA;

    //Stop having the Master to read the slave data
    TWI0_MCTRLB = TWI_MCMD_STOP_gc | TWI_ACKACT_NACK_gc;

    //The case if the LSB 11 bits are not all 0's meaning data is ready
    if(getDataStatusReadyResponse & 0x7FF){
        return 1;
    }

    //Else go there meaning that all the LSB 11 bits are all 0s meaning data is not
      ready
    return 0;
}



//This is what is responsible for computing the checksum
uint8_t sensirion_common_generate_crc(const uint8_t* data, uint16_t count) {
    uint16_t current_byte;
    uint8_t crc = CRC8_INIT;
    uint8_t crc_bit;
    /* calculates 8-Bit checksum with given polynomial */
    for (current_byte = 0; current_byte < count; ++current_byte) {
        crc ^= (data[current_byte]);
        for (crc_bit = 8; crc_bit > 0; --crc_bit) {
            if (crc & 0x80)
            crc = (crc << 1) ^ CRC8_POLYNOMIAL;
            else
            crc = (crc << 1);
        }
    }
    return crc;
}

#endif /* SCD41_AVR128_DRIVER_H_ */
```

```
/*
 * USART3_asynch_transmit.h
 *
 * Created: 4/26/2022 8:06:04 PM
 *  Author: jason
 */


#ifndef USART3_ASYNCH_TRANSMIT_H_
#define USART3_ASYNCH_TRANSMIT_H_

#define F_CPU 4000000
#define USART3_BAUD_RATE(BAUD_RATE) ((float)(F_CPU * 64 / (16 *(float)BAUD_RATE)))  //
  Calculation of baud rate from data sheet
#include <avr/io.h>
#include <util/delay.h>
#include <string.h>

//Header functions
void USART3_sendChar(char c);
void USART3_init (uint16_t, uint8_t, unsigned char);
void USART_sw_write(char);


//A simple function to configure a USART might have a single parameter that
//specifies the desired baud rate. The function that you must write for this task
//goes further than that, it allows both baud rate and the frame format to be
  specified
void USART3_init (uint16_t baud, uint8_t data_bits, unsigned char parity){
    PORTB_DIR |= PIN0_bm;  //To transmit the data

    //Specify the baud rate value for the USART3
    USART3.BAUD = (uint16_t)USART3_BAUD_RATE(baud);

    //Initialize the data bits and the parity bits type
    USART3_CTRLC |= data_bits | parity;
    USART3.CTRLB |= USART_TXEN_bm;  //Enable USART transmitter
}


//To be able to send the string of characters
void USART3_sendString(char* input){
    for(size_t i = 0; i < strlen(input); i++){
        USART_sw_write(input[i]);
    }
}



//Function to be able to transmit characters
//to the TX pin and display on the Tera Term
void USART_sw_write(char c)
```

```
{
    //Poll until the  transmit buffer register are empty
    //when they contain data that has not been moved to
    //transmit shift register
    while (!(USART3.STATUS & USART_DREIF_bm))
    {
        ;
    }

    //Load data to transmit shift register and
    //output each of the bits serially to the TXD pin
    USART3.TXDATAL = c;
}



#endif /* USART3_ASYNCH_TRANSMIT_H_ */
```

DESIGN TASK 3

Here is a list of all functions, variables, defines, enums, and typedefs with links to the files they belong to:

- a -

- ADDRESS_GETDATAREADY_LSB : **SCD41_AVR128_driver.h**
- ADDRESS_GETDATAREADY_MSB : **SCD41_AVR128_driver.h**
- ADDRESS_READMEASUR_LSB : **SCD41_AVR128_driver.h**
- ADDRESS_READMEASUR_MSB : **SCD41_AVR128_driver.h**
- ADDRESS_STARTPERIODIC_LSB : **SCD41_AVR128_driver.h**
- ADDRESS_STARTPERIODIC_MSB : **SCD41_AVR128_driver.h**
- ADDRESS_STOPPERIODIC_LSB : **SCD41_AVR128_driver.h**
- ADDRESS_STOPPERIODIC_MSB : **SCD41_AVR128_driver.h**

- b -

- baudRate : **main.c**

- c -

- CO2 : **main.c**
- CRC8_INIT : **SCD41_AVR128_driver.h**
- CRC8_POLYNOMIAL : **SCD41_AVR128_driver.h**

- d -

- dataBits : **main.c**
- delay_30uS() : **lcd_dog_AVR128_driver.h**
- delay_40mS() : **lcd_dog_AVR128_driver.h**
- dsp_buff1 : **lcd_dog_AVR128_driver.h**
- dsp_buff2 : **lcd_dog_AVR128_driver.h**
- dsp_buff3 : **lcd_dog_AVR128_driver.h**

- f -

- F_CPU : **main.c**, **MCP23017_CO2_level_LED_simple_display.h**, **SCD41_AVR128_driver.h**, **USART3_asynch_transmit.h**

- g -

- getDataStatusReadyResponse : **SCD41_AVR128_driver.h**
- getParseCO2 : **SCD41_AVR128_driver.h**
- getParseRh : **SCD41_AVR128_driver.h**
- getParseTemp : **SCD41_AVR128_driver.h**
- GPIOAaddr_b1 : **MCP23017_CO2_level_LED_simple_display.h**
- GPPUAaddr_b1 : **MCP23017_CO2_level_LED_simple_display.h**

- i -

- I2C0_MCP23017_init() : **MCP23017_CO2_level_LED_simple_display.h**
- I2C0_SCD41_init() : **SCD41_AVR128_driver.h**
- I2CSLAVE_ADDR_READ : **SCD41_AVR128_driver.h**
- I2CSLAVE_ADDR_WRITE : **SCD41_AVR128_driver.h**
- init_lcd_dog() : **lcd_dog_AVR128_driver.h**
- init_spi_lcd() : **lcd_dog_AVR128_driver.h**
- IOCONaddr_b0 : **MCP23017_CO2_level_LED_simple_display.h**
- IOCONaddr_b1 : **MCP23017_CO2_level_LED_simple_display.h**
- IODIRAaddr_b1 : **MCP23017_CO2_level_LED_simple_display.h**
- IODIRBaddr_b1 : **MCP23017_CO2_level_LED_simple_display.h**

- l -

- lcd_spi_transmit_CMD() : **lcd_dog_AVR128_driver.h**
- lcd_spi_transmit_DATA() : **lcd_dog_AVR128_driver.h**

- m -

- main() : **main.c**
- MAX_INPUT_DISPLAY : **main.c**
- MCP23017_I2C_init() : **MCP23017_CO2_level_LED_simple_display.h**
- MCP23017_I2C_write() : **MCP23017_CO2_level_LED_simple_display.h**

- o -

- OLATBaddr_b1 : **MCP23017_CO2_level_LED_simple_display.h**

- p -

- parity : **main.c**

- r -

- READ_opcode : **MCP23017_CO2_level_LED_simple_display.h**
- readCO2CRC : **SCD41_AVR128_driver.h**
- readCO2LSB : **SCD41_AVR128_driver.h**
- readCO2MSB : **SCD41_AVR128_driver.h**
- readDataStatusCRC : **SCD41_AVR128_driver.h**
- readDataStatusLSB : **SCD41_AVR128_driver.h**
- readDataStatusMSB : **SCD41_AVR128_driver.h**
- readRhCRC : **SCD41_AVR128_driver.h**
- readRhLSB : **SCD41_AVR128_driver.h**
- readRhMSB : **SCD41_AVR128_driver.h**
- readTempCRC : **SCD41_AVR128_driver.h**
- readTempLSB : **SCD41_AVR128_driver.h**
- readTempMSB : **SCD41_AVR128_driver.h**
- Rh : **main.c**

- s -

- SCD41_get_data_ready_status() : **SCD41_AVR128_driver.h**
- SCD41_read_measurement() : **SCD41_AVR128_driver.h**
- SCD41_start_periodic_measurement() : **SCD41_AVR128_driver.h**
- SCD41_stop_periodic_measurement() : **SCD41_AVR128_driver.h**
- sensirion_common_generate_crc() : **SCD41_AVR128_driver.h**
- storedCO2 : **SCD41_AVR128_driver.h**
- storedRH : **SCD41_AVR128_driver.h**
- storedTemp : **SCD41_AVR128_driver.h**

- t -

- Temp : **main.c**

- u -

- update_lcd_dog() : **lcd_dog_AVR128_driver.h**
- USART3_BAUD_RATE : **USART3_asynch_transmit.h**
- USART3_init() : **USART3_asynch_transmit.h**
- USART3_sendChar() : **USART3_asynch_transmit.h**
- USART3_sendString() : **USART3_asynch_transmit.h**
- USART_sw_write() : **USART3_asynch_transmit.h**

- w -

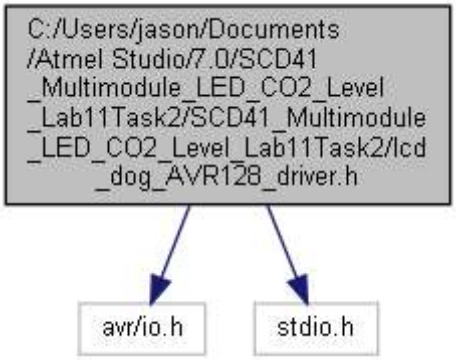
- WRITE_opcode : **MCP23017_CO2_level_LED_simple_display.h**

- delay_30uS() : **lcd_dog_AVR128_driver.h**
- delay_40mS() : **lcd_dog_AVR128_driver.h**
- I2C0_MCP23017_init() : **MCP23017_CO2_level_LED_simple_display.h**
- I2C0_SCD41_init() : **SCD41_AVR128_driver.h**
- init_lcd_dog() : **lcd_dog_AVR128_driver.h**
- init_spi_lcd() : **lcd_dog_AVR128_driver.h**
- lcd_spi_transmit_CMD() : **lcd_dog_AVR128_driver.h**
- lcd_spi_transmit_DATA() : **lcd_dog_AVR128_driver.h**
- main() : **main.c**
- MCP23017_I2C_init() : **MCP23017_CO2_level_LED_simple_display.h**
- MCP23017_I2C_write() : **MCP23017_CO2_level_LED_simple_display.h**
- SCD41_get_data_ready_status() : **SCD41_AVR128_driver.h**
- SCD41_read_measurement() : **SCD41_AVR128_driver.h**
- SCD41_start_periodic_measurement() : **SCD41_AVR128_driver.h**
- SCD41_stop_periodic_measurement() : **SCD41_AVR128_driver.h**
- sensirion_common_generate_crc() : **SCD41_AVR128_driver.h**
- update_lcd_dog() : **lcd_dog_AVR128_driver.h**
- USART3_init() : **USART3_asynch_transmit.h**
- USART3_sendChar() : **USART3_asynch_transmit.h**
- USART3_sendString() : **USART3_asynch_transmit.h**
- USART_sw_write() : **USART3_asynch_transmit.h**

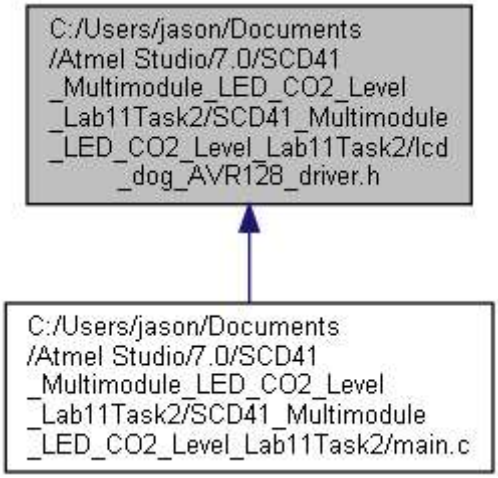
lcd_dog_AVR128_driver.h File Reference

```
#include <avr/io.h>
#include <stdio.h>
```

Include dependency graph for lcd_dog_AVR128_driver.h:



This graph shows which files directly or indirectly include this file:



Go to the source code of this file.

Functions

void	**lcd_spi_transmit_CMD** (unsigned char cmd)
void	**lcd_spi_transmit_DATA** (unsigned char cmd)
void	**init_spi_lcd** (void)
void	**init_lcd_dog** (void)
void	**delay_40mS** (void)
void	**delay_30uS** (void)
void	**update_lcd_dog** (void)

Variables

char	**dsp_buff1** [17]

char **dsp_buff2** [17]

char **dsp_buff3** [17]

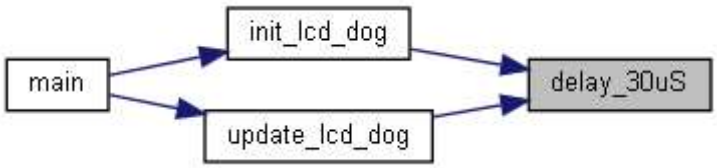
Function Documentation

◆ delay_30uS()

void delay_30uS (void)

Definition at line **158** of file **lcd_dog_AVR128_driver.h**.

Here is the caller graph for this function:



◆ delay_40mS()

void delay_40mS (void)

Definition at line **151** of file **lcd_dog_AVR128_driver.h**.

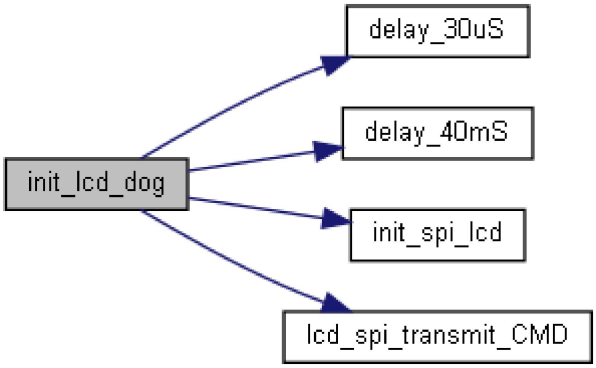
Here is the caller graph for this function:

◆ init_lcd_dog()

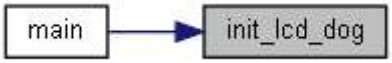
void init_lcd_dog (void)

Definition at line **96** of file **lcd_dog_AVR128_driver.h**.

Here is the call graph for this function:



Here is the caller graph for this function:

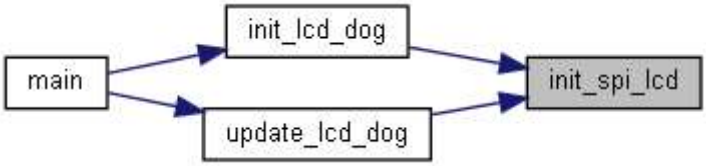


◆ init_spi_lcd()

void init_spi_lcd (void)

Definition at line **78** of file **lcd_dog_AVR128_driver.h**.

Here is the caller graph for this function:

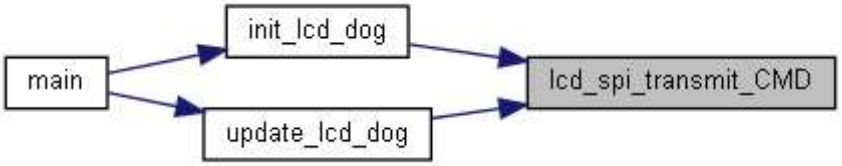


◆ lcd_spi_transmit_CMD()

void lcd_spi_transmit_CMD (unsigned char cmd)

Definition at line **54** of file **lcd_dog_AVR128_driver.h**.

Here is the caller graph for this function:

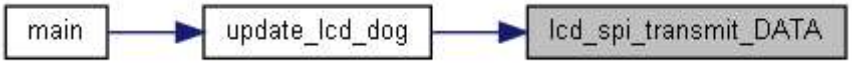


◆ lcd_spi_transmit_DATA()

void lcd_spi_transmit_DATA (unsigned char cmd)

Definition at line **66** of file **lcd_dog_AVR128_driver.h**.

Here is the caller graph for this function:

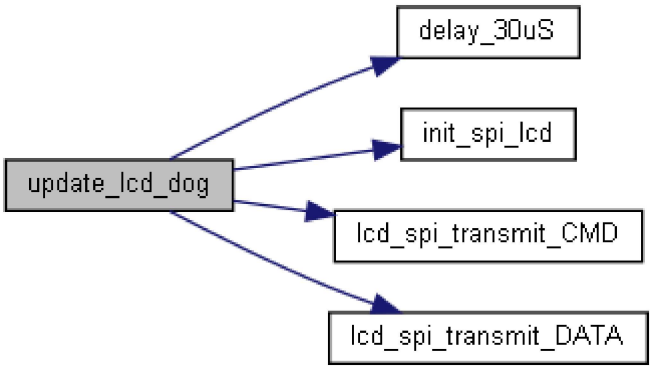


◆ update_lcd_dog()

void update_lcd_dog (void)

Definition at line **168** of file **lcd_dog_AVR128_driver.h**.

Here is the call graph for this function:



Here is the caller graph for this function:

Variable Documentation

◆ dsp_buff1

char dsp_buff1[17]

Definition at line **48** of file **lcd_dog_AVR128_driver.h**.

◆ dsp_buff2

char dsp_buff2[17]

Definition at line **49** of file **lcd_dog_AVR128_driver.h**.

◆ dsp_buff3

char dsp_buff3[17]

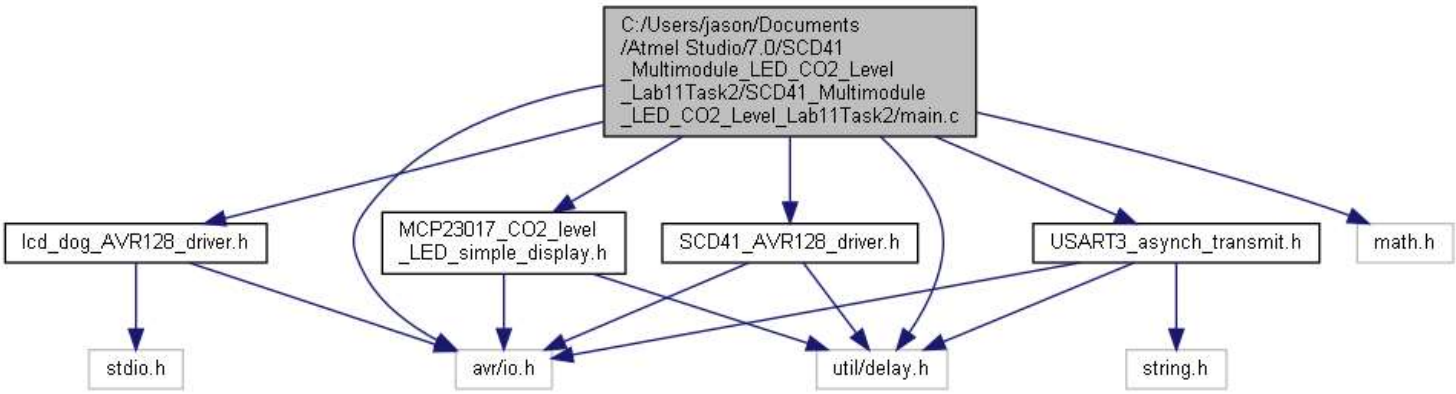
Definition at line **50** of file **lcd_dog_AVR128_driver.h**.

- ADDRESS_GETDATAREADY_LSB : **SCD41_AVR128_driver.h**
- ADDRESS_GETDATAREADY_MSB : **SCD41_AVR128_driver.h**
- ADDRESS_READMEASUR_LSB : **SCD41_AVR128_driver.h**
- ADDRESS_READMEASUR_MSB : **SCD41_AVR128_driver.h**
- ADDRESS_STARTPERIODIC_LSB : **SCD41_AVR128_driver.h**
- ADDRESS_STARTPERIODIC_MSB : **SCD41_AVR128_driver.h**
- ADDRESS_STOPPERIODIC_LSB : **SCD41_AVR128_driver.h**
- ADDRESS_STOPPERIODIC_MSB : **SCD41_AVR128_driver.h**
- CRC8_INIT : **SCD41_AVR128_driver.h**
- CRC8_POLYNOMIAL : **SCD41_AVR128_driver.h**
- F_CPU : **main.c**, **MCP23017_CO2_level_LED_simple_display.h**, **SCD41_AVR128_driver.h**, **USART3_asynch_transmit.h**
- GPIOAaddr_b1 : **MCP23017_CO2_level_LED_simple_display.h**
- GPPUAaddr_b1 : **MCP23017_CO2_level_LED_simple_display.h**
- I2CSLAVE_ADDR_READ : **SCD41_AVR128_driver.h**
- I2CSLAVE_ADDR_WRITE : **SCD41_AVR128_driver.h**
- IOCONaddr_b0 : **MCP23017_CO2_level_LED_simple_display.h**
- IOCONaddr_b1 : **MCP23017_CO2_level_LED_simple_display.h**
- IODIRAaddr_b1 : **MCP23017_CO2_level_LED_simple_display.h**
- IODIRBaddr_b1 : **MCP23017_CO2_level_LED_simple_display.h**
- MAX_INPUT_DISPLAY : **main.c**
- OLATBaddr_b1 : **MCP23017_CO2_level_LED_simple_display.h**
- READ_opcode : **MCP23017_CO2_level_LED_simple_display.h**
- USART3_BAUD_RATE : **USART3_asynch_transmit.h**
- WRITE_opcode : **MCP23017_CO2_level_LED_simple_display.h**

main.c File Reference

```
#include <avr/io.h>
#include <math.h>
#include "lcd_dog_AVR128_driver.h"
#include "SCD41_AVR128_driver.h"
#include "USART3_asynch_transmit.h"
#include "MCP23017_CO2_level_LED_simple_display.h"
#include <util/delay.h>
```

Include dependency graph for main.c:



Go to the source code of this file.

Macros

#define	**F_CPU** 4000000
#define	**MAX_INPUT_DISPLAY** 5

Functions

int	**main** (void)

Variables

uint16_t	**CO2**
uint16_t	**Temp**
uint16_t	**Rh**
uint16_t	**baudRate** = 9600
uint8_t	**dataBits** = USART_CHSIZE_8BIT_gc
unsigned char	**parity** = 0x00

Macro Definition Documentation

◆ F_CPU

#define F_CPU 4000000

Definition at line **10** of file **main.c**.

◆ MAX_INPUT_DISPLAY

```
#define MAX_INPUT_DISPLAY  5
```

Definition at line **17** of file **main.c**.

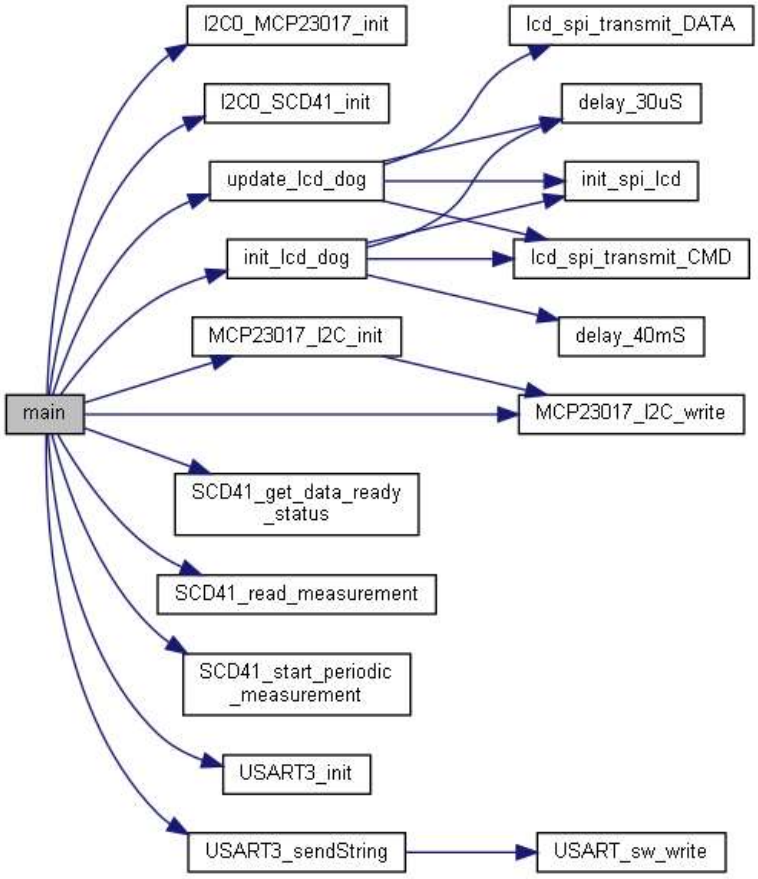
Function Documentation

◆ main()

```
int main ( void  )
```

Definition at line **29** of file **main.c**.

Here is the call graph for this function:



Variable Documentation

◆ baudRate

```
uint16_t baudRate = 9600
```

Definition at line **24** of file **main.c**.

◆ CO2

uint16_t CO2

Definition at line **20** of file **main.c**.

◆ dataBits

uint8_t dataBits = USART_CHSIZE_8BIT_gc

Definition at line **25** of file **main.c**.

◆ parity

unsigned char parity = 0x00

Definition at line **26** of file **main.c**.

◆ Rh

uint16_t Rh

Definition at line **22** of file **main.c**.

◆ Temp

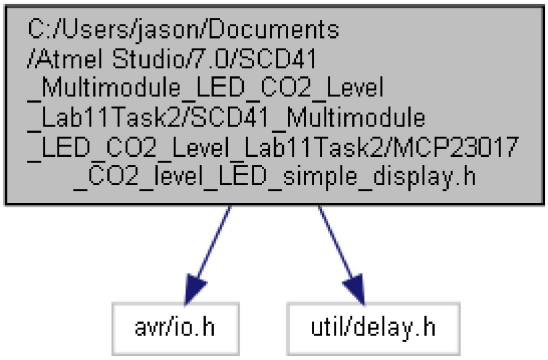
uint16_t Temp

Definition at line **21** of file **main.c**.

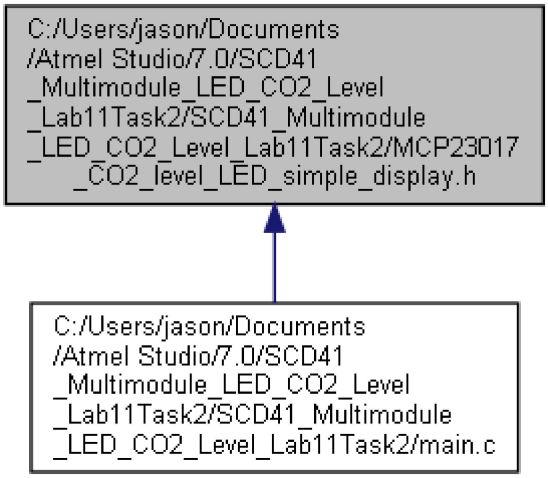
MCP23017_CO2_level_LED_simple_display.h File Reference

```
#include <avr/io.h>
#include <util/delay.h>
```

Include dependency graph for MCP23017_CO2_level_LED_simple_display.h:



This graph shows which files directly or indirectly include this file:



Go to the source code of this file.

Macros

#define **F_CPU**	4000000
#define **IOCONaddr_b0**	0x0A
#define **IOCONaddr_b1**	0x05
#define **IODIRAaddr_b1**	0x00
#define **IODIRBaddr_b1**	0x10
#define **GPPUAaddr_b1**	0x06
#define **GPIOAaddr_b1**	0x09
#define **OLATBaddr_b1**	0x1A
#define **WRITE_opcode**	0x40
#define **READ_opcode**	0x41

Functions

void	**I2C0_MCP23017_init** ()
void	**MCP23017_I2C_init** ()
void	**MCP23017_I2C_write** (uint8_t, uint8_t, uint8_t)

Macro Definition Documentation

◆ F_CPU

```
#define F_CPU   4000000
```

Definition at line **14** of file **MCP23017_CO2_level_LED_simple_display.h**.

◆ GPIOAaddr_b1

```
#define GPIOAaddr_b1   0x09
```

Definition at line **23** of file **MCP23017_CO2_level_LED_simple_display.h**.

◆ GPPUAaddr_b1

```
#define GPPUAaddr_b1   0x06
```

Definition at line **22** of file **MCP23017_CO2_level_LED_simple_display.h**.

◆ IOCONaddr_b0

```
#define IOCONaddr_b0   0x0A
```

Definition at line **18** of file **MCP23017_CO2_level_LED_simple_display.h**.

◆ IOCONaddr_b1

```
#define IOCONaddr_b1  0x05
```

Definition at line **19** of file **MCP23017_CO2_level_LED_simple_display.h**.

◆ IODIRAaddr_b1

```
#define IODIRAaddr_b1  0x00
```

Definition at line **20** of file **MCP23017_CO2_level_LED_simple_display.h**.

◆ IODIRBaddr_b1

```
#define IODIRBaddr_b1  0x10
```

Definition at line **21** of file **MCP23017_CO2_level_LED_simple_display.h**.

◆ OLATBaddr_b1

```
#define OLATBaddr_b1  0x1A
```

Definition at line **24** of file **MCP23017_CO2_level_LED_simple_display.h**.

◆ READ_opcode

```
#define READ_opcode  0x41
```

Definition at line **26** of file **MCP23017_CO2_level_LED_simple_display.h**.

◆ WRITE_opcode

```
#define WRITE_opcode  0x40
```

Definition at line **25** of file **MCP23017_CO2_level_LED_simple_display.h**.

Function Documentation

◆ I2C0_MCP23017_init()

void I2C0_MCP23017_init ( )

Definition at line **38** of file **MCP23017_CO2_level_LED_simple_display.h**.

Here is the caller graph for this function:

◆ MCP23017_I2C_init()

void MCP23017_I2C_init ( )

Definition at line **54** of file **MCP23017_CO2_level_LED_simple_display.h**.

Here is the call graph for this function:

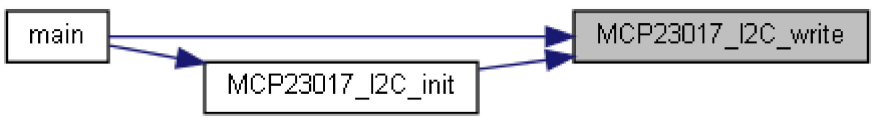
Here is the caller graph for this function:

◆ MCP23017_I2C_write()

void MCP23017_I2C_write (uint8_t opcode,
uint8_t address,
uint8_t data
)

Definition at line **65** of file **MCP23017_CO2_level_LED_simple_display.h**.

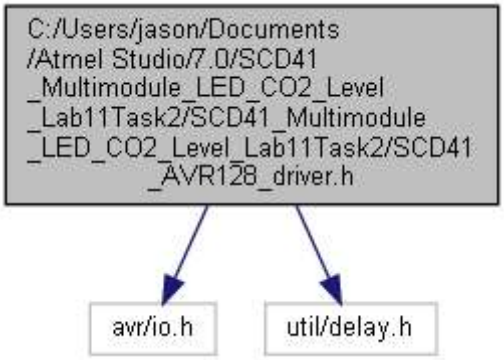
Here is the caller graph for this function:



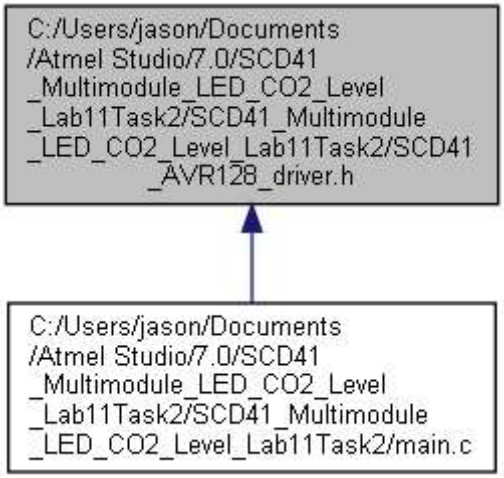
SCD41_AVR128_driver.h File Reference

```
#include <avr/io.h>
#include <util/delay.h>
```

Include dependency graph for SCD41_AVR128_driver.h:



This graph shows which files directly or indirectly include this file:



Go to the source code of this file.

Macros

#define	**F_CPU** 4000000
#define	**CRC8_POLYNOMIAL** 0x31
#define	**CRC8_INIT** 0xFF
#define	**I2CSLAVE_ADDR_WRITE** 0xC4
#define	**I2CSLAVE_ADDR_READ** 0xC5
#define	**ADDRESS_STARTPERIODIC_LSB** 0xB1
#define	**ADDRESS_STARTPERIODIC_MSB** 0x21
#define	**ADDRESS_STOPPERIODIC_LSB** 0x86
#define	**ADDRESS_STOPPERIODIC_MSB** 0x3F
#define	**ADDRESS_READMEASUR_LSB** 0x05
#define	**ADDRESS_READMEASUR_MSB** 0xEC

#define **ADDRESS_GETDATAREADY_LSB** 0xB8

#define **ADDRESS_GETDATAREADY_MSB** 0xE4

Functions

void **I2C0_SCD41_init** ()

void **SCD41_start_periodic_measurement** (uint8_t, uint8_t, uint8_t)

void **SCD41_stop_periodic_measurement** (uint8_t, uint8_t, uint8_t)

void **SCD41_read_measurement** (uint8_t, uint8_t, uint8_t)

uint8_t **SCD41_get_data_ready_status** (uint8_t, uint8_t, uint8_t)

uint8_t **sensirion_common_generate_crc** (const uint8_t *, uint16_t)

Variables

uint8_t **readDataStatusMSB**

uint8_t **readDataStatusLSB**

uint16_t **getDataStatusReadyResponse**

uint8_t **readCO2MSB**

uint8_t **readCO2LSB**

uint8_t **readCO2CRC**

uint16_t **getParseCO2**

uint8_t **readTempMSB**

uint8_t **readTempLSB**

uint8_t **readTempCRC**

uint32_t **getParseTemp**

uint8_t **readRhMSB**

uint8_t **readRhLSB**

uint8_t **readRhCRC**

uint16_t **getParseRh**

uint8_t **readDataStatusCRC**

uint8_t **storedCO2** [2]

uint8_t **storedTemp** [2]

uint8_t **storedRH** [2]

Macro Definition Documentation

◆ ADDRESS_GETDATAREADY_LSB

```
#define ADDRESS_GETDATAREADY_LSB  0xB8
```

Definition at line **44** of file **SCD41_AVR128_driver.h**.

◆ ADDRESS_GETDATAREADY_MSB

```
#define ADDRESS_GETDATAREADY_MSB  0xE4
```

Definition at line **45** of file **SCD41_AVR128_driver.h**.

◆ ADDRESS_READMEASUR_LSB

```
#define ADDRESS_READMEASUR_LSB  0x05
```

Definition at line **40** of file **SCD41_AVR128_driver.h**.

◆ ADDRESS_READMEASUR_MSB

```
#define ADDRESS_READMEASUR_MSB  0xEC
```

Definition at line **41** of file **SCD41_AVR128_driver.h**.

◆ ADDRESS_STARTPERIODIC_LSB

```
#define ADDRESS_STARTPERIODIC_LSB  0xB1
```

Definition at line **32** of file **SCD41_AVR128_driver.h**.

◆ ADDRESS_STARTPERIODIC_MSB

```
#define ADDRESS_STARTPERIODIC_MSB  0x21
```

Definition at line **33** of file **SCD41_AVR128_driver.h**.

◆ ADDRESS_STOPPERIODIC_LSB

```
#define ADDRESS_STOPPERIODIC_LSB   0x86
```

Definition at line **36** of file **SCD41_AVR128_driver.h**.

◆ ADDRESS_STOPPERIODIC_MSB

```
#define ADDRESS_STOPPERIODIC_MSB   0x3F
```

Definition at line **37** of file **SCD41_AVR128_driver.h**.

◆ CRC8_INIT

```
#define CRC8_INIT   0xFF
```

Definition at line **26** of file **SCD41_AVR128_driver.h**.

◆ CRC8_POLYNOMIAL

```
#define CRC8_POLYNOMIAL   0x31
```

Definition at line **25** of file **SCD41_AVR128_driver.h**.

◆ F_CPU

```
#define F_CPU   4000000
```

Definition at line **13** of file **SCD41_AVR128_driver.h**.

◆ I2CSLAVE_ADDR_READ

```
#define I2CSLAVE_ADDR_READ   0xC5
```

Definition at line **29** of file **SCD41_AVR128_driver.h**.

◆ I2CSLAVE_ADDR_WRITE

```
#define I2CSLAVE_ADDR_WRITE   0xC4
```

Definition at line **28** of file **SCD41_AVR128_driver.h**.

Function Documentation

◆ I2C0_SCD41_init()

```
void I2C0_SCD41_init ( )
```

Definition at line **83** of file **SCD41_AVR128_driver.h**.

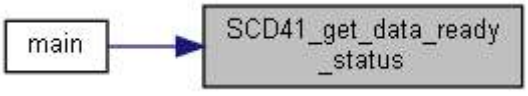
Here is the caller graph for this function:

◆ SCD41_get_data_ready_status()

```
uint8_t SCD41_get_data_ready_status ( uint8_t  SCD41_address,
                                      uint8_t  SCD41_MSB,
                                      uint8_t  SCD41_LSB
                                    )
```

Definition at line **239** of file **SCD41_AVR128_driver.h**.

Here is the caller graph for this function:

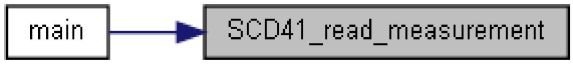


◆ SCD41_read_measurement()

```
void SCD41_read_measurement ( uint8_t  SCD41_address,
                              uint8_t  SCD41_MSB,
                              uint8_t  SCD41_LSB
                            )
```

Definition at line **135** of file **SCD41_AVR128_driver.h**.

Here is the caller graph for this function:



◆ SCD41_start_periodic_measurement()

```
void SCD41_start_periodic_measurement ( uint8_t  SCD41_address,
                                        uint8_t  SCD41_MSB,
                                        uint8_t  SCD41_LSB
                                      )
```

Definition at line **95** of file **SCD41_AVR128_driver.h**.

Here is the caller graph for this function:

◆ SCD41_stop_periodic_measurement()

```
void SCD41_stop_periodic_measurement ( uint8_t  SCD41_address,
                                       uint8_t  SCD41_MSB,
                                       uint8_t  SCD41_LSB
                                     )
```

Definition at line **116** of file **SCD41_AVR128_driver.h**.

◆ sensirion_common_generate_crc()

```
uint8_t sensirion_common_generate_crc ( const uint8_t *  data,
                                        uint16_t         count
                                      )
```

Definition at line **293** of file **SCD41_AVR128_driver.h**.

Variable Documentation

◆ getDataStatusReadyResponse

uint16_t getDataStatusReadyResponse

Definition at line **51** of file **SCD41_AVR128_driver.h**.

◆ getParseCO2

uint16_t getParseCO2

Definition at line **57** of file **SCD41_AVR128_driver.h**.

◆ getParseRh

uint16_t getParseRh

Definition at line **69** of file **SCD41_AVR128_driver.h**.

◆ getParseTemp

uint32_t getParseTemp

Definition at line **63** of file **SCD41_AVR128_driver.h**.

◆ readCO2CRC

uint8_t readCO2CRC

Definition at line **56** of file **SCD41_AVR128_driver.h**.

◆ readCO2LSB

uint8_t readCO2LSB

Definition at line **55** of file **SCD41_AVR128_driver.h**.

◆ readCO2MSB

uint8_t readCO2MSB

Definition at line **54** of file **SCD41_AVR128_driver.h**.

◆ readDataStatusCRC

uint8_t readDataStatusCRC

Definition at line **72** of file **SCD41_AVR128_driver.h**.

◆ readDataStatusLSB

uint8_t readDataStatusLSB

Definition at line **50** of file **SCD41_AVR128_driver.h**.

◆ readDataStatusMSB

uint8_t readDataStatusMSB

Definition at line **49** of file **SCD41_AVR128_driver.h**.

◆ readRhCRC

uint8_t readRhCRC

Definition at line **68** of file **SCD41_AVR128_driver.h**.

◆ readRhLSB

uint8_t readRhLSB

Definition at line **67** of file **SCD41_AVR128_driver.h**.

◆ readRhMSB

uint8_t readRhMSB

Definition at line **66** of file **SCD41_AVR128_driver.h**.

◆ readTempCRC

uint8_t readTempCRC

Definition at line **62** of file **SCD41_AVR128_driver.h**.

◆ readTempLSB

uint8_t readTempLSB

Definition at line **61** of file **SCD41_AVR128_driver.h**.

◆ readTempMSB

uint8_t readTempMSB

Definition at line **60** of file **SCD41_AVR128_driver.h**.

◆ storedCO2

uint8_t storedCO2[2]

Definition at line **75** of file **SCD41_AVR128_driver.h**.

◆ storedRH

uint8_t storedRH[2]

Definition at line **77** of file **SCD41_AVR128_driver.h**.

◆ storedTemp

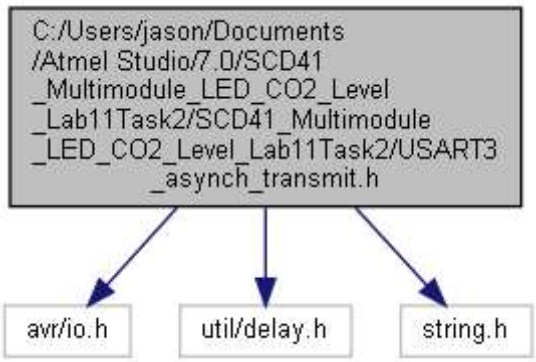
uint8_t storedTemp[2]

Definition at line **76** of file **SCD41_AVR128_driver.h**.

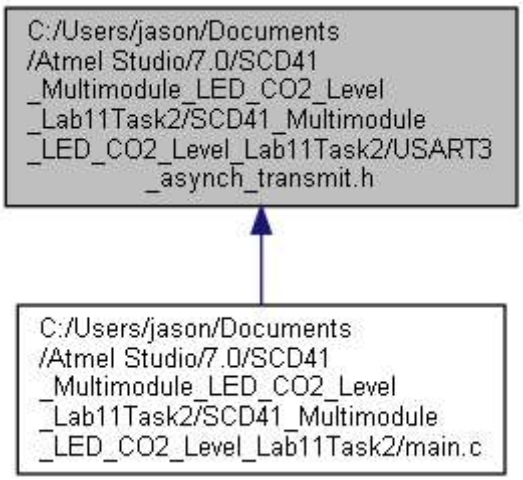
USART3_asynch_transmit.h File Reference

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

Include dependency graph for USART3_asynch_transmit.h:



This graph shows which files directly or indirectly include this file:



Go to the source code of this file.

Macros

#define	**F_CPU** 4000000
#define	**USART3_BAUD_RATE**(BAUD_RATE) ((float)(**F_CPU** * 64 / (16 *(float)BAUD_RATE)))

Functions

void	**USART3_sendChar** (char c)
void	**USART3_init** (uint16_t, uint8_t, unsigned char)
void	**USART_sw_write** (char)
void	**USART3_sendString** (char *input)

Macro Definition Documentation

◆ F_CPU

```
#define F_CPU   4000000
```

Definition at line **12** of file **USART3_asynch_transmit.h**.

◆ USART3_BAUD_RATE

```
#define USART3_BAUD_RATE (   BAUD_RATE )   ((float)(F_CPU * 64 / (16 *(float)BAUD_RATE)))
```

Definition at line **13** of file **USART3_asynch_transmit.h**.

Function Documentation

◆ USART3_init()

```
void USART3_init ( uint16_t          baud,
                   uint8_t           data_bits,
                   unsigned char     parity
                 )
```

Definition at line **27** of file **USART3_asynch_transmit.h**.

Here is the caller graph for this function:

◆ USART3_sendChar()

```
void USART3_sendChar ( char  c )
```

◆ USART3_sendString()

void USART3_sendString (char * input)

Definition at line **40** of file **USART3_asynch_transmit.h**.

Here is the call graph for this function:

Here is the caller graph for this function:

◆ USART_sw_write()

void USART_sw_write (char c)

Definition at line **50** of file **USART3_asynch_transmit.h**.

Here is the caller graph for this function:

- baudRate : **main.c**
- CO2 : **main.c**
- dataBits : **main.c**
- dsp_buff1 : **lcd_dog_AVR128_driver.h**
- dsp_buff2 : **lcd_dog_AVR128_driver.h**
- dsp_buff3 : **lcd_dog_AVR128_driver.h**
- getDataStatusReadyResponse : **SCD41_AVR128_driver.h**
- getParseCO2 : **SCD41_AVR128_driver.h**
- getParseRh : **SCD41_AVR128_driver.h**
- getParseTemp : **SCD41_AVR128_driver.h**
- parity : **main.c**
- readCO2CRC : **SCD41_AVR128_driver.h**
- readCO2LSB : **SCD41_AVR128_driver.h**
- readCO2MSB : **SCD41_AVR128_driver.h**
- readDataStatusCRC : **SCD41_AVR128_driver.h**
- readDataStatusLSB : **SCD41_AVR128_driver.h**
- readDataStatusMSB : **SCD41_AVR128_driver.h**
- readRhCRC : **SCD41_AVR128_driver.h**
- readRhLSB : **SCD41_AVR128_driver.h**
- readRhMSB : **SCD41_AVR128_driver.h**
- readTempCRC : **SCD41_AVR128_driver.h**
- readTempLSB : **SCD41_AVR128_driver.h**
- readTempMSB : **SCD41_AVR128_driver.h**
- Rh : **main.c**
- storedCO2 : **SCD41_AVR128_driver.h**
- storedRH : **SCD41_AVR128_driver.h**
- storedTemp : **SCD41_AVR128_driver.h**
- Temp : **main.c**