ESE 381

Lab 11: Air Monitoring System II - Sensirion Driver Operation of SCD41 CO2, Humidity, and Temperature Sensor

5/1/2025

Muhammad Sharjeel and Naafiul Hossain

115185427

115107623

Section: L02 Bench 7

```c
/*
 * sensirion_i2c.c
 *
 * Created: 4/27/2025 2:16:57 AM
 *  Author: Naafiul Hossain
 */
#include "sensirion_i2c_hal.h"
#include "sensirion_common.h"
#include "sensirion_config.h"
#include <avr/io.h>
#include <util/delay.h>
#include <stdint.h>

/*
 * INSTRUCTIONS
 * ============
 *
 * Implement all functions where they are marked as IMPLEMENT.
 * Follow the function specification in the comments.
 */

/**
 * Select the current i2c bus by index.
 * All following i2c operations will be directed at that bus.
 *
 * THE IMPLEMENTATION IS OPTIONAL ON SINGLE-BUS SETUPS (all sensors on the same
 * bus)
 *
 * @param bus_idx   Bus index to select
 * @returns         0 on success, an error code otherwise
 */
int16_t sensirion_i2c_hal_select_bus(uint8_t bus_idx) {
    // prof short said we can ignore
    /* TODO:IMPLEMENT or leave empty if all sensors are located on one single
     * bus
     */
    return NOT_IMPLEMENTED_ERROR;
}

/**
 * Initialize all hard- and software components that are needed for the I2C
 * communication.
 */
void sensirion_i2c_hal_init(void) {
    /* TODO:IMPLEMENT */
    // same as lab 10
     PORTMUX.TWIROUTEA = PORTMUX_TWI0_ALT1_gc; // Use PA2 (SDA) and PA3 (SCL)
     TWI0.MBAUD = 0x01; // 400kHz if F_CPU = 4MHz
```

```
49      TWI0.MCTRLA = TWI_ENABLE_bm;
50      TWI0.MSTATUS = TWI_BUSSTATE_IDLE_gc;
51      PORTA_PIN2CTRL |=PORT_PULLUPEN_bm;
52      PORTA_PIN3CTRL |=PORT_PULLUPEN_bm;
53  }
54
55  /**
56   * Release all resources initialized by sensirion_i2c_hal_init().
57   */
58  void sensirion_i2c_hal_free(void) {
59      /* TODO:IMPLEMENT or leave empty if no resources need to be freed */
60      // short did say we dont have to this function
61      TWI0_MCTRLA &= ~TWI_ENABLE_bm; //DISABLE TWI...?
62  }
63
64  /**
65  PER PROF SHORT-MAKE CHANGES IN THE READ FUNCTION
66
67   * Execute one read transaction on the I2C bus, reading a given number of
       bytes.
68   * If the device does not acknowledge the read command, an error shall be
69   * returned.
70   *
71   * @param address 7-bit I2C address to read from
72   * @param data    pointer to the buffer where the data is to be stored
73   * @param count   number of bytes to read from I2C and store in the buffer
74   * @returns 0 on success, error code otherwise
75   */
76  int8_t sensirion_i2c_hal_read(uint8_t address, uint8_t* data, uint8_t count) {
77      /* TODO:IMPLEMENT */
78      TWI0.MADDR = (address << 1) | 0x01;
79      // read 9 bytes
80      for(uint8_t i = 0; i < count; i++) {
81          while(!(TWI0.MSTATUS & TWI_RIF_bm));
82          data[i] = TWI0.MDATA;
83          if(i < count-1) {
84              // TWI0.MCTRLB = TWI_MCMD_RECVTRANS_gc;                 // ack and
                   continue read but same as 0x02
85              TWI0.MCTRLB = 0x02; // ACK
86          }
87          else {
88              TWI0.MCTRLB = TWI_ACKACT_NACK_gc | TWI_MCMD_STOP_gc;    // Read
                  last byte and send nack+stop condition
89          }
90      }
91
92      return NOT_IMPLEMENTED_ERROR;
93  }
94
```

```
 95
 96  /**
 97  PER PROF SHORT-MAKE CHANGES IN THE WRITE
 98   * Execute one write transaction on the I2C bus, sending a given number of
 99   * bytes. The bytes in the supplied buffer must be sent to the given address. If
100   * the slave device does not acknowledge any of the bytes, an error shall be
101   * returned.
102   *
103   * @param address 7-bit I2C address to write to
104   * @param data    pointer to the buffer containing the data to write
105   * @param count   number of bytes to read from the buffer and send over I2C
106   * @returns 0 on success, error code otherwise
107   */
108  int8_t sensirion_i2c_hal_write(uint8_t address, const uint8_t* data,
109                                 uint8_t count) {
110      /* TODO:IMPLEMENT */
111       TWI0.MADDR = (address << 1); // 7-bit address, write
112
113       while (!(TWI0.MSTATUS & TWI_WIF_bm)); // Wait for write interrupt flag
114
115       for (uint8_t i = 0; i < count; i++) {
116           TWI0.MDATA = data[i];
117           while (!(TWI0.MSTATUS & TWI_WIF_bm)); // Wait for data write or until slave ACKs
118           if (TWI0.MSTATUS & (TWI_ARBLOST_bm | TWI_BUSERR_bm)) { //it didnt ack
119               return -1; // Error
120           }
121       }
122
123       TWI0.MCTRLB = TWI_MCMD_STOP_gc; // Send STOP condition
124       return 0; // Success
125      return NOT_IMPLEMENTED_ERROR;
126  }
127
128  /**
129   * Sleep for a given number of microseconds. The function should delay the
130   * execution for at least the given time, but may also sleep longer.
131   *
132   * Despite the unit, a <10 millisecond precision is sufficient.
133   *
134   * @param useconds the sleep time in microseconds
135   */
136  void sensirion_i2c_hal_sleep_usec(uint32_t useconds) {
137      // don't need according to short but eh---this might still work
138      /* TODO:IMPLEMENT */
139      while (useconds--) {
140          _delay_us(1);
141      }
```

```
142 }
```

```c
/*
 * Labb11_Task2_sensirion_i2c_hal.c.c
 *
 * Created: 4/30/2025 1:17:08 AM
 * Author : Naafiul Hossain
 */

/*
 * sensirion_i2c_hal.c - Task 2: Interrupt-based HAL using array-like buffers
 * Author: Naafiul Hossain
 */

#include "sensirion_i2c_hal.h"
#include "sensirion_common.h"
#include "sensirion_config.h"
#include <avr/io.h>
#include <util/delay.h>
#include <stdint.h>
#include <avr/interrupt.h>

#define I2C_BUFFER_SIZE 32


// Global array-based buffers

volatile uint8_t i2c_tx_buffer[I2C_BUFFER_SIZE];
volatile uint8_t i2c_rx_buffer[I2C_BUFFER_SIZE];
volatile uint8_t i2c_tx_len = 0;
volatile uint8_t i2c_rx_len = 0;
volatile uint8_t i2c_tx_index = 0;
volatile uint8_t i2c_rx_index = 0;
volatile uint8_t i2c_done_flag = 0;
volatile uint8_t i2c_error = 0;


int16_t sensirion_i2c_hal_select_bus(uint8_t bus_idx) {
    return 0;
}


// TWI0 Initialization

void sensirion_i2c_hal_init(void) {
    PORTMUX.TWIROUTEA = PORTMUX_TWI0_ALT1_gc;
    TWI0.MBAUD = 0x01; // 400kHz for F_CPU = 4MHz
    TWI0.MCTRLA = TWI_ENABLE_bm | TWI_WIEN_bm | TWI_RIEN_bm;
    TWI0.MSTATUS = TWI_BUSSTATE_IDLE_gc;

    PORTA.PIN2CTRL |= PORT_PULLUPEN_bm; // Pull-ups on SDA
```

```
50      PORTA.PIN3CTRL |= PORT_PULLUPEN_bm; // Pull-ups on SCL
51
52      sei(); // Global interrupt enable
53  }
54
55
56  // TWI0 Free
57
58  void sensirion_i2c_hal_free(void) {
59      TWI0.MCTRLA &= ~TWI_ENABLE_bm;
60  }
61
62
63  // I2C Write (Interrupt-based)
64
65  int8_t sensirion_i2c_hal_write(uint8_t address, const uint8_t* data, uint8_t
      count) {
66      // Reject if data size exceeds buffer capacity
67      if (count > I2C_BUFFER_SIZE) return -1;
68
69      // Copy user data into internal transmit buffer
70      for (uint8_t i = 0; i < count; i++) {
71          i2c_tx_buffer[i] = data[i];
72      }
73
74      // Initialize state variables for the ISR
75      i2c_tx_len = count;        // Total bytes to send
76      i2c_tx_index = 0;          // Reset byte index
77      i2c_done_flag = 0;         // Clear "done" signal
78      i2c_error = 0;             // Clear error status
79
80      // Begin transmission by writing slave address + write bit (0) to MADDR
81      TWI0.MADDR = (address << 1); // Automatically triggers ISR via WIF
82
83      // Wait (block) until the ISR completes the entire transmission
84      while (!i2c_done_flag);
85
86      // Return error status (0 = success, -1 = error)
87      return i2c_error ? -1 : 0;
88  }
89
90
91
92  // I2C Read (Interrupt-based)
93
94  int8_t sensirion_i2c_hal_read(uint8_t address, uint8_t* data, uint8_t count) {
95      // Reject if requested read size exceeds buffer size
96      if (count > I2C_BUFFER_SIZE) return -1;
97
```

```
 98      // Initialize read-related state variables
 99      i2c_rx_len = count;        // Total number of bytes to receive
100      i2c_rx_index = 0;          // Start at beginning of RX buffer
101      i2c_done_flag = 0;         // Clear completion flag
102      i2c_error = 0;             // Clear error flag
103
104      // Send I2C START + address + read bit (1)
105      TWI0.MADDR = (address << 1) | 0x01;
106
107      // Wait (block) until ISR finishes reading all bytes
108      while (!i2c_done_flag);
109
110      // Copy data from internal RX buffer to user-provided buffer
111      for (uint8_t i = 0; i < count; i++) {
112          data[i] = i2c_rx_buffer[i];
113      }
114
115      // Return success or error status
116      return i2c_error ? -1 : 0;
117  }
118
119
120
121  void sensirion_i2c_hal_sleep_usec(uint32_t useconds) {
122      while (useconds--) {
123          _delay_us(1);
124      }
125  }
126
127
128  // TWI0 ISR
129
130  ISR(TWI0_TWIM_vect) {
131      // Error case
132      if (TWI0.MSTATUS & (TWI_BUSERR_bm | TWI_ARBLOST_bm)) { //if it doesnt find
         an ack
133          i2c_error = 1;
134          i2c_done_flag = 1;
135          TWI0.MCTRLB = TWI_MCMD_STOP_gc;
136          TWI0.MSTATUS |= TWI_RIF_bm | TWI_WIF_bm;
137          return;
138      }
139
140      // Write mode
141      if (TWI0.MSTATUS & TWI_WIF_bm) {
142          if (i2c_tx_index < i2c_tx_len) {
143              TWI0.MDATA = i2c_tx_buffer[i2c_tx_index++]; //loads next byte. do
                 we need to ack? i assumed hardware handles in
144          } else {
```

```
145             TWI0.MCTRLB = TWI_MCMD_STOP_gc; //send stop
146             i2c_done_flag = 1;
147         }
148     }
149
150     // Read mode
151     if (TWI0.MSTATUS & TWI_RIF_bm) {
152         if (i2c_rx_index < i2c_rx_len) {
153             i2c_rx_buffer[i2c_rx_index++] = TWI0.MDATA;
154
155             if (i2c_rx_index == i2c_rx_len) {
156                 TWI0.MCTRLB = TWI_ACKACT_bm | TWI_MCMD_STOP_gc; // This is the
                       LAST byte ? Send NACK + STOP
157                 i2c_done_flag = 1;
158             } else {
159                 TWI0.MCTRLB = TWI_MCMD_RECVTRANS_gc; //Not the last byte ?
                      Send ACK and continue
160             }
161         }
162     }
163 }
164
```

```c
#include "sensirion_i2c_hal.h"
#include "scd4x_i2c.h"
#include "lcd.h"
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>

int main(void) {
    // === Init I²C for sensor ===
    sensirion_i2c_hal_init();
    scd4x_init(0);                      // Initialize sensor with default address..task 2 or sei
    // task 1 scd4x_init(); // No address parameter in polling version

    scd4x_start_periodic_measurement(); // Begin measurement loop

    // === Init SPI for LCD ===
    init_spi0_SerLCD();
    clear_display_buffs();

    // === Sensor variables ===
    bool data_ready = false;
    uint16_t co2 = 0;
    int32_t temperature = 0;
    int32_t humidity = 0;

    while (1) {
        // Check if new sensor data is available
        scd4x_get_data_ready_status(&data_ready);

        if (data_ready) {
            // Read latest CO2, temperature, humidity values
            scd4x_read_measurement(&co2, &temperature, &humidity);

            // === Format sensor data into LCD buffers ===
            sprintf(dsp_buff1, "CO2: %u ppm", co2);
            sprintf(dsp_buff2, "Temp: %ld.%02ld C", temperature / 1000, (temperature % 1000) / 10);
            sprintf(dsp_buff3, "RH: %ld.%02ld %%", humidity / 1000, (humidity % 1000) / 10);
            sprintf(dsp_buff4, "Air Monitor Ready");

            update_SerLCD();  // Push display buffers to the LCD
        }

        // Small delay before checking again
        sensirion_i2c_hal_sleep_usec(5000);
    }
}
```