```c
/*
 * 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_ */
```