```c
/*
 * Copyright (c) 2018, Sensirion AG
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * * Redistributions of source code must retain the above copyright notice, this
 *   list of conditions and the following disclaimer.
 *
 * * Redistributions in binary form must reproduce the above copyright notice,
 *   this list of conditions and the following disclaimer in the documentation
 *   and/or other materials provided with the distribution.
 *
 * * Neither the name of Sensirion AG nor the names of its
 *   contributors may be used to endorse or promote products derived from
 *   this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef SENSIRION_I2C_HAL_H
#define SENSIRION_I2C_HAL_H

#include "sensirion_config.h"

// Hardware abstraction layer for AVR128B48
// requires includes for said MCU

#include <avr/io.h>

#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

/**
 * Select the current i2c bus by index.
 * All following i2c operations will be directed at that bus.
```

```
49   *
50   * THE IMPLEMENTATION IS OPTIONAL ON SINGLE-BUS SETUPS (all sensors on the
       same
51   * bus)
52   *
53   * @param bus_idx   Bus index to select
54   * @returns         0 on success, an error code otherwise
55   */
56  int16_t sensirion_i2c_hal_select_bus(uint8_t bus_idx);
57
58  /**
59   * Initialize all hard- and software components that are needed for the I2C
60   * communication.
61   */
62  void sensirion_i2c_hal_init(void);
63
64  /**
65   * Release all resources initialized by sensirion_i2c_hal_init().
66   */
67  void sensirion_i2c_hal_free(void);
68
69  /**
70   * Execute one read transaction on the I2C bus, reading a given number of
       bytes.
71   * If the device does not acknowledge the read command, an error shall be
72   * returned.
73   *
74   * @param address 7-bit I2C address to read from
75   * @param data    pointer to the buffer where the data is to be stored
76   * @param count   number of bytes to read from I2C and store in the buffer
77   * @returns 0 on success, error code otherwise
78   */
79  int8_t sensirion_i2c_hal_read(uint8_t address, uint8_t* data, uint8_t count);
80
81  /**
82   * Execute one write transaction on the I2C bus, sending a given number of
83   * bytes. The bytes in the supplied buffer must be sent to the given address.
       If
84   * the slave device does not acknowledge any of the bytes, an error shall be
85   * returned.
86   *
87   * @param address 7-bit I2C address to write to
88   * @param data    pointer to the buffer containing the data to write
89   * @param count   number of bytes to read from the buffer and send over I2C
90   * @returns 0 on success, error code otherwise
91   */
92  int8_t sensirion_i2c_hal_write(uint8_t address, const uint8_t* data,
93                                 uint8_t count);
94
```

```
 95 /**
 96  * Sleep for a given number of microseconds. The function should delay the
 97  * execution approximately, but no less than, the given time.
 98  *
 99  * When using hardware i2c:
100  * Despite the unit, a <10 millisecond precision is sufficient.
101  *
102  * When using software i2c:
103  * The precision needed depends on the desired i2c frequency, i.e. should be
104  * exact to about half a clock cycle (defined in
105  * `SENSIRION_I2C_CLOCK_PERIOD_USEC` in `sensirion_sw_i2c_gpio.h`).
106  *
107  * Example with 400kHz requires a precision of 1 / (2 * 400kHz) == 1.25usec.
108  *
109  * @param useconds the sleep time in microseconds
110  */
111 void sensirion_i2c_hal_sleep_usec(uint32_t useconds);
112
113 #ifdef __cplusplus
114 }
115 #endif /* __cplusplus */
116
117 #endif /* SENSIRION_I2C_HAL_H */
118
```

```c
/*
 * Copyright (c) 2018, Sensirion AG
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * * Redistributions of source code must retain the above copyright notice, this
 *   list of conditions and the following disclaimer.
 *
 * * Redistributions in binary form must reproduce the above copyright notice,
 *   this list of conditions and the following disclaimer in the documentation
 *   and/or other materials provided with the distribution.
 *
 * * Neither the name of Sensirion AG nor the names of its
 *   contributors may be used to endorse or promote products derived from
 *   this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

#include "sensirion_i2c_hal.h"
#include "sensirion_common.h"
#include "sensirion_config.h"

#define F_CPU 4000000UL
#include <util/delay.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) {
    /* 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) {
    TWI0.CTRLA = TWI_INPUTLVL_I2C_gc | TWI_SDASETUP_8CYC_gc | TWI_SDAHOLD_50NS_gc;
    TWI0.MBAUD = 0;
    TWI0.MCTRLA = TWI_ENABLE_bm;
    TWI0.MSTATUS = TWI_BUSSTATE_IDLE_gc;
}

/**
 * Release all resources initialized by sensirion_i2c_hal_init().
 */
void sensirion_i2c_hal_free(void) {
    /* TODO:IMPLEMENT or leave empty if no resources need to be freed */
}

/**
 * Execute one read transaction on the I2C bus, reading a given number of bytes.
 * If the device does not acknowledge the read command, an error shall be
 * returned.
 *
 * @param address 7-bit I2C address to read from
 * @param data    pointer to the buffer where the data is to be stored
 * @param count   number of bytes to read from I2C and store in the buffer
 * @returns 0 on success, error code otherwise
 */
int8_t sensirion_i2c_hal_read(uint8_t address, uint8_t* data, uint8_t count) {
    // wait until bus is idle or we own the bus
    while ((TWI0.MSTATUS & 0x03) != TWI_BUSSTATE_IDLE_gc && (TWI0.MSTATUS &
```

```
          0x03) != TWI_BUSSTATE_OWNER_gc) {}
 95
 96      // bitshift to the right and then bit mask for the write
 97      TWI0.MADDR = address << 1 | 0x01;
 98
 99      // wait until the address is done shifted out
100      while (!(TWI0.MSTATUS & TWI_RIF_bm)){}
101
102      // check if nack or ack
103      // if 1 == NACK
104      // if 0 == ACK
105      if (TWI0.MSTATUS & TWI_RXACK_bm)
106      {
107          TWI0.MCTRLB = TWI_MCMD_STOP_gc;
108          return 1;
109      }
110
111      for (uint8_t i = 0; i < count - 1; i++)
112      {
113          // wait until I can read
114          while (!(TWI0.MSTATUS & TWI_RIF_bm)){}
115
116          // data available, put in pointer at idx i
117          data[i] = TWI0.MDATA;
118
119          // Receiver always sends acknowledge
120          TWI0.MCTRLB = TWI_MCMD_RECVTRANS_gc;
121
122      }
123
124      // wait until I can read the final bit
125      while (!(TWI0.MSTATUS & TWI_RIF_bm)){}
126
127      // place it in the final index
128      data[count - 1] = TWI0.MDATA;
129
130      // terminate by sending NACK and stop condition
131      TWI0.MCTRLB = TWI_ACKACT_NACK_gc | TWI_MCMD_STOP_gc;
132      return NO_ERROR;
133  }
134
135  /**
136   * Execute one write transaction on the I2C bus, sending a given number of
137   * bytes. The bytes in the supplied buffer must be sent to the given address.
        If
138   * the slave device does not acknowledge any of the bytes, an error shall be
139   * returned.
140   *
141   * @param address 7-bit I2C address to write to
```

```
142   * @param data    pointer to the buffer containing the data to write
143   * @param count   number of bytes to read from the buffer and send over I2C
144   * @returns 0 on success, error code otherwise
145   */
146  int8_t sensirion_i2c_hal_write(uint8_t address, const uint8_t* data,
147                                 uint8_t count) {
148
149      // Wait until the bus state is idle before writing
150      while ((TWI0.MSTATUS & 0x03) != TWI_BUSSTATE_IDLE_gc && (TWI0.MSTATUS &
           0x03) != TWI_BUSSTATE_OWNER_gc) {}
151
152      // the default address is 0x62
153      // bitshift to the right and then bit mask for the write
154      TWI0.MADDR =  address << 1;
155
156      // wait until the address is done shifted out
157      while (!(TWI0.MSTATUS & TWI_WIF_bm)){}
158
159      // check if nack or ack
160      // if 1 == NACK
161      // if 0 == ACK
162      if (TWI0.MSTATUS & TWI_RXACK_bm)
163      {
164          TWI0.MCTRLB = TWI_MCMD_STOP_gc;
165          return 1;
166      }
167
168      // otherwise, from here on forth, writing is possible.
169      for (int i = 0; i < count-1; i++)
170      {
171          TWI0.MDATA = data[i];
172
173          // Wait until you can write more data
174          while (!(TWI0.MSTATUS & TWI_WIF_bm)){}
175
176          // verify constant ACKS
177          if (TWI0.MSTATUS & TWI_RXACK_bm)
178          {
179              TWI0.MCTRLB = TWI_MCMD_STOP_gc;
180              return 1;
181          }
182      }
183
184      // last chunk of data to be sent
185      TWI0.MDATA = data[count-1];
186
187      // Verified that there is no more data to be shifted out
188      while (!(TWI0.MSTATUS & TWI_WIF_bm)){}
189
```

```
190       // finally, send the stop condition
191       TWI0.MCTRLB = TWI_MCMD_STOP_gc;
192       return NO_ERROR;
193   }
194
195   /**
196    * Sleep for a given number of microseconds. The function should delay the
197    * execution for at least the given time, but may also sleep longer.
198    *
199    * Despite the unit, a < 10 millisecond precision is sufficient.
200    *
201    * @param useconds the sleep time in microseconds
202    */
203   void sensirion_i2c_hal_sleep_usec(uint32_t useconds) {
204       // Unique delays from the file.
205       // I typecasted the stuff'
206       /*
207       sensirion_i2c_hal_sleep_usec(1 * 1000);
208       sensirion_i2c_hal_sleep_usec(30 * 1000);
209       sensirion_i2c_hal_sleep_usec((uint32_t) 50 * 1000);
210       sensirion_i2c_hal_sleep_usec((uint32_t) 400 * 1000);
211       sensirion_i2c_hal_sleep_usec((uint32_t) 500 * 1000);
212       sensirion_i2c_hal_sleep_usec((uint32_t) 800 * 1000);
213       sensirion_i2c_hal_sleep_usec((uint32_t) 1200 * 1000);
214       sensirion_i2c_hal_sleep_usec((uint32_t) 5000 * 1000);
215       sensirion_i2c_hal_sleep_usec((uint32_t) 10000 * 1000);
216       */
217
218       // handles all the cases from the file.
219       switch(useconds){
220           case 1000:
221               _delay_ms(1);
222               break;
223           case 30000:
224               _delay_ms(30);
225               break;
226           case 50000:
227               _delay_ms(50);
228               break;
229           case 400000:
230               _delay_ms(400);
231               break;
232           case 500000:
233               _delay_ms(500);
234               break;
235           case 800000:
236               _delay_ms(800);
237               break;
238           case 1200000:
```

```
239             _delay_ms(1200);
240             break;
241         case 5000000:
242             _delay_ms(5000);
243             break;
244         case 10000000:
245             _delay_ms(1000);
246             _delay_ms(1000);
247             _delay_ms(1000);
248             _delay_ms(1000);
249             _delay_ms(1000);
250             _delay_ms(1000);
251             _delay_ms(1000);
252             break;
253         case 0:
254             break;
255     }
256 }
257
```