

I2C HDC1080 SENSOR DRIVERS

Dan Blanchette

February 20, 2023

1 Source Code

1.1 hdc1080drivers.h

```
1  /**
2   * @file hdc1080Drivers.h
3   * @author Dan Blanchette
4   * @brief HDC1080 Header File
5   * @version 0.1
6   * @date 2023-02-20
7   *
8   * @copyright Copyright (c) 2023
9   *
10  */
11
12  #ifndef HDC1080_DRIVERS
13  #define HDC1080_DRIVERS
14
15  #include <stdio.h>
16  #include "pico/stdlib.h"
17  #include "pico/binary_info.h"
18  #include "hardware/i2c.h"
19
20  // I2C reserves some addresses for special purposes. We exclude
21  // these from the scan.
22  // These are any addresses of the form 000 0xxx or 111 1xxx
23  #define HDC1080ADDRESS 0x40 // In the Data Sheet this address
24  // identifies the device
25  #define HDC1080_MANF_DEVICE_ID_REG 0xFE
26  #define TEMPERATURE 0x00
27  #define HUMIDITY 0x00
28  #define CONFIG 0x02
29  #define SERIAL_FIRST 0xFB
30  #define SERIAL_MID 0xFC
31  #define SERIAL_LAST 0xFD
32  #define DEV_ID 0xFF
33
34  #define I2C_PORT i2c1
35
36
```

```

37  /* ENUMERATED DATA TYPES*/
38  enum Temp_Type
39  {
40      CEL = 0,
41      FAHR = 1
42  };
43  enum HDC_Read_Measurements
44  {
45      TEMP_C,
46      TEMP_F,
47      PERCENT_HUM
48  };
49  enum Resolution_Type
50  {
51      HIGH = 14,
52      MED = 11,
53      LOW = 8
54  };
55  enum HDC_Config_Reg
56  {
57      T_OR_H_14R = 0x00, // read the temperature and humidity at 14
                          // bit resolution
58      TEMP_11R = 0x004, // read the temperature at 11 bit resolution
59      HUMID_11R = 0x01, // read the humidity at 11 bit resolution
60      HUMID_8R = 0x02, // read the humidity at 8 bit resolution
61      BOTH_14 = 0x10, // read both the temperature and humidity at
                      // 14 bit resolution
62      BOTH_11 = 0x15, // read both the temperature and humidity at
                      // 11 bit resolution
63      RESET_VAL = 0x10, // reset the configuration register - this
                      // cannot be read
64      HEATER_1 = 0x20, // turn heater on
65      HEATER_0 = 0x10 // turn off the heater which is achieved by
                      // resetting the config register
66  };
67
68  /*FUNCTIONS*/
69
70  // calculate the temperature from the sensor
71  float temperature(enum Temp_Type, enum Resolution_Type);
72  // read the unique serial ID
73  int readSerial1(void);
74  int readSerial2(void);
75  int readSerial3(void);
76  // end serial ID functions
77
78
79  int readDeviceID(void);
80  void setConfig(enum HDC_Config_Reg);
81
82  int readConfig(void); // reads the bits of the config register
83
84  //read temps at 14 bit Res
85  float tempFahr(void);
86  float tempCels(void);
87  float calc_humidity(enum Resolution_Type);
88  #endif

```

1.2 hdc1080.c

```
1  /**
2  * @file hdc1080.c
3  * @author Dan Blanchette
4  * @brief Device drivers for the HDC1080 Temperature and Humidity
5  *        Sensor
6  * @version 0.1
7  * @date 2023-02-19
8  *
9  * @copyright Copyright (c) 2023
10 *
11 * CREDITS: James Lasso and Garrett Wells for help with this project
12 *
13 */
14 #include "hdc1080Drivers.h"
15
16 int main()
17 {
18     // Enable UART so we can print status output
19     stdio_init_all(); // Initialize STD I/O for printing over serial
20     // while (!tud_cdc_connected()) { sleep_ms(100); }
21     printf("HDC1080 connected!\n");
22
23     printf("Test Print\n");
24
25     // This example will use I2C1 on the default SDA and SCL pins
26     // Parameter 1 specifies the port address for i2c device, this
27     // value is measured in HZ and is initilaized to 100,000 or 100Khz
28     // Max pico speed is 1Mhz
29     i2c_init(I2C_PORT, 100 * 1000);
30     gpio_set_function(PICO_DEFAULT_I2C_SDA_PIN, GPIO_FUNC_I2C);
31     gpio_set_function(PICO_DEFAULT_I2C_SCL_PIN, GPIO_FUNC_I2C);
32     gpio_pull_up(PICO_DEFAULT_I2C_SDA_PIN);
33     gpio_pull_up(PICO_DEFAULT_I2C_SCL_PIN);
34     // Make the I2C pins available to picotool
35     bi_decl(bi_2pins_with_func(PICO_DEFAULT_I2C_SDA_PIN,
36                               PICO_DEFAULT_I2C_SCL_PIN, GPIO_FUNC_I2C));
37     sleep_ms(1000);
38
39     while (1)
40     {
41         int deviceID, serialID1, serialID2, serialID3, config;
42         float tempF, tempC, perH;
43         deviceID = readDeviceID();
44         serialID1 = readSerial1();
45         serialID2 = readSerial2();
46         serialID3 = readSerial3();
47         config = readConfig();
48         tempC = tempCels();
49         tempF = tempFahr();
50         perH = calc_humidity(HIGH);
51         printf("HDC1080_device: ID=0x%X\n", deviceID);
52         printf("Unique_serial: ID_1=0x%X, ID_2=0x%X, ID_3=0x%X\n",
53               serialID1, serialID2, serialID3);
54     }
55 }
```

```

51     printf("Config Register: 0x%X\n\n", config);
52     printf("Temp Farhenheit: %f\n", tempF);
53     printf("Temp Celsius: %f\n\n", tempC);
54     printf("Percent Humidity: %f\n\n", perH);
55     sleep_ms(1000);
56     // RTOS Scheduler() Here
57 }
58
59 return 0;
60 }
61
62 /*FUNCTION DEFINITIONS*/
63
64 /**
65  * @brief
66  *
67  * @return float
68  */
69 float temperature(enum Temp_Type degrees, enum Resolution_Type
70                  resolution)
71 {
72     // points to address 0x00
73     const uint8_t TEMP_REGISTER = TEMPERATURE;
74     // byte array
75     uint8_t data[2];
76
77     // read just the temperature
78     if (resolution == HIGH)
79     {
80         // for high resolution reading 14 bits
81         setConfig(T_OR_H_14R);
82     }
83     else
84     {
85         // for med resolution reading 11 bits
86         setConfig(TEMP_11R);
87     }
88
89     // get the reading from the temperature sensor
90     int ret = i2c_write_blocking(I2C_PORT, HDC1080ADDRESS, &
91                                TEMP_REGISTER, 1, false);
92
93     if (resolution == HIGH)
94     {
95         sleep_ms(9);
96     }
97     else if (resolution == MED)
98     {
99         sleep_ms(5);
100     }
101
102     ret = i2c_read_blocking(I2C_PORT, HDC1080ADDRESS, data, 2, false);
103
104     int16_t bit_temp_val = data[0] << 8 | data[1];
105     double hex_conv = ((double)bit_temp_val) / ((double)65536);
106     float final_temp = (hex_conv * 165) - 40;

```

```

105
106     if (degrees == CEL)
107     {
108         return final_temp; // degrees C
109     }
110
111     return (final_temp * 1.8) + 32; // degrees F conversion
112 }
113
114 /**
115  * @brief
116  *
117  * @return int
118  */
119 int readDeviceID()
120 {
121     // unsigned integer array that holds the device ID's
122     uint8_t deviceID[2];
123     int ret;
124
125     /*Assign Register*/
126     uint8_t manReg = HDC1080_MANF_DEVICE_ID_REG;
127
128     ret = i2c_write_blocking(I2C_PORT, // type of port
129                             HDC1080ADDRESS, // device address
130                             &manReg, // device's register address to
131
132                             read
133                             1, // expected data size to receive in
134
135                             bytes
136                             false // bool value to tell the I2C
137
138                             controller to: True = use and hold onto the bus, False =
139                             release the bus
140     );
141
142     ret = i2c_read_blocking(I2C_PORT, // type of port
143                             HDC1080ADDRESS, // device address
144                             deviceID, // pass the uint_8 array to receive
145
146                             the data from the register
147                             2, // expected data size to receive in
148
149                             bytes
150                             false // bool value to tell the I2C
151
152                             controller to: True = use the bus, False = release the bus
153     );
154
155     int returnValue = deviceID[0] << 8 | deviceID[1];
156
157     return returnValue;
158 }
159
160 /**
161  * @brief This Block of Functions reads and returns the unique
162  * serial
163  * number from the HDC1080 Device
164  *
165  * @return int
166  */
167 int readSerial1()

```

```

154 {
155     uint8_t deviceID[2];
156     uint8_t serial1 = SERIAL_FIRST;
157
158     int ret1;
159     ret1 = i2c_write_blocking(I2C_PORT, HDC1080ADDRESS, &serial1, 1,
160         false);
161
162     ret1 = i2c_read_blocking(I2C_PORT, HDC1080ADDRESS, deviceID, 2,
163         false);
164
165     int returnVal1 = deviceID[0] << 8 | deviceID[1];
166
167     return returnVal1;
168 }
169
170 int readSerial2()
171 {
172     uint8_t deviceID[2];
173     uint8_t serial2 = SERIAL_MID;
174
175     int ret;
176
177     ret = i2c_write_blocking(I2C_PORT, HDC1080ADDRESS, &serial2, 1,
178         false);
179
180     ret = i2c_read_blocking(I2C_PORT, HDC1080ADDRESS, deviceID, 2,
181         false);
182
183     int returnVal = deviceID[0] << 8 | deviceID[1];
184
185     return returnVal;
186 }
187
188 int readSerial3()
189 {
190     uint8_t deviceID[2];
191     uint8_t serial3 = SERIAL_LAST;
192
193     int ret;
194
195     ret = i2c_write_blocking(I2C_PORT, HDC1080ADDRESS, &serial3, 1,
196         false);
197
198     ret = i2c_read_blocking(I2C_PORT, HDC1080ADDRESS, deviceID, 2,
199         false);
200
201     int returnVal = deviceID[0] << 8 | deviceID[1];
202
203     return returnVal;
204 }
205
206 /**
207  * @brief Set the configuration for the HDC1080's output for
208  *         temperature and humidity
209  *
210  * @param conf_val
211  */

```

```

204 void setConfig(enum HDC_Config_Reg conf_val)
205 {
206     // CONFIG = 0x02
207     const uint8_t configReg = CONFIG;
208     uint8_t set[] = {configReg, conf_val, 0x00};
209
210     // write 3 bytes at a time
211     int value = i2c_write_blocking(I2C_PORT, HDC1080ADDRESS, &set[0],
212                                   3, false);
213 }
214 /**
215  * @brief
216  *
217  * @return int
218  */
219 int readConfig()
220 {
221     int ret;
222     uint8_t configOut[2];
223     uint8_t config = CONFIG;
224
225     ret = i2c_write_blocking(I2C_PORT, HDC1080ADDRESS, &config, 1,
226                             false);
227
228     ret = i2c_read_blocking(I2C_PORT, HDC1080ADDRESS, configOut, 2,
229                             false);
230
231     int returnVal = configOut[0];
232
233     return returnVal;
234 }
235 /**
236  * @brief
237  *
238  * @return float
239  */
240 float tempFahr(void)
241 {
242     return temperature(FAHR, HIGH);
243 }
244 /**
245  * @brief
246  *
247  * @return float
248  */
249 float tempCels(void)
250 {
251     return temperature(CEL, HIGH);
252 }
253
254 float calc_humidity(enum Resolution_Type resolution)
255 {
256     // address 0x01
257     const uint8_t HUMIDITY_REG = HUMIDITY;

```

```

258     uint8_t data[2];
259
260     // set config to read humidity
261     if (resolution == HIGH)
262     {
263         setConfig(T_OR_H_14R);
264     }
265     else if (resolution == MED)
266     {
267         setConfig(HUMID_11R);
268     }
269     else
270     {
271         setConfig(HUMID_8R);
272     }
273
274     // write humidity reading to register
275     int hum_val = i2c_write_blocking(I2C_PORT, HDC1080ADDRESS, &
        HUMIDITY_REG, 1, false);
276
277     if(resolution == HIGH)
278     {
279         sleep_ms(9);
280     }
281     else if (resolution == MED)
282     {
283         sleep_ms(7);
284     }
285     else
286     {
287         sleep_ms(5);
288     }
289
290     hum_val = i2c_read_blocking(I2C_PORT, HDC1080ADDRESS, data, 2,
        false);
291
292     int16_t bit_val = data[0] << 8 | data[1];
293     double hex_conv = ((double) bit_val) / ((double)65536);
294     float tot_hum = hex_conv * 100;
295     return tot_hum;
296 }

```