Annexe S.2

```c
#include "shtc3.h"

#define SHTC3_ADDRESS_READ          (0x70 << 1) | 0x01
#define SHTC3_ADDRESS_WRITE         (0x70 << 1)

#define SHTC3_PRODUCT_CODE_MASK     0x083F
#define SHTC3_SENSOR_ID_MASK        0xF7C0

// NOTE: all commands are "byte swapped" (ntohs), meaning:
// 0x3517 -> 0x1735

#define SHTC3_CMD_WAKEUP                                    0x1735
#define SHTC3_CMD_SLEEP                                     0x98B0
#define SHTC3_CMD_SOFT_RESET                                0x5D80
#define SHTC3_CMD_READ_ID                                   0xC8EF

// Clock stretching based commands
#define SHTC3_CMD_CLK_STRETCH_READ_HUM_FIRST                0x245C
#define SHTC3_CMD_CLK_STRETCH_READ_HUM_FIRST_LOW_POWER      0xDE44

// Polling commands
#define SHTC3_CMD_POLL_HUM_FIRST                            0xE058
#define SHTC3_CMD_POLL_HUM_FIRST_LOW_POWER                  0x1A40


uint16_t shtc3_read_id(I2C_HandleTypeDef *hi2c)
{
  uint8_t data[2];
  uint16_t command = SHTC3_CMD_READ_ID;

  uint32_t res =  HAL_I2C_Master_Transmit(hi2c, SHTC3_ADDRESS_WRITE, (uint8_t*)&command, 2, 100);
  if (res != HAL_OK) {
    return 0;
  }
  res = HAL_I2C_Master_Receive(hi2c, SHTC3_ADDRESS_READ, (uint8_t*)data, 2, 100);
  if (res != HAL_OK) {
    return 0;
  }

  // SHTC3 16 bit ID encoded as:
  // xxxx 1xxx xx00 0111
  // where "x" are actual, sensor ID, while the rest
  // sensor product code (unchangeable)
  uint16_t id = data[0] << 8 | data[1];
  uint16_t code = id & SHTC3_PRODUCT_CODE_MASK;
  if (code == 0x807) {
    // Sensor preset, return actual ID
    return id & SHTC3_SENSOR_ID_MASK;
  }

  return 0;
}

uint32_t shtc3_sleep(I2C_HandleTypeDef *hi2c)
{
  uint16_t command = SHTC3_CMD_SLEEP;
  uint32_t res = HAL_I2C_Master_Transmit(hi2c, SHTC3_ADDRESS_WRITE, (uint8_t*)&command, 2, 100);

  return res == HAL_OK;
}

uint32_t shtc3_wakeup(I2C_HandleTypeDef *hi2c)
{
  uint16_t command = SHTC3_CMD_WAKEUP;
  uint32_t res = HAL_I2C_Master_Transmit(hi2c, SHTC3_ADDRESS_WRITE, (uint8_t*)&command, 2, 100);

  return res == HAL_OK;
}


static uint32_t checkCRC(uint16_t value, uint8_t expected)
{
  uint8_t data[2] = {value >> 8, value & 0xFF};
  uint8_t crc = 0xFF;
  uint8_t poly = 0x31;

  for (uint8_t indi = 0; indi < 2; indi++) {
```

```
78        crc ^= data[indi];
79        for (uint8_t indj = 0; indj < 8; indj++) {
80          if (crc & 0x80) {
81            crc = (uint8_t)((crc << 1) ^ poly);
82          } else {
83            crc <<= 1;
84          }
85        }
86      }
87
88      if (expected ^ crc) {
89        return 0;
90      }
91      return 1;
92    }
93
94    static uint32_t _read_values(uint8_t* data, int32_t* out_temp, int32_t* out_hum)
95    {
96      // Check CRC
97      uint32_t raw_hum = data[0] << 8 | data[1];
98      uint32_t raw_temp = data[3] << 8 | data[4];
99
100     if (!checkCRC(raw_hum, data[2])) {
101       return 0;
102     }
103     if (!checkCRC(raw_temp, data[5])) {
104       return 0;
105     }
106
107     // Convert values
108     if (out_hum) {
109       *out_hum = raw_hum * 10000 / 65535;
110     }
111     if (out_temp) {
112       *out_temp = raw_temp * 17500 / 65535 - 4500;
113     }
114
115     return 1;
116   }
117
118   static uint32_t _perform_measurements(I2C_HandleTypeDef *hi2c, uint16_t command, int32_t* out_temp,
      int32_t* out_hum)
119   {
120     uint8_t result[6];
121
122     uint32_t res = HAL_I2C_Master_Transmit(hi2c, SHTC3_ADDRESS_WRITE, (uint8_t*)&command, 2, 100);
123     if (res != HAL_OK) {
124       return 0;
125     }
126
127     res = HAL_I2C_Master_Receive(hi2c, SHTC3_ADDRESS_READ, result, 6, 100);
128     if (res != HAL_OK) {
129       return 0;
130     }
131
132     return _read_values(result, out_temp, out_hum);
133   }
134
135   uint32_t shtc3_perform_measurements(I2C_HandleTypeDef *hi2c, int32_t* out_temp, int32_t* out_hum)
136   {
137     return _perform_measurements(hi2c, SHTC3_CMD_CLK_STRETCH_READ_HUM_FIRST, out_temp, out_hum);
138   }
139
140   uint32_t shtc3_perform_measurements_low_power(I2C_HandleTypeDef *hi2c, int32_t* out_temp, int32_t*
      out_hum)
141   {
142     return _perform_measurements(hi2c, SHTC3_CMD_CLK_STRETCH_READ_HUM_FIRST_LOW_POWER, out_temp,
      out_hum);
143   }
144
```