Sample Code

For the SHTC1, SHTW2 and STSC1 Humidity and Temperature Sensors

Introduction

This document contains sample code in C for communication with the SHTC1 humidity and temperature sensor. The purpose of the code is to ease the user's software programming when implementing SHTC1 sensors. Besides simple measurement of humidity and temperature, the code contains calculation of CRC checksum and calculation of physical humidity and temperature values. This sample code was written and optimized for the STM32-Discovery board from STMicroelectronics, but it can easily be applied to other microcontrollers with few changes.

For reasons of simplicity, the text below refers to SHTC1, but the document also applies to SHTW2 and STSC1.

If the sample code is used for STSC1, it must be noted that the STSC1 has a different I2C address than SHTC1 and all parts related to humidity measurement can be omitted.

I2C address for SHTC1/SHTW2: 0x70

I2C address for STSC1: 0x4A

1 Structure and Hierarchy of Code

The sample code is structured into various files. The relationship among the different files is given in Figure 1.

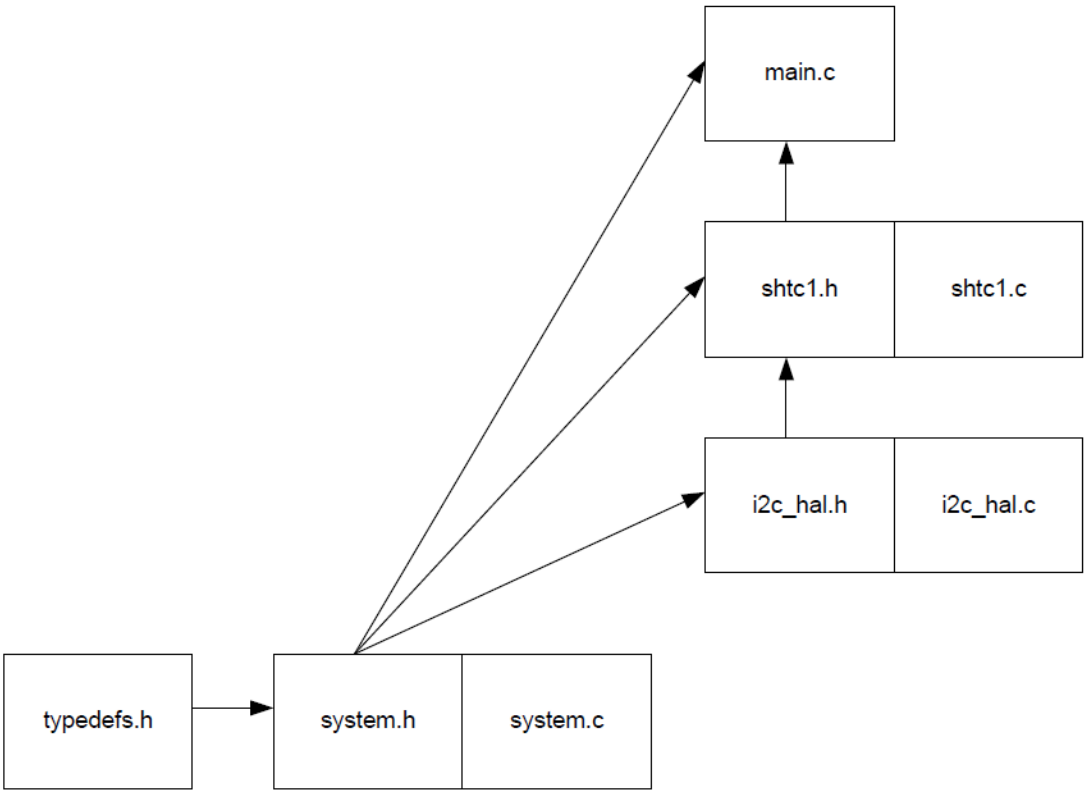

Figure 1 Structure of sample code for SHTC1.

2 Sample Code

Below is the C code for the different files. The code was written and optimized for the STM32-Discovery board from STMicroelectronics (STM32VLDISCOVERY) and can be easily adapted to other microcontrollers. The portions that need to be adapted for porting to a different microcontroller are indicated in the comments.

2.1 main.c

```
//===========================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//===========================================================================
// Project   :  SHTC1 Sample Code (V1.0)
// File      :  main.c (V1.0)
// Author    :  RFU
// Date      :  26-Nov-2012
// Controller:  STM32F100RB
// IDE       :  µVision V4.60.0.0
// Compiler  :  Armcc
// Brief     :  This code shows how to implement the basic commands for the
//              SHTC1 sensor chip.
//              Due to compatibility reasons the I2C interface is implemented
//              as "bit-banging" on normal I/O's. This code is written for an
//              easy understanding and is neither optimized for speed nor code
//              size.
//
// Porting to a different microcontroller (uC):
//   - the definitions of basic types may have to be changed  in typedefs.h
//   - adapt the button and led functions for your platform   in main.c
//   - adapt the port functions / definitions for your uC     in i2c_hal.h/.c
//   - adapt the timing of the delay function for your uC     in system.c
//   - adapt the SystemInit()                                 in system.c
//   - change the uC register definition file <stm32f10x.h>   in system.h
//===========================================================================

//-- Includes ---------------------------------------------------------------
#include "system.h"
#include "shtc1.h"

//===========================================================================
void Led_Init(void){                /* -- adapt this code for your platform -- */
//===========================================================================
  RCC->APB2ENR |= 0x00000010;  // I/O port C clock enabled
  GPIOC->CRH   &= 0xFFFFFF00;  // set general purpose output mode for LEDs
  GPIOC->CRH   |= 0x00000011;  //
  GPIOC->BSRR   = 0x03000000;  // LEDs off
}

//===========================================================================
void UserButton_Init(void){         /* -- adapt this code for your platform -- */
//===========================================================================
  RCC->APB2ENR |= 0x00000004;  // I/O port A clock enabled
  GPIOA->CRH   &= 0xFFFFFFF0;  // set general purpose input mode for User Button
  GPIOA->CRH   |= 0x00000004;  //
}

//===========================================================================
void LedBlueOn(void){               /* -- adapt this code for your platform -- */
//===========================================================================
  GPIOC->BSRR = 0x00000100;
}

//===========================================================================
void LedBlueOff(void){              /* -- adapt this code for your platform -- */
//===========================================================================
  GPIOC->BSRR = 0x01000000;
}

//===========================================================================
void LedGreenOn(void){              /* -- adapt this code for your platform -- */
//===========================================================================
  GPIOC->BSRR = 0x00000200;
}

//===========================================================================
void LedGreenOff(void){             /* -- adapt this code for your platform -- */
//===========================================================================
  GPIOC->BSRR = 0x02000000;
}

//===========================================================================
u8t ReadUserButton(void){           /* -- adapt this code for your platform -- */
//===========================================================================
  return (GPIOA->IDR & 0x00000001);
```

```
}

//===========================================================================
int main(void){
//===========================================================================
  etError  error;       // error code
  u16t     id;          // sensor ID
  ft       temperature; // temperature
  ft       humidity;    // relative humidity

  SystemInit();
  Led_Init();
  UserButton Init();
  SHTC1_Init();

  while(1)
  {
    error = NO_ERROR;

    if(ReadUserButton() == 0)
    // if the user button is not pressed
    {
      // read sensor ID
      error |= SHTC1_GetId(&id);

      // read temperature and relative humidity
      error |= SHTC1_GetTempAndHumiPolling(&temperature, &humidity);

      // the blue LED lights up when the relative humidity is greater than 50%
      if(humidity > 50) LedBlueOn();
      else              LedBlueOff();
    }
    else
    // if the user button is pressed
    {
      // disable green and blue LED
      LedGreenOff();
      LedBlueOff();

      // perform a soft reset on the sensor
      error |= SHTC1_SoftReset();

      // wait until button is released
      while(ReadUserButton() != 0);
    }

    // the green LED lights if no error occurs
    if(!error) LedGreenOn();
    else       LedGreenOff();

    // wait 100ms
    DelayMicroSeconds(100000);
  }
}
```

2.2 shtc1.h

```
//===========================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//===========================================================================
// Project   :  SHTC1 Sample Code (V1.0)
// File      :  shtc1.h (V1.0)
// Author    :  RFU
// Date      :  26-Nov-2012
// Controller:  STM32F100RB
// IDE       :  µVision V4.60.0.0
// Compiler  :  Armcc
// Brief     :  Sensor Layer: Definitions of commands and functions for sensor
//                            access.
//===========================================================================

#ifndef SHTC1_H
#define SHTC1_H

//-- Includes ---------------------------------------------------------------
#include "system.h"

//-- Defines ----------------------------------------------------------------
// CRC
#define POLYNOMIAL  0x131 // P(x) = x^8 + x^5 + x^4 + 1 = 100110001

//-- Enumerations -----------------------------------------------------------
// Sensor Commands
typedef enum{
  READ_ID            = 0xEFC8, // command: read ID register
  SOFT_RESET         = 0x805D, // soft reset
```

```
  MEAS_T_RH_POLLING  = 0x7866, // meas. read T first, clock stretching disabled
  MEAS_T_RH_CLOCKSTR = 0x7CA2, // meas. read T first, clock stretching enabled
  MEAS_RH_T_POLLING  = 0x58E0, // meas. read RH first, clock stretching disabled
  MEAS_RH_T_CLOCKSTR = 0x5C24  // meas. read RH first, clock stretching enabled
}etCommands;

// I2C address
typedef enum{
  I2C_ADR_W         = 0xE0,   // sensor I2C address + write bit
  I2C_ADR_R         = 0xE1    // sensor I2C address + read bit
}etI2cHeader;

//===========================================================================
void SHTC1_Init(void);
//===========================================================================
// Initializes the I2C bus for communication with the sensor.
//---------------------------------------------------------------------------

//===========================================================================
etError SHTC1_GetId(u16t *id);
//===========================================================================
// Gets the ID from the sensor.
//---------------------------------------------------------------------------
// input:  *id          pointer to a integer, where the id will be stored
//
// return: error:       ACK ERROR      = no acknowledgment from sensor
//                      CHECKSUM ERROR = checksum mismatch
//                      NO_ERROR       = no error

//===========================================================================
etError SHTC1_GetTempAndHumi(ft *temp, ft *humi);
//===========================================================================
// Gets the temperature [°C] and the humidity [%RH].
//---------------------------------------------------------------------------
// input:  *temp        pointer to a floating point value, where the calculated
//                      temperature will be stored
//         *humi        pointer to a floating point value, where the calculated
//                      humidity will be stored
//
// return: error:       ACK ERROR      = no acknowledgment from sensor
//                      CHECKSUM_ERROR = checksum mismatch
//                      NO_ERROR       = no error
//
// remark: If you use this function, then the sensor blocks the I2C-bus with
//         clock stretching during the measurement.

//===========================================================================
etError SHTC1_GetTempAndHumiPolling(ft *temp, ft *humi);
//===========================================================================
// Gets the temperature [°C] and the humidity [%RH]. This function polls every
// 1ms until measurement is ready.
//---------------------------------------------------------------------------
// input:  *temp        pointer to a floating point value, where the calculated
//                      temperature will be stored
//         *humi        pointer to a floating point value, where the calculated
//                      humidity will be stored
//
// return: error:       ACK ERROR      = no acknowledgment from sensor
//                      CHECKSUM_ERROR = checksum mismatch
//                      NO_ERROR       = no error

//===========================================================================
etError SHTC1_SoftReset(void);
//===========================================================================
// Calls the soft reset mechanism that forces the sensor into a well-defined
// state without removing the power supply.
//---------------------------------------------------------------------------
// return: error:       ACK_ERROR      = no acknowledgment from sensor
//                      NO_ERROR       = no error

//===========================================================================
etError SHTC1_StartWriteAccess(void);
//===========================================================================
// Writes a start condition and the sensor I2C address with the write flag.
//---------------------------------------------------------------------------
// return: error:       ACK_ERROR      = no acknowledgment from sensor
//                      NO_ERROR       = no error

//===========================================================================
etError SHTC1_StartReadAccess(void);
//===========================================================================
// Writes a start condition and the sensor I2C address with the read flag.
//---------------------------------------------------------------------------
// return: error:       ACK_ERROR      = no acknowledgment from sensor
//                      NO_ERROR       = no error

//===========================================================================
void SHTC1_StopAccess(void);
//===========================================================================
```

```
// Writes a stop condition.
//-----------------------------------------------------------------------------

//=============================================================================
etError SHTC1_Read2BytesAndCrc(u16t *data);
//=============================================================================
// Reads two bytes plus the checksum and verifies this. If the checksum
// verification is successful, then the two bytes stored in a 16-bit integer.
//-----------------------------------------------------------------------------
// input:  *data        pointer to a 16-bit int, where the data will be stored
//
// return: error:       CHECKSUM_ERROR = checksum does not match
//                      NO_ERROR       = checksum matches

//=============================================================================
etError SHTC1_WriteCommand(etCommands cmd);
//=============================================================================
// Writes command to the sensor.
//-----------------------------------------------------------------------------
// input:  cmd          command which is to be written to the sensor
//
// return: error:       ACK_ERROR = no acknowledgment from sensor
//                      NO_ERROR  = no error

//=============================================================================
etError SHTC1_CheckCrc(u8t data[], u8t nbrOfBytes, u8t checksum);
//=============================================================================
// Calculates checksum for n bytes of data and compares it with expected
// checksum.
//-----------------------------------------------------------------------------
// input:  data[]       checksum is built based on this data
//         nbrOfBytes   checksum is built for n bytes of data
//         checksum     expected checksum
//
// return: error:       CHECKSUM_ERROR = checksum does not match
//                      NO_ERROR       = checksum matches

//=============================================================================
ft SHTC1_CalcTemperature(u16t rawValue);
//=============================================================================
// Calculates the temperature [°C] as a floating point value from the raw data
// that are read from the sensor.
//-----------------------------------------------------------------------------
// input:  rawValue     temperature raw value (16bit scaled)
//
// return:              temperature [°C] as a floating point value

//=============================================================================
ft SHTC1_CalcHumidity(u16t rawValue);
//=============================================================================
// Calculates the relative humidity [%RH] as a floating point value from the raw
// data that are read from the sensor.
//-----------------------------------------------------------------------------
// input:  rawValue     humidity raw value (16bit scaled)
//
// return:              relative humidity [%RH] as a floating point value

#endif
```

2.3 shtc1.c

```
//=============================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//=============================================================================
// Project   :  SHTC1 Sample Code (V1.0)
// File      :  shtc1.c (V1.0)
// Author    :  RFU
// Date      :  26-Nov-2012
// Controller:  STM32F100RB
// IDE       :  µVision V4.60.0.0
// Compiler  :  Armcc
// Brief     :  Sensor Layer: Implementation of functions for sensor access.
//=============================================================================

//-- Includes -----------------------------------------------------------------
#include "shtc1.h"
#include "i2c_hal.h"

//=============================================================================
void SHTC1 Init(void){
//=============================================================================
  I2c_Init(); // init I2C
}

//=============================================================================
etError SHTC1_GetTempAndHumi(ft *temp, ft *humi){
```

```
//===========================================================================
  etError error;        // error code
  u16t    rawValueTemp; // temperature raw value from sensor
  u16t    rawValueHumi; // humidity raw value from sensor

  error = SHTC1_StartWriteAccess();

  // measure, read temperature first, clock streching enabled
  error |= SHTC1_WriteCommand(MEAS_T_RH_CLOCKSTR);

  // if no error, read temperature and humidity raw values
  if(error == NO_ERROR)
  {
    error |= SHTC1 StartReadAccess();
    error |= SHTC1 Read2BytesAndCrc(&rawValueTemp);
    error |= SHTC1_Read2BytesAndCrc(&rawValueHumi);
  }

  SHTC1_StopAccess();

  // if no error, calculate temperature in °C and humidity in %RH
  if(error == NO_ERROR)
  {
    *temp = SHTC1_CalcTemperature(rawValueTemp);
    *humi = SHTC1_CalcHumidity(rawValueHumi);
  }

  return error;
}

//===========================================================================
etError SHTC1_GetTempAndHumiPolling(ft *temp, ft *humi){
//===========================================================================
  etError error;           // error code
  u8t     maxPolling = 20; // max. retries to read the measurement (polling)
  u16t    rawValueTemp;    // temperature raw value from sensor
  u16t    rawValueHumi;    // humidity raw value from sensor

  error  = SHTC1_StartWriteAccess();

  // measure, read temperature first, clock streching disabled (polling)
  error |= SHTC1_WriteCommand(MEAS_T_RH_POLLING);

  // if no error, ...
  if(error == NO_ERROR)
  {
    // poll every 1ms for measurement ready
    while(maxPolling--)
    {
      // check if the measurement has finished
      error = SHTC1_StartReadAccess();

      // if measurement has finished -> exit loop
      if(error == NO_ERROR) break;

      // delay 1ms
      DelayMicroSeconds(1000);
    }

    // if no error, read temperature and humidity raw values
    if(error == NO_ERROR)
    {
      error |= SHTC1_Read2BytesAndCrc(&rawValueTemp);
      error |= SHTC1_Read2BytesAndCrc(&rawValueHumi);
    }
  }

  SHTC1_StopAccess();

  // if no error, calculate temperature in °C and humidity in %RH
  if(error == NO_ERROR)
  {
    *temp = SHTC1_CalcTemperature(rawValueTemp);
    *humi = SHTC1_CalcHumidity(rawValueHumi);
  }

  return error;
}

//===========================================================================
etError SHTC1_GetId(u16t *id){
//===========================================================================
  etError error; // error code

  error = SHTC1_StartWriteAccess();

  // write ID read command
  error |= SHTC1_WriteCommand(READ_ID);
```

```
  // if no error, read ID
  if(error == NO_ERROR)
  {
    SHTC1_StartReadAccess();
    error = SHTC1_Read2BytesAndCrc(id);
  }

  SHTC1_StopAccess();

  return error;
}

//==============================================================================
etError SHTC1 SoftReset(void){
//==============================================================================
  etError error; // error code

  error = SHTC1_StartWriteAccess();

  // write reset command
  error |= SHTC1_WriteCommand(SOFT_RESET);

  SHTC1_StopAccess();

  return error;
}

//==============================================================================
etError SHTC1_StartWriteAccess(void){
//==============================================================================
  etError error; // error code

  // write a start condition
  I2c_StartCondition();

  // write the sensor I2C address with the write flag
  error = I2c_WriteByte(I2C_ADR_W);

  return error;
}

//==============================================================================
etError SHTC1_StartReadAccess(void){
//==============================================================================
  etError error; // error code

  // write a start condition
  I2c_StartCondition();

  // write the sensor I2C address with the read flag
  error = I2c_WriteByte(I2C_ADR_R);

  return error;
}

//==============================================================================
void SHTC1_StopAccess(void){
//==============================================================================
  // write a stop condition
  I2c_StopCondition();
}

//==============================================================================
etError SHTC1_WriteCommand(etCommands cmd){
//==============================================================================
  etError error; // error code

  // write the upper 8 bits of the command to the sensor
  error  = I2c_WriteByte(cmd >> 8);

  // write the lower 8 bits of the command to the sensor
  error |= I2c_WriteByte(cmd & 0xFF);

  return error;
}

//==============================================================================
etError SHTC1_Read2BytesAndCrc(u16t *data){
//==============================================================================
  etError error;    // error code
  u8t     bytes[2]; // read data array
  u8t     checksum; // checksum byte

  // read two data bytes and one checksum byte
  bytes[0] = I2c_ReadByte(ACK);
  bytes[1] = I2c_ReadByte(ACK);
  checksum = I2c_ReadByte(ACK);

  // verify checksum
```

```
  error = SHTC1_CheckCrc(bytes, 2, checksum);

  // combine the two bytes to a 16-bit value
  *data = (bytes[0] << 8) | bytes[1];

  return error;
}

//==========================================================================
etError SHTC1_CheckCrc(u8t data[], u8t nbrOfBytes, u8t checksum){
//==========================================================================
  u8t bit;        // bit mask
  u8t crc = 0xFF; // calculated checksum
  u8t byteCtr;    // byte counter

  // calculates 8-Bit checksum with given polynomial
  for(byteCtr = 0; byteCtr < nbrOfBytes; byteCtr++)
  {
    crc ^= (data[byteCtr]);
    for(bit = 8; bit > 0; --bit)
    {
      if(crc & 0x80) crc = (crc << 1) ^ POLYNOMIAL;
      else           crc = (crc << 1);
    }
  }

  // verify checksum
  if(crc != checksum) return CHECKSUM_ERROR;
  else                return NO_ERROR;
}

//==========================================================================
ft SHTC1 CalcTemperature(u16t rawValue){
//==========================================================================
  // calculate temperature [°C]
  // T = -45 + 175 * rawValue / 2^16
  return 175 * (ft)rawValue / 65536 - 45;
}

//==========================================================================
ft SHTC1_CalcHumidity(u16t rawValue){
//==========================================================================
  // calculate relative humidity [%RH]
  // RH = rawValue / 2^16 * 100
  return 100 * (ft)rawValue / 65536;
}
```

2.4 i2c_hal.h

```
//==========================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//==========================================================================
// Project   :  SHTC1 Sample Code (V1.0)
// File      :  i2c_hal.h (V1.0)
// Author    :  RFU
// Date      :  26-Nov-2012
// Controller:  STM32F100RB
// IDE       :  µVision V4.60.0.0
// Compiler  :  Armcc
// Brief     :  I2C hardware abstraction layer
//==========================================================================

#ifndef I2C_HAL_H
#define I2C_HAL_H

//-- Includes ------------------------------------------------------------
#include "system.h"

//-- Defines -------------------------------------------------------------
// I2C IO-Pins                          /* -- adapt the defines for your uC -- */
// SDA on port B, bit 9
#define SDA_LOW()   (GPIOB->BSRR = 0x02000000) // set SDA to low
#define SDA_OPEN()  (GPIOB->BSRR = 0x00000200) // set SDA to open-drain
#define SDA_READ    (GPIOB->IDR  & 0x0200)     // read SDA

// SCL on port B, bit 8                 /* -- adapt the defines for your uC -- */
#define SCL_LOW()   (GPIOB->BSRR = 0x01000000) // set SCL to low
#define SCL_OPEN()  (GPIOB->BSRR = 0x00000100) // set SCL to open-drain
#define SCL_READ    (GPIOB->IDR  & 0x0100)     // read SCL

//-- Enumerations --------------------------------------------------------

// I2C acknowledge
typedef enum{
  ACK    = 0,
  NO_ACK = 1,
```

```
}etI2cAck;

//==============================================================================
void I2c_Init(void);
//==============================================================================
// Initializes the ports for I2C interface.
//------------------------------------------------------------------------------

//==============================================================================
void I2c_StartCondition(void);
//==============================================================================
// Writes a start condition on I2C-Bus.
//------------------------------------------------------------------------------
// remark: Timing (delay) may have to be changed for different microcontroller.
//
// SDA:       |_____
//         _______
// SCL:          |___

//==============================================================================
void I2c_StopCondition(void);
//==============================================================================
// Writes a stop condition on I2C-Bus.
//------------------------------------------------------------------------------
// remark: Timing (delay) may have to be changed for different microcontroller.
//
// SDA:         |
//                _____
// SCL:    ___|

//==============================================================================
etError I2c_WriteByte(u8t txByte);
//==============================================================================
// Writes a byte to I2C-Bus and checks acknowledge.
//------------------------------------------------------------------------------
// input:  txByte       transmit byte
//
// return: error:       ACK_ERROR = no acknowledgment from sensor
//                      NO_ERROR  = no error
//
// remark: Timing (delay) may have to be changed for different microcontroller.

//==============================================================================
u8t I2c_ReadByte(etI2cAck ack);
//==============================================================================
// Reads a byte on I2C-Bus.
//------------------------------------------------------------------------------
// input:  ack          Acknowledge: ACK or NO_ACK
//
// return: rxByte
//
// remark: Timing (delay) may have to be changed for different microcontroller.

#endif
```

2.5 i2c_hal.c

```
//==============================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//==============================================================================
// Project   :  SHTC1 Sample Code (V1.0)
// File      :  i2c_hal.c (V1.0)
// Author    :  RFU
// Date      :  26-Nov-2012
// Controller:  STM32F100RB
// IDE       :  µVision V4.60.0.0
// Compiler  :  Armcc
// Brief     :  I2C hardware abstraction layer
//==============================================================================

//-- Includes ------------------------------------------------------------------
#include "i2c_hal.h"

//==============================================================================
void I2c_Init(void){                        /* -- adapt the init for your uC -- */
//==============================================================================
  RCC->APB2ENR |= 0x00000008;  // I/O port B clock enabled

  GPIOB->CRH   &= 0xFFFFFF00;  // set open-drain output for SDA and SCL
  GPIOB->CRH   |= 0x00000055;  //

  SDA_OPEN();                  // I2C-bus idle mode SDA released
  SCL_OPEN();                  // I2C-bus idle mode SCL released
}

//==============================================================================
```

```
void I2c StartCondition(void){
//===========================================================================
  SDA_OPEN();
  DelayMicroSeconds(1);
  SCL_OPEN();
  DelayMicroSeconds(1);
  SDA_LOW();
  DelayMicroSeconds(10);  // hold time start condition (t_HD;STA)
  SCL_LOW();
  DelayMicroSeconds(10);
}

//===========================================================================
void I2c StopCondition(void){
//===========================================================================
  SCL_LOW();
  DelayMicroSeconds(1);
  SDA_LOW();
  DelayMicroSeconds(1);
  SCL OPEN();
  DelayMicroSeconds(10);  // set-up time stop condition (t_SU;STO)
  SDA_OPEN();
  DelayMicroSeconds(10);
}

//===========================================================================
etError I2c WriteByte(u8t txByte){
//===========================================================================
  u8t     mask;
  etError error = NO_ERROR;
  for(mask = 0x80; mask > 0; mask >>= 1)// shift bit for masking (8 times)
  {
    if((mask & txByte) == 0) SDA LOW(); // masking txByte, write bit to SDA-Line
    else                     SDA_OPEN();
    DelayMicroSeconds(1);               // data set-up time (t_SU;DAT)
    SCL_OPEN();                         // generate clock pulse on SCL
    DelayMicroSeconds(5);               // SCL high time (t_HIGH)
    SCL_LOW();
    DelayMicroSeconds(1);               // data hold time(t_HD;DAT)
  }
  SDA_OPEN();                           // release SDA-line
  SCL_OPEN();                           // clk #9 for ack
  DelayMicroSeconds(1);                 // data set-up time (t_SU;DAT)
  if(SDA_READ) error = ACK_ERROR;       // check ack from i2c slave
  SCL LOW();
  DelayMicroSeconds(20);                // wait to see byte package on scope
  return error;                         // return error code
}

//===========================================================================
u8t I2c_ReadByte(etI2cAck ack){
//===========================================================================
  u8t mask;
  u8t rxByte = NO_ERROR;
  SDA_OPEN();                           // release SDA-line
  for(mask = 0x80; mask > 0; mask >>= 1) // shift bit for masking (8 times)
  {
    SCL OPEN();                         // start clock on SCL-line
    DelayMicroSeconds(1);               // data set-up time (t_SU;DAT)
    while(SCL_READ == 0);
    DelayMicroSeconds(3);               // SCL high time (t_HIGH)
    if(SDA_READ) rxByte = rxByte | mask; // read bit
    SCL_LOW();
    DelayMicroSeconds(1);               // data hold time(t_HD;DAT)
  }
  if(ack == ACK) SDA LOW();             // send acknowledge if necessary
  else           SDA_OPEN();
  DelayMicroSeconds(1);                 // data set-up time (t_SU;DAT)
  SCL_OPEN();                           // clk #9 for ack
  DelayMicroSeconds(5);                 // SCL high time (t_HIGH)
  SCL LOW();
  SDA_OPEN();                           // release SDA-line
  DelayMicroSeconds(20);                // wait to see byte package on scope
  return rxByte;                        // return error code
}
```

2.6 system.h

```
//===========================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//===========================================================================
// Project   :  SHTC1 Sample Code (V1.0)
// File      :  system.h (V1.0)
// Author    :  RFU
// Date      :  26-Nov-2012
// Controller:  STM32F100RB
```

```
// IDE       :  µVision V4.60.0.0
// Compiler  :  Armcc
// Brief     :  System functions, global definitions
//==============================================================================

#ifndef SYSTEM_H
#define SYSTEM_H

//-- Includes ------------------------------------------------------------------
#include <stm32f10x.h>             // controller register definitions
#include "typedefs.h"              // type definitions

//-- Enumerations --------------------------------------------------------------
// Error codes
typedef enum{
  NO_ERROR       = 0x00, // no error
  ACK_ERROR      = 0x01, // no acknowledgment error
  CHECKSUM_ERROR = 0x02  // checksum mismatch error
}etError;

//==============================================================================
void SystemInit(void);
//==============================================================================
// Initializes the system
//------------------------------------------------------------------------------

//==============================================================================
void DelayMicroSeconds(u32t nbrOfUs);
//==============================================================================
// Wait function for small delays.
//------------------------------------------------------------------------------
// input:  nbrOfUs   wait x times approx. one micro second (fcpu = 8MHz)
// return: -
// remark: smallest delay is approx. 15us due to function call

#endif
```

2.7 system.c

```
//==============================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//==============================================================================
// Project   :  SHTC1 Sample Code (V1.0)
// File      :  system.c (V1.0)
// Author    :  RFU
// Date      :  26-Nov-2012
// Controller:  STM32F100RB
// IDE       :  µVision V4.60.0.0
// Compiler  :  Armcc
// Brief     :  System functions
//==============================================================================

//-- Includes ------------------------------------------------------------------
#include "system.h"

//==============================================================================
void SystemInit(void){
//==============================================================================
  // no initialization required
}

//==============================================================================
void DelayMicroSeconds(u32t nbrOfUs){   /* -- adapt this delay for your uC -- */
//==============================================================================
  u32t i;
  for(i = 0; i < nbrOfUs; i++)
  {
    __nop();  // nop's may be added or removed for timing adjustment
    __nop();
    __nop();
    __nop();
  }
}
```

2.8 typedefs.h

```
//==============================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//==============================================================================
// Project   :  SHTC1 Sample Code (V1.0)
// File      :  typedefs.h (V1.0)
// Author    :  RFU
// Date      :  26-Nov-2012
```

```
// Controller:  STM32F100RB
// IDE       :  µVision V4.60.0.0
// Compiler  :  Armcc
// Brief     :  Definitions of typedefs for good readability and portability.
//==============================================================================

#ifndef TYPEDEFS_H
#define TYPEDEFS_H

//-- Defines -------------------------------------------------------------------
//Processor endian system
//#define BIG ENDIAN   //e.g. Motorola (not tested at this time)
#define LITTLE ENDIAN  //e.g. PIC, 8051, NEC V850
//==============================================================================
// basic types: making the size of types clear
//==============================================================================
typedef unsigned char   u8t;      ///< range: 0 .. 255
typedef signed char     i8t;      ///< range: -128 .. +127

typedef unsigned short  u16t;     ///< range: 0 .. 65535
typedef signed short    i16t;     ///< range: -32768 .. +32767

typedef unsigned long   u32t;     ///< range: 0 .. 4'294'967'295
typedef signed long     i32t;     ///< range: -2'147'483'648 .. +2'147'483'647

typedef float           ft;       ///< range: +-1.18E-38 .. +-3.39E+38
typedef double          dt;       ///< range:            .. +-1.79E+308

typedef enum{
  FALSE     = 0,
  TRUE      = 1
}bt;

typedef union {
  u16t u16;               // element specifier for accessing whole u16
  i16t i16;               // element specifier for accessing whole i16
  struct {
    #ifdef LITTLE_ENDIAN  // Byte-order is little endian
    u8t u8L;              // element specifier for accessing low u8
    u8t u8H;              // element specifier for accessing high u8
    #else                 // Byte-order is big endian
    u8t u8H;              // element specifier for accessing low u8
    u8t u8L;              // element specifier for accessing high u8
    #endif
  } s16;                  // element spec. for acc. struct with low or high u8
} nt16;

typedef union {
  u32t u32;               // element specifier for accessing whole u32
  i32t i32;               // element specifier for accessing whole i32
 struct {
    #ifdef LITTLE ENDIAN  // Byte-order is little endian
    u16t u16L;            // element specifier for accessing low u16
    u16t u16H;            // element specifier for accessing high u16
    #else                 // Byte-order is big endian
    u16t u16H;            // element specifier for accessing low u16
    u16t u16L;            // element specifier for accessing high u16
    #endif
  } s32;                  // element spec. for acc. struct with low or high u16
} nt32;

#endif
```

Revision History

Date	Version	Page(s)	Changes
17 June, 2013	1	All	Initial release
29. January 2015	2	All	Added SHTW1 and STSC1
13. May 2016	3	All	Added SHTW2, removed SHTW1

Headquarters and Subsidiaries

Sensirion AG
Laubisruetistr. 50
CH-8712 Staefa ZH
Switzerland

Phone: +41 44 306 40 00
Fax: +41 44 306 40 30
info@sensirion.com
www.sensirion.com

Sensirion AG (Germany)
Phone: +41 44 927 11 66
info@sensirion.com
www.sensirion.com

Sensirion Inc., USA
Phone: +1 805 409 4900
info_us@sensirion.com
www.sensirion.com

Sensirion Japan Co. Ltd.
Phone: +81 3 3444 4940
info@sensirion.co.jp
www.sensirion.co.jp

Sensirion Korea Co. Ltd.
Phone: +82 31 345 0031 3
info@sensirion.co.kr
www.sensirion.co.kr

Sensirion China Co. Ltd.
Phone: +86 755 8252 1501
info@sensirion.com.cn
www.sensirion.com.cn

To find your local representative, please visit www.sensirion.com/contact