Sample Code I²C

For communication with the SHT3x Humidity and Temperature Sensor through the I²C Interface

Introduction

This document contains sample code in C for communication with the SHT3x humidity and temperature sensor through the I²C Interface. The purpose of the code is to ease the user's software programming when implementing SHT3x 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.

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.

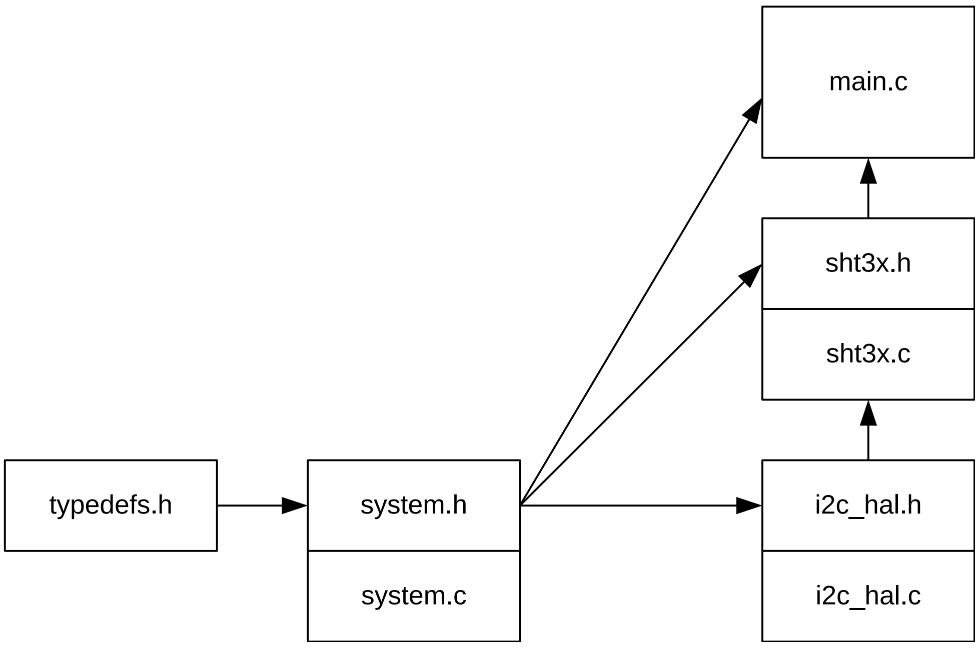

Figure 1 Structure of sample code for SHT3x

Table of Contents

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   :  SHT3x Sample Code (V1.1)
// File      :  main.c (V1.1)
// Author    :  RFU
// Date      :  6-Mai-2015
// Controller:  STM32F100RB
// IDE       :  µVision V5.12.0.0
// Compiler  :  Armcc
// Brief     :  This code shows how to implement the basic commands for the
//              SHT3x 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 "sht3x.h"

//-- Static function prototypes ------------------------------------------------
static void EvalBoardPower_Init(void);
static void Led_Init(void);
static void UserButton_Init(void);
static void LedBlueOn(void);
static void LedBlueOff(void);
static void LedGreenOn(void);
static void LedGreenOff(void);
static u8t ReadUserButton(void);

//------------------------------------------------------------------------------
int main(void)
{
  etError   error;       // error code
  u32t      serialNumber;// serial number
  regStatus status;      // sensor status
  ft        temperature; // temperature [°C]
  ft        humidity;    // relative humidity [%RH]
  bt        heater;      // heater, false: off, true: on

  SystemInit();
  Led_Init();
  UserButton_Init();
  EvalBoardPower_Init();

  SHT3X_Init(0x45); // Address: 0x44 = Sensor on EvalBoard connector
                    //          0x45 = Sensor on EvalBoard

  // wait 50ms after power on
  DelayMicroSeconds(50000);

  error = SHT3x_ReadSerialNumber(&serialNumber);
  if(error != NO_ERROR){} // do error handling here

  // demonstrate a single shot measurement with clock-stretching
  error = SHT3X_GetTempAndHumi(&temperature, &humidity, REPEATAB_HIGH,
MODE_CLKSTRETCH, 50);
  if(error != NO_ERROR){} // do error handling here

  // demonstrate a single shot measurement with polling and 50ms timeout
  error = SHT3X_GetTempAndHumi(&temperature, &humidity, REPEATAB_HIGH, MODE_POLLING,
50);
  if(error != NO_ERROR){} // do error handling here

  // loop forever
  while(1)
  {
    error = NO_ERROR;

    // loop while no error
    while(error == NO_ERROR)
    {
      // read status register
      error |= SHT3X_ReadStatus(&status.u16);
      if(error != NO_ERROR) break;

      // check if the reset bit is set after a reset or power-up
      if(status.bit.ResetDetected)
      {
        //override default temperature and humidity alert limits (red LED)
        error = SHT3X_SetAlertLimits( 70.0f,  50.0f,  // high set:   RH [%], T [°C]
                                      68.0f,  48.0f,  // high clear: RH [%], T [°C]
                                      32.0f,  -2.0f,  // low clear:  RH [%], T [°C]
                                      30.0f,  -4.0f); // low set:    RH [%], T [°C]
                if(error != NO_ERROR) break;


        // clear reset and alert flags
        error = SHT3X_ClearAllAlertFlags();
        if(error != NO_ERROR) break;

        //start periodic measurement, with high repeatability and 1 measurements per
second
        error = SHT3X_StartPeriodicMeasurment(REPEATAB_HIGH, FREQUENCY_1HZ);
        if(error != NO_ERROR) break;

        //switch green LED on
        LedGreenOn();
      }

      // read measurment buffer
      error = SHT3X_ReadMeasurementBuffer(&temperature, &humidity);
      if(error == NO_ERROR)
      {
        // flash blue LED to signalise new temperature and humidity values
        LedBlueOn();
        DelayMicroSeconds(10000);
        LedBlueOff();
      }
      else if (error == ACK_ERROR)
      {
        // there were no new values in the buffer -> ignore this error
        error = NO_ERROR;
      }
      else break;

      // read heater status
      heater = status.bit.HeaterStatus ? TRUE : FALSE;

      // if the user button is not pressed ...
      if(ReadUserButton() == 0)
      {
         // ... and the heater is on
         if(heater)
         {
           // switch off the sensor internal heater
           error |= SHT3X_DisableHeater();
           if(error != NO_ERROR) break;
         }
      }
      else
      // if the user button is pressed ...
      {
         // ... and the heater is off
         if(!heater)
         {
           // switch on the sensor internal heater
           error |= SHT3X_EnableHeater();
           if(error != NO_ERROR) break;
         }
      }

      // wait 100ms
      DelayMicroSeconds(100000);
    }

    // in case of an error ...

    // ... switch green and blue LED off
    LedGreenOff();
    LedBlueOff();

    // ... try first a soft reset ...
    error = SHT3X_SoftReset();

    // ... if the soft reset fails, do a hard reset
    if(error != NO_ERROR)
    {
      SHT3X_HardReset();
    }

    // flash green LED to signalise an error
    LedGreenOn();
    DelayMicroSeconds(10000);
    LedGreenOff();
  }
}
```

```
//-----------------------------------------------------------------------------
static void EvalBoardPower_Init(void)    /* -- adapt this code for your platform --
*/
{
  RCC->APB2ENR |= 0x00000008;  // I/O port B clock enabled

  GPIOB->CRH   &= 0x0FFF0FFF;  // set push-pull output for Vdd & GND pins
  GPIOB->CRH   |= 0x10001000;  //

  GPIOB->BSRR = 0x08008000;    // set Vdd to High, set GND to Low
}

//-----------------------------------------------------------------------------
static 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
}

//-----------------------------------------------------------------------------
static 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;  //
}

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

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

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

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

//-----------------------------------------------------------------------------
```

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

}
```

2.2 sht3x.h

```
//==============================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//==============================================================================
// Project   :  SHT3x Sample Code (V1.1)
// File      :  sht3x.h (V1.1)
// Author    :  RFU
// Date      :  6-Mai-2015
// Controller:  STM32F100RB
// IDE       :  µVision V5.12.0.0
// Compiler  :  Armcc
// Brief     :  Sensor Layer: Definitions of commands and functions for sensor
//                            access.
//==============================================================================

#ifndef SHT3X_H
#define SHT3X_H

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

//-- Enumerations --------------------------------------------------------------
// Sensor Commands
typedef enum{
  CMD_READ_SERIALNBR  = 0x3780, // read serial number
  CMD_READ_STATUS     = 0xF32D, // read status register
  CMD_CLEAR_STATUS    = 0x3041, // clear status register
  CMD_HEATER_ENABLE   = 0x306D, // enabled heater
  CMD_HEATER_DISABLE  = 0x3066, // disable heater
  CMD_SOFT_RESET      = 0x30A2, // soft reset
  CMD_MEAS_CLOCKSTR_H = 0x2C06, // measurement: clock stretching, high repeatability
  CMD_MEAS_CLOCKSTR_M = 0x2C0D, // measurement: clock stretching, medium
repeatability
  CMD_MEAS_CLOCKSTR_L = 0x2C10, // measurement: clock stretching, low repeatability
  CMD_MEAS_POLLING_H  = 0x2400, // measurement: polling, high repeatability
  CMD_MEAS_POLLING_M  = 0x240B, // measurement: polling, medium repeatability
  CMD_MEAS_POLLING_L  = 0x2416, // measurement: polling, low repeatability
  CMD_MEAS_PERI_05_H  = 0x2032, // measurement: periodic 0.5 mps, high repeatability
  CMD_MEAS_PERI_05_M  = 0x2024, // measurement: periodic 0.5 mps, medium
repeatability
  CMD_MEAS_PERI_05_L  = 0x202F, // measurement: periodic 0.5 mps, low repeatability
  CMD_MEAS_PERI_1_H   = 0x2130, // measurement: periodic 1 mps, high repeatability
  CMD_MEAS_PERI_1_M   = 0x2126, // measurement: periodic 1 mps, medium repeatability
  CMD_MEAS_PERI_1_L   = 0x212D, // measurement: periodic 1 mps, low repeatability
  CMD_MEAS_PERI_2_H   = 0x2236, // measurement: periodic 2 mps, high repeatability
  CMD_MEAS_PERI_2_M   = 0x2220, // measurement: periodic 2 mps, medium repeatability
  CMD_MEAS_PERI_2_L   = 0x222B, // measurement: periodic 2 mps, low repeatability
  CMD_MEAS_PERI_4_H   = 0x2334, // measurement: periodic 4 mps, high repeatability
  CMD_MEAS_PERI_4_M   = 0x2322, // measurement: periodic 4 mps, medium repeatability
  CMD_MEAS_PERI_4_L   = 0x2329, // measurement: periodic 4 mps, low repeatability
  CMD_MEAS_PERI_10_H  = 0x2737, // measurement: periodic 10 mps, high repeatability
  CMD_MEAS_PERI_10_M  = 0x2721, // measurement: periodic 10 mps, medium
repeatability
  CMD_MEAS_PERI_10_L  = 0x272A, // measurement: periodic 10 mps, low repeatability
  CMD_FETCH_DATA      = 0xE000, // readout measurements for periodic mode
  CMD_R_AL_LIM_LS     = 0xE102, // read alert limits, low set
  CMD_R_AL_LIM_LC     = 0xE109, // read alert limits, low clear
  CMD_R_AL_LIM_HS     = 0xE11F, // read alert limits, high set
  CMD_R_AL_LIM_HC     = 0xE114, // read alert limits, high clear
  CMD_W_AL_LIM_HS     = 0x611D, // write alert limits, high set
  CMD_W_AL_LIM_HC     = 0x6116, // write alert limits, high clear
  CMD_W_AL_LIM_LC      = 0x610B, // write alert limits, low clear
  CMD_W_AL_LIM_LS      = 0x6100, // write alert limits, low set
  CMD_NO_SLEEP         = 0x303E,
}etCommands;

// Measurement Repeatability
typedef enum{
  REPEATAB_HIGH,   // high repeatability
  REPEATAB_MEDIUM, // medium repeatability
  REPEATAB_LOW,    // low repeatability
}etRepeatability;

// Measurement Mode
typedef enum{
  MODE_CLKSTRETCH, // clock stretching
  MODE_POLLING,    // polling
}etMode;

typedef enum{
  FREQUENCY_HZ5,  //  0.5 measurements per seconds
  FREQUENCY_1HZ,  //  1.0 measurements per seconds
  FREQUENCY_2HZ,  //  2.0 measurements per seconds
  FREQUENCY_4HZ,  //  4.0 measurements per seconds
  FREQUENCY_10HZ, // 10.0 measurements per seconds
}etFrequency;

//-- Typedefs -----------------------------------------------------------------
// Status-Register
typedef union {
  u16t u16;
  struct{
    #ifdef LITTLE_ENDIAN  // bit-order is little endian
    u16t CrcStatus     : 1; // write data checksum status
    u16t CmdStatus     : 1; // command status
    u16t Reserve0      : 2; // reserved
    u16t ResetDetected : 1; // system reset detected
    u16t Reserve1      : 5; // reserved
    u16t T_Alert       : 1; // temperature tracking alert
    u16t RH_Alert      : 1; // humidity tracking alert
    u16t Reserve2      : 1; // reserved
    u16t HeaterStatus  : 1; // heater status
    u16t Reserve3      : 1; // reserved
    u16t AlertPending  : 1; // alert pending status
    #else                 // bit-order is big endian
    u16t AlertPending  : 1;
    u16t Reserve3      : 1;
    u16t HeaterStatus  : 1;
    u16t Reserve2      : 1;
    u16t RH_Alert      : 1;
    u16t T_Alert       : 1;
    u16t Reserve1      : 5;
    u16t ResetDetected : 1;
    u16t Reserve0      : 2;
    u16t CmdStatus     : 1;
    u16t CrcStatus     : 1;
    #endif
  }bit;
} regStatus;


//=============================================================================
// Initializes the I2C bus for communication with the sensor.
```

```
//-----------------------------------------------------------------------------
// input: i2cAddress    I2C address, 0x44 ADDR pin low / 0x45 ADDR pin high
//-----------------------------------------------------------------------------
void SHT3X_Init(u8t i2cAddress);


//=============================================================================
// Sets the I2C address.
//-----------------------------------------------------------------------------
// input: i2cAddress    I2C address, 0x44 ADDR pin low / 0x45 ADDR pin high
//-----------------------------------------------------------------------------
void SHT3X_SetI2cAdr(u8t i2cAddress);


//=============================================================================
// Reads the serial number from sensor.
//-----------------------------------------------------------------------------
// input: serialNumber  pointer to serialNumber
//
// return: error:       ACK_ERROR      = no acknowledgment from sensor
//                      CHECKSUM_ERROR = checksum mismatch
//                      TIMEOUT_ERROR  = timeout
//                      NO_ERROR       = no error
//-----------------------------------------------------------------------------
etError SHT3x_ReadSerialNumber(u32t* serialNumber);


//=============================================================================
// Reads the status register from the sensor.
//-----------------------------------------------------------------------------
// input: status        pointer to status
//
// return: error:       ACK_ERROR      = no acknowledgment from sensor
//                      CHECKSUM_ERROR = checksum mismatch
//                      TIMEOUT_ERROR  = timeout
//                      NO_ERROR       = no error
//-----------------------------------------------------------------------------
etError SHT3X_ReadStatus(u16t* status);


//=============================================================================
// Clears all alert flags in status register from sensor.
//-----------------------------------------------------------------------------
// return: error:       ACK_ERROR      = no acknowledgment from sensor
//                      CHECKSUM_ERROR = checksum mismatch
//                      TIMEOUT_ERROR  = timeout
//                      NO_ERROR       = no error
//-----------------------------------------------------------------------------
etError SHT3X_ClearAllAlertFlags(void);


//=============================================================================
// Gets the temperature [°C] and the relative humidity [%RH] from the sensor.
//-----------------------------------------------------------------------------
// input: temperature   pointer to temperature
//        humiditiy     pointer to humidity
//        repeatability repeatability for the measurement [low, medium, high]
//        mode          command mode [clock stretching, polling]
//        timeout       timeout in milliseconds
//
// return: error:       ACK_ERROR      = no acknowledgment from sensor
//                      CHECKSUM_ERROR = checksum mismatch
//                        TIMEOUT_ERROR  = timeout
//                        PARM_ERROR     = parameter out of range
//                        NO_ERROR       = no error
//-----------------------------------------------------------------------------
etError SHT3X_GetTempAndHumi(ft* temperature, ft* humiditiy,
                             etRepeatability repeatability, etMode mode,
                             u8t timeout);


//=============================================================================
// Gets the temperature [°C] and the relative humidity [%RH] from the sensor.
// This function uses the i2c clock stretching for waiting until measurement is
// ready.
//-----------------------------------------------------------------------------
// input: temperature   pointer to temperature
//        humiditiy     pointer to humidity
//        repeatability repeatability for the measurement [low, medium, high]
//        timeout       clock stretching timeout in milliseconds
//
// return: error:        ACK_ERROR      = no acknowledgment from sensor
//                        CHECKSUM_ERROR = checksum mismatch
//                        TIMEOUT_ERROR  = timeout
//                        PARM_ERROR     = parameter out of range
//                        NO_ERROR       = no error
//-----------------------------------------------------------------------------
etError SHT3X_GetTempAndHumiClkStretch(ft* temperature, ft* humiditiy,
                                       etRepeatability repeatability,
                                       u8t timeout);


//=============================================================================
// Gets the temperature [°C] and the relative humidity [%RH] from the sensor.
// This function polls every 1ms until measurement is ready.
//-----------------------------------------------------------------------------
// input: temperature   pointer to temperature
//        humiditiy     pointer to humidity
//        repeatability repeatability for the measurement [low, medium, high]
//        timeout       polling timeout in milliseconds
//
// return: error:        ACK_ERROR      = no acknowledgment from sensor
//                        CHECKSUM_ERROR = checksum mismatch
//                        TIMEOUT_ERROR  = timeout
//                        PARM_ERROR     = parameter out of range
//                        NO_ERROR       = no error
//-----------------------------------------------------------------------------
etError SHT3X_GetTempAndHumiPolling(ft* temperature, ft* humiditiy,
                                    etRepeatability repeatability,
                                    u8t timeout);


//=============================================================================
// Starts periodic measurement.
//-----------------------------------------------------------------------------
// input: repeatability repeatability for the measurement [low, medium, high]
//        frequency     measurement frequency [0.5, 1, 2, 4, 10] Hz
//
// return: error:        ACK_ERROR      = no acknowledgment from sensor
//                        CHECKSUM_ERROR = checksum mismatch
//                        TIMEOUT_ERROR  = timeout
//                        PARM_ERROR     = parameter out of range
//                        NO_ERROR       = no error
//-----------------------------------------------------------------------------
```

```
etError SHT3X_StartPeriodicMeasurment(etRepeatability repeatability,
                                      etFrequency frequency);


//=============================================================================
// Reads last measurement from the sensor buffer
//-----------------------------------------------------------------------------
// input: temperature   pointer to temperature
//        humidity      pointer to humidity
//
// return: error:        ACK_ERROR      = no acknowledgment from sensor
//                       CHECKSUM_ERROR = checksum mismatch
//                       TIMEOUT_ERROR  = timeout
//                       NO_ERROR       = no error
//-----------------------------------------------------------------------------
etError SHT3X_ReadMeasurementBuffer(ft* temperature, ft* humidity);


//=============================================================================
// Enables the heater on sensor
//-----------------------------------------------------------------------------
// return: error:        ACK_ERROR      = no acknowledgment from sensor
//                       CHECKSUM_ERROR = checksum mismatch
//                       TIMEOUT_ERROR  = timeout
//                       NO_ERROR       = no error
//-----------------------------------------------------------------------------
etError SHT3X_EnableHeater(void);


//=============================================================================
// Disables the heater on sensor
//-----------------------------------------------------------------------------
// return: error:        ACK_ERROR      = no acknowledgment from sensor
//                       CHECKSUM_ERROR = checksum mismatch
//                       TIMEOUT_ERROR  = timeout
//                       NO_ERROR       = no error
//-----------------------------------------------------------------------------
etError SHT3X_DisableHeater(void);

//=============================================================================
//
//-----------------------------------------------------------------------------
etError SHT3X_SetAlertLimits(ft humidityHighSet,   ft temperatureHighSet,
                             ft humidityHighClear, ft temperatureHighClear,
                             ft humidityLowClear,  ft temperatureLowClear,
                             ft humidityLowSet,    ft temperatureLowSet);

//=============================================================================
//
//-----------------------------------------------------------------------------
etError SHT3X_GetAlertLimits(ft* humidityHighSet,   ft* temperatureHighSet,
                             ft* humidityHighClear, ft* temperatureHighClear,
                             ft* humidityLowClear,  ft* temperatureLowClear,
                             ft* humidityLowSet,    ft* temperatureLowSet);

//=============================================================================
// Returns the state of the Alert-Pin.
//-----------------------------------------------------------------------------
// return:               true:  Alert-Pin is high
//                       false: Alter-Pin is low
//-----------------------------------------------------------------------------
bt SHT3X_ReadAlert(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
//                      CHECKSUM_ERROR = checksum mismatch
//                      TIMEOUT_ERROR  = timeout
//                      NO_ERROR       = no error
//------------------------------------------------------------------------------
etError SHT3X_SoftReset(void);


//==============================================================================
// Resets the sensor by pulling down the reset pin.
//------------------------------------------------------------------------------
void SHT3X_HardReset(void);


#endif
```

2.3 sht3x.c

```
//==============================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//==============================================================================
// Project   :  SHT3x Sample Code (V1.1)
// File      :  sht3x.c (V1.1)
// Author    :  RFU
// Date      :  6-Mai-2015
// Controller:  STM32F100RB
// IDE       :  µVision V5.12.0.0
// Compiler  :  Armcc
// Brief     :  Sensor Layer: Implementation of functions for sensor access.
//==============================================================================

//-- Includes ------------------------------------------------------------------
#include "sht3x.h"
#include "i2c_hal.h"

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

//==============================================================================
// IO-Pins                            /* -- adapt the defines for your uC -- */
//------------------------------------------------------------------------------
// Reset on port B, bit 12
#define RESET_LOW()  (GPIOB->BSRR = 0x10000000) // set Reset to low
#define RESET_HIGH() (GPIOB->BSRR = 0x00001000) // set Reset to high

// Alert on port B, bit 10
#define ALERT_READ   (GPIOB->IDR  & 0x0400)     // read Alert
//==============================================================================

//-- Global variables ----------------------------------------------------------
static u8t _i2cAddress; // I2C Address

//-- Static function prototypes ------------------------------------------------
static etError SHT3X_WriteAlertLimitData(ft humidity, ft temperature);
static etError SHT3X_ReadAlertLimitData(ft* humidity, ft* temperature);
static etError SHT3X_StartWriteAccess(void);
static etError SHT3X_StartReadAccess(void);
static void SHT3X_StopAccess(void);
static etError SHT3X_WriteCommand(etCommands command);
static etError SHT3X_Read2BytesAndCrc(u16t* data, etI2cAck finaleAckNack,
                                      u8t timeout);
static etError SHT3X_Write2BytesAndCrc(u16t data);
static u8t SHT3X_CalcCrc(u8t data[], u8t nbrOfBytes);
static etError SHT3X_CheckCrc(u8t data[], u8t nbrOfBytes, u8t checksum);
static ft SHT3X_CalcTemperature(u16t rawValue);
static ft SHT3X_CalcHumidity(u16t rawValue);
static u16t SHT3X_CalcRawTemperature(ft temperature);
static u16t SHT3X_CalcRawHumidity(ft humidity);

//------------------------------------------------------------------------------
void SHT3X_Init(u8t i2cAddress)          /* -- adapt the init for your uC -- */
{
  // init I/O-pins
  RCC->APB2ENR |= 0x00000008;  // I/O port B clock enabled

  // Alert on port B, bit 10
  GPIOB->CRH   &= 0xFFFFF0FF;  // set floating input for Alert-Pin
  GPIOB->CRH   |= 0x00000400;  //
  // Reset on port B, bit 12
  GPIOB->CRH   &= 0xFFF0FFFF;  // set push-pull output for Reset pin
  GPIOB->CRH   |= 0x00010000;  //
  RESET_LOW();

  I2c_Init(); // init I2C
  SHT3X_SetI2cAdr(i2cAddress);

  // release reset
  RESET_HIGH();
}

//-----------------------------------------------------------------------------
void SHT3X_SetI2cAdr(u8t i2cAddress)
{
  _i2cAddress = i2cAddress;
}

//-----------------------------------------------------------------------------
etError SHT3x_ReadSerialNumber(u32t* serialNumber)
{
  etError error; // error code
  u16t serialNumWords[2];

  error = SHT3X_StartWriteAccess();

  // write "read serial number" command
  error |= SHT3X_WriteCommand(CMD_READ_SERIALNBR);
  // if no error, start read access
  if(error == NO_ERROR) error = SHT3X_StartReadAccess();
  // if no error, read first serial number word
  if(error == NO_ERROR) error = SHT3X_Read2BytesAndCrc(&serialNumWords[0], ACK,
100);
  // if no error, read second serial number word
  if(error == NO_ERROR) error = SHT3X_Read2BytesAndCrc(&serialNumWords[1], NACK, 0);

  SHT3X_StopAccess();

  // if no error, calc serial number as 32-bit integer
  if(error == NO_ERROR)
  {
    *serialNumber = (serialNumWords[0] << 16) | serialNumWords[1];
  }

  return error;
}

//-----------------------------------------------------------------------------
etError SHT3X_ReadStatus(u16t* status)
{
  etError error; // error code

  error = SHT3X_StartWriteAccess();

  // if no error, write "read status" command
  if(error == NO_ERROR) error = SHT3X_WriteCommand(CMD_READ_STATUS);
  // if no error, start read access
  if(error == NO_ERROR) error = SHT3X_StartReadAccess();
  // if no error, read status
  if(error == NO_ERROR) error = SHT3X_Read2BytesAndCrc(status, NACK, 0);
  SHT3X_StopAccess();

  return error;
}

//-----------------------------------------------------------------------------
etError SHT3X_ClearAllAlertFlags(void)
{
  etError error; // error code

  error = SHT3X_StartWriteAccess();

  // if no error, write clear status register command
  if(error == NO_ERROR) error = SHT3X_WriteCommand(CMD_CLEAR_STATUS);

  SHT3X_StopAccess();

  return error;
}

//-----------------------------------------------------------------------------
etError SHT3X_GetTempAndHumi(ft* temperature, ft* humidity,
                             etRepeatability repeatability, etMode mode,
                             u8t timeout)
{
  etError error;

  switch(mode)
  {
    case MODE_CLKSTRETCH: // get temperature with clock stretching mode
      error = SHT3X_GetTempAndHumiClkStretch(temperature, humidity,
                                             repeatability, timeout);
      break;
    case MODE_POLLING:    // get temperature with polling mode
      error = SHT3X_GetTempAndHumiPolling(temperature, humidity,
                                          repeatability, timeout);
      break;
    default:
      error = PARM_ERROR;
      break;
  }

  return error;
}


//-----------------------------------------------------------------------------
etError SHT3X_GetTempAndHumiClkStretch(ft* temperature, ft* humidity,
                                       etRepeatability repeatability,
                                       u8t timeout)
{
  etError error;        // error code
  u16t    rawValueTemp; // temperature raw value from sensor
  u16t    rawValueHumi; // humidity raw value from sensor

  error = SHT3X_StartWriteAccess();

  // if no error ...
  if(error == NO_ERROR)
  {
    // start measurement in clock stretching mode
    // use depending on the required repeatability, the corresponding command
    switch(repeatability)
    {
      case REPEATAB_LOW:
        error = SHT3X_WriteCommand(CMD_MEAS_CLOCKSTR_L);
        break;
      case REPEATAB_MEDIUM:
        error = SHT3X_WriteCommand(CMD_MEAS_CLOCKSTR_M);
        break;
      case REPEATAB_HIGH:
        error = SHT3X_WriteCommand(CMD_MEAS_CLOCKSTR_H);
        break;
      default:
        error = PARM_ERROR;
        break;
    }
  }

  // if no error, start read access
  if(error == NO_ERROR) error = SHT3X_StartReadAccess();
  // if no error, read temperature raw values
  if(error == NO_ERROR) error = SHT3X_Read2BytesAndCrc(&rawValueTemp, ACK, timeout);
  // if no error, read humidity raw values
  if(error == NO_ERROR) error = SHT3X_Read2BytesAndCrc(&rawValueHumi, NACK, 0);

  SHT3X_StopAccess();

  // if no error, calculate temperature in °C and humidity in %RH
  if(error == NO_ERROR)
  {
    *temperature = SHT3X_CalcTemperature(rawValueTemp);
    *humidity = SHT3X_CalcHumidity(rawValueHumi);
  }

  return error;
}

//-----------------------------------------------------------------------------
etError SHT3X_GetTempAndHumiPolling(ft* temperature, ft* humidity,
                                    etRepeatability repeatability,
                                    u8t timeout)
{
  etError error;           // error code
  u16t    rawValueTemp;    // temperature raw value from sensor
  u16t    rawValueHumi;    // humidity raw value from sensor

  error  = SHT3X_StartWriteAccess();

  // if no error ...
  if(error == NO_ERROR)
  {
    // start measurement in polling mode
    // use depending on the required repeatability, the corresponding command
    switch(repeatability)
    {
      case REPEATAB_LOW:
        error = SHT3X_WriteCommand(CMD_MEAS_POLLING_L);
        break;
      case REPEATAB_MEDIUM:
        error = SHT3X_WriteCommand(CMD_MEAS_POLLING_M);
        break;
      case REPEATAB_HIGH:
        error = SHT3X_WriteCommand(CMD_MEAS_POLLING_H);
        break;
      default:
       error = PARM_ERROR;
        break;
    }
  }

  // if no error, wait until measurement ready
  if(error == NO_ERROR)
  {
    // poll every 1ms for measurement ready until timeout
    while(timeout--)
    {
      // check if the measurement has finished
      error = SHT3X_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 |= SHT3X_Read2BytesAndCrc(&rawValueTemp, ACK, 0);
    error |= SHT3X_Read2BytesAndCrc(&rawValueHumi, NACK, 0);
  }

  SHT3X_StopAccess();

  // if no error, calculate temperature in °C and humidity in %RH
  if(error == NO_ERROR)
  {
    *temperature = SHT3X_CalcTemperature(rawValueTemp);
    *humidity = SHT3X_CalcHumidity(rawValueHumi);
  }

  return error;
}

//-----------------------------------------------------------------------------
etError SHT3X_StartPeriodicMeasurment(etRepeatability repeatability,
                                      etFrequency frequency)
{
  etError error;        // error code

  error = SHT3X_StartWriteAccess();

  // if no error, start periodic measurement
  if(error == NO_ERROR)
  {
    // use depending on the required repeatability and frequency,
    // the corresponding command
    switch(repeatability)
    {
      case REPEATAB_LOW: // low repeatability
        switch(frequency)
        {
          case FREQUENCY_HZ5:  // low repeatability,  0.5 Hz
            error |= SHT3X_WriteCommand(CMD_MEAS_PERI_05_L);
            break;
          case FREQUENCY_1HZ:  // low repeatability,  1.0 Hz
            error |= SHT3X_WriteCommand(CMD_MEAS_PERI_1_L);
            break;
          case FREQUENCY_2HZ:  // low repeatability,  2.0 Hz
            error |= SHT3X_WriteCommand(CMD_MEAS_PERI_2_L);
            break;
          case FREQUENCY_4HZ:  // low repeatability,  4.0 Hz
            error |= SHT3X_WriteCommand(CMD_MEAS_PERI_4_L);
            break;
          case FREQUENCY_10HZ: // low repeatability, 10.0 Hz
            error |= SHT3X_WriteCommand(CMD_MEAS_PERI_10_L);
            break;
          default:
            error |= PARM_ERROR;
            break;
        }
        break;

      case REPEATAB_MEDIUM: // medium repeatability
        switch(frequency)
        {
          case FREQUENCY_HZ5:  // medium repeatability,  0.5 Hz
            error |= SHT3X_WriteCommand(CMD_MEAS_PERI_05_M);
                  break;
          case FREQUENCY_1HZ:  // medium repeatability,  1.0 Hz
            error |= SHT3X_WriteCommand(CMD_MEAS_PERI_1_M);
                  break;
          case FREQUENCY_2HZ:  // medium repeatability,  2.0 Hz
            error |= SHT3X_WriteCommand(CMD_MEAS_PERI_2_M);
                  break;
          case FREQUENCY_4HZ:  // medium repeatability,  4.0 Hz
            error |= SHT3X_WriteCommand(CMD_MEAS_PERI_4_M);
                  break;
          case FREQUENCY_10HZ: // medium repeatability, 10.0 Hz
            error |= SHT3X_WriteCommand(CMD_MEAS_PERI_10_M);
                  break;
          default:
            error |= PARM_ERROR;
                  break;
        }
        break;

      case REPEATAB_HIGH: // high repeatability
        switch(frequency)
        {
          case FREQUENCY_HZ5:  // high repeatability,  0.5 Hz
            error |= SHT3X_WriteCommand(CMD_MEAS_PERI_05_H);
            break;
          case FREQUENCY_1HZ:  // high repeatability,  1.0 Hz
            error |= SHT3X_WriteCommand(CMD_MEAS_PERI_1_H);
            break;
          case FREQUENCY_2HZ:  // high repeatability,  2.0 Hz
            error |= SHT3X_WriteCommand(CMD_MEAS_PERI_2_H);
            break;
          case FREQUENCY_4HZ:  // high repeatability,  4.0 Hz
            error |= SHT3X_WriteCommand(CMD_MEAS_PERI_4_H);
            break;
          case FREQUENCY_10HZ: // high repeatability, 10.0 Hz
            error |= SHT3X_WriteCommand(CMD_MEAS_PERI_10_H);
            break;
          default:
            error |= PARM_ERROR;
            break;
        }
        break;
      default:
        error |= PARM_ERROR;
        break;
    }
  }

  SHT3X_StopAccess();

  return error;
}

//-----------------------------------------------------------------------------
etError SHT3X_ReadMeasurementBuffer(ft* temperature, ft* humidity)
{
  etError  error;        // error code
  u16t     rawValueTemp; // temperature raw value from sensor
  u16t     rawValueHumi; // humidity raw value from sensor

  error = SHT3X_StartWriteAccess();

  // if no error, read measurements
  if(error == NO_ERROR) error = SHT3X_WriteCommand(CMD_FETCH_DATA);
  if(error == NO_ERROR) error = SHT3X_StartReadAccess();
  if(error == NO_ERROR) error = SHT3X_Read2BytesAndCrc(&rawValueTemp, ACK, 0);
  if(error == NO_ERROR) error = SHT3X_Read2BytesAndCrc(&rawValueHumi, NACK, 0);

  // if no error, calculate temperature in °C and humidity in %RH
  if(error == NO_ERROR)
  {
    *temperature = SHT3X_CalcTemperature(rawValueTemp);
    *humidity = SHT3X_CalcHumidity(rawValueHumi);
  }

  SHT3X_StopAccess();

  return error;
}

//-----------------------------------------------------------------------------
etError SHT3X_EnableHeater(void)
{
  etError error; // error code

  error = SHT3X_StartWriteAccess();

  // if no error, write heater enable command
  if(error == NO_ERROR) error = SHT3X_WriteCommand(CMD_HEATER_ENABLE);

  SHT3X_StopAccess();

  return error;
}

//-----------------------------------------------------------------------------
etError SHT3X_DisableHeater(void)
{
  etError error; // error code
  error = SHT3X_StartWriteAccess();

  // if no error, write heater disable command
  if(error == NO_ERROR) error = SHT3X_WriteCommand(CMD_HEATER_DISABLE);

  SHT3X_StopAccess();

  return error;
}


//-----------------------------------------------------------------------------
etError SHT3X_SetAlertLimits(ft humidityHighSet,   ft temperatureHighSet,
                             ft humidityHighClear, ft temperatureHighClear,
                             ft humidityLowClear,  ft temperatureLowClear,
                             ft humidityLowSet,    ft temperatureLowSet)
{
  etError  error;  // error code

  // write humidity & temperature alter limits, high set
  error = SHT3X_StartWriteAccess();
  if(error == NO_ERROR) error = SHT3X_WriteCommand(CMD_W_AL_LIM_HS);
  if(error == NO_ERROR) error = SHT3X_WriteAlertLimitData(humidityHighSet,
                                                          temperatureHighSet);
  SHT3X_StopAccess();

  if(error == NO_ERROR)
  {
    // write humidity & temperature alter limits, high clear
    error = SHT3X_StartWriteAccess();
    if(error == NO_ERROR) error = SHT3X_WriteCommand(CMD_W_AL_LIM_HC);
    if(error == NO_ERROR) error = SHT3X_WriteAlertLimitData(humidityHighClear,
                                                            temperatureHighClear);
    SHT3X_StopAccess();
  }

  if(error == NO_ERROR)
  {
    // write humidity & temperature alter limits, low clear
    error = SHT3X_StartWriteAccess();
    if(error == NO_ERROR) error = SHT3X_WriteCommand(CMD_W_AL_LIM_LC);
    if(error == NO_ERROR) error = SHT3X_WriteAlertLimitData(humidityLowClear,
                                                            temperatureLowClear);
    SHT3X_StopAccess();
  }

  if(error == NO_ERROR)
  {
    // write humidity & temperature alter limits, low set
    error = SHT3X_StartWriteAccess();
    if(error == NO_ERROR) error = SHT3X_WriteCommand(CMD_W_AL_LIM_LS);
    if(error == NO_ERROR) error = SHT3X_WriteAlertLimitData(humidityLowSet,
                                                            temperatureLowSet);
    SHT3X_StopAccess();
  }

  return error;
}

//-----------------------------------------------------------------------------
etError SHT3X_GetAlertLimits(ft* humidityHighSet,   ft* temperatureHighSet,
                             ft* humidityHighClear, ft* temperatureHighClear,
                              ft* humidityLowClear,  ft* temperatureLowClear,
                              ft* humidityLowSet,    ft* temperatureLowSet)
{
  etError  error;  // error code

  // read humidity & temperature alter limits, high set
  error = SHT3X_StartWriteAccess();
  if(error == NO_ERROR) error = SHT3X_WriteCommand(CMD_R_AL_LIM_HS);
  if(error == NO_ERROR) error = SHT3X_StartReadAccess();
  if(error == NO_ERROR) error = SHT3X_ReadAlertLimitData(humidityHighSet,
                                                         temperatureHighSet);
  SHT3X_StopAccess();

  if(error == NO_ERROR)
  {
    // read humidity & temperature alter limits, high clear
    error = SHT3X_StartWriteAccess();
    if(error == NO_ERROR) error = SHT3X_WriteCommand(CMD_R_AL_LIM_HC);
    if(error == NO_ERROR) error = SHT3X_StartReadAccess();
    if(error == NO_ERROR) error = SHT3X_ReadAlertLimitData(humidityHighClear,
                                                           temperatureHighClear);
    SHT3X_StopAccess();
  }

  if(error == NO_ERROR)
  {
    // read humidity & temperature alter limits, low clear
    error = SHT3X_StartWriteAccess();
    if(error == NO_ERROR) error = SHT3X_WriteCommand(CMD_R_AL_LIM_LC);
    if(error == NO_ERROR) error = SHT3X_StartReadAccess();
    if(error == NO_ERROR) error = SHT3X_ReadAlertLimitData(humidityLowClear,
                                                           temperatureLowClear);
    SHT3X_StopAccess();
  }

  if(error == NO_ERROR)
  {
    // read humidity & temperature alter limits, low set
    error = SHT3X_StartWriteAccess();
    if(error == NO_ERROR) error = SHT3X_WriteCommand(CMD_R_AL_LIM_LS);
    if(error == NO_ERROR) error = SHT3X_StartReadAccess();
    if(error == NO_ERROR) error = SHT3X_ReadAlertLimitData(humidityLowSet,
                                                           temperatureLowSet);
    SHT3X_StopAccess();
  }

  return error;
}

//-----------------------------------------------------------------------------
bt SHT3X_ReadAlert(void)
{
  // read alert pin
  return (ALERT_READ != 0) ? TRUE : FALSE;
}

//-----------------------------------------------------------------------------
etError SHT3X_SoftReset(void)
{
  etError error; // error code

  error = SHT3X_StartWriteAccess();
  // write reset command
  error |= SHT3X_WriteCommand(CMD_SOFT_RESET);

  SHT3X_StopAccess();

  // if no error, wait 50 ms after reset
  if(error == NO_ERROR) DelayMicroSeconds(50000);

  return error;
}

//-----------------------------------------------------------------------------
void SHT3X_HardReset(void)
{
  // set reset low
  RESET_LOW();

  // wait 100 ms
  DelayMicroSeconds(100000);

  // release reset
  RESET_HIGH();

  // wait 50 ms after reset
  DelayMicroSeconds(50000);
}


//-----------------------------------------------------------------------------
static etError SHT3X_WriteAlertLimitData(ft humidity, ft temperature)
{
  etError  error;           // error code

  i16t rawHumidity;
  i16t rawTemperature;

  if((humidity < 0.0f) || (humidity > 100.0f)
  || (temperature < -45.0f) || (temperature > 130.0f))
  {
    error = PARM_ERROR;
  }
  else
  {
    rawHumidity    = SHT3X_CalcRawHumidity(humidity);
    rawTemperature = SHT3X_CalcRawTemperature(temperature);

    error = SHT3X_Write2BytesAndCrc((rawHumidity & 0xFE00) | ((rawTemperature >> 7)
& 0x001FF));
  }

  return error;
}

//-----------------------------------------------------------------------------
static etError SHT3X_ReadAlertLimitData(ft* humidity, ft* temperature)
{
  etError  error;           // error code
  u16t     data;

  error = SHT3X_Read2BytesAndCrc(&data, NACK, 0);
  if(error == NO_ERROR)
  {
    *humidity = SHT3X_CalcHumidity(data & 0xFE00);
    *temperature = SHT3X_CalcTemperature(data << 7);
  }

  return error;
}

//-----------------------------------------------------------------------------
static etError SHT3X_StartWriteAccess(void)
{
  etError error; // error code

  // write a start condition
  I2c_StartCondition();

  // write the sensor I2C address with the write flag
  error = I2c_WriteByte(_i2cAddress << 1);

  return error;
}

//-----------------------------------------------------------------------------
static etError SHT3X_StartReadAccess(void)
{
  etError error; // error code

  // write a start condition
  I2c_StartCondition();

  // write the sensor I2C address with the read flag
  error = I2c_WriteByte(_i2cAddress << 1 | 0x01);

  return error;
}

//-----------------------------------------------------------------------------
static void SHT3X_StopAccess(void)
{
  // write a stop condition
  I2c_StopCondition();
}

//-----------------------------------------------------------------------------
static etError SHT3X_WriteCommand(etCommands command)
{
  etError error; // error code

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

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

  return error;
}

//-----------------------------------------------------------------------------
static etError SHT3X_Read2BytesAndCrc(u16t* data, etI2cAck finaleAckNack,
                                      u8t timeout)
{
  etError error;    // error code
  u8t     bytes[2]; // read data array
  u8t     checksum; // checksum byte
 
  // read two data bytes and one checksum byte
                        error = I2c_ReadByte(&bytes[0], ACK, timeout);
  if(error == NO_ERROR) error = I2c_ReadByte(&bytes[1], ACK, 0);
  if(error == NO_ERROR) error = I2c_ReadByte(&checksum, finaleAckNack, 0);
 
  // verify checksum
  if(error == NO_ERROR) error = SHT3X_CheckCrc(bytes, 2, checksum);
 
  // combine the two bytes to a 16-bit value
  *data = (bytes[0] << 8) | bytes[1];
 
  return error;
}

//-----------------------------------------------------------------------------
static etError SHT3X_Write2BytesAndCrc(u16t data)
{
  etError error;    // error code
  u8t     bytes[2]; // read data array
  u8t     checksum; // checksum byte
 
  bytes[0] = data >> 8;
  bytes[1] = data & 0xFF;
  checksum = SHT3X_CalcCrc(bytes, 2);
 
  // write two data bytes and one checksum byte
                        error = I2c_WriteByte(bytes[0]); // write data MSB
  if(error == NO_ERROR) error = I2c_WriteByte(bytes[1]); // write data LSB
  if(error == NO_ERROR) error = I2c_WriteByte(checksum); // write checksum
 
  return error;
}

//-----------------------------------------------------------------------------
static u8t SHT3X_CalcCrc(u8t data[], u8t nbrOfBytes)
{
  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);
    }
  }
 
  return crc;
}

//-----------------------------------------------------------------------------
static etError SHT3X_CheckCrc(u8t data[], u8t nbrOfBytes, u8t checksum)
{
  u8t crc;     // calculated checksum
  // calculates 8-Bit checksum
  crc = SHT3X_CalcCrc(data, nbrOfBytes);

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

//-----------------------------------------------------------------------------
static ft SHT3X_CalcTemperature(u16t rawValue)
{
  // calculate temperature [°C]
  // T = -45 + 175 * rawValue / (2^16-1)
  return 175.0f * (ft)rawValue / 65535.0f - 45.0f;
}

//-----------------------------------------------------------------------------
static ft SHT3X_CalcHumidity(u16t rawValue)
{
  // calculate relative humidity [%RH]
  // RH = rawValue / (2^16-1) * 100
  return 100.0f * (ft)rawValue / 65535.0f;
}

//-----------------------------------------------------------------------------
static u16t SHT3X_CalcRawTemperature(ft temperature)
{
  // calculate raw temperature [ticks]
  // rawT = (temperature + 45) / 175 * (2^16-1)
  return (temperature + 45.0f) / 175.0f * 65535.0f;
}

//-----------------------------------------------------------------------------
static u16t SHT3X_CalcRawHumidity(ft humidity)
{
  // calculate raw relative humidity [ticks]
  // rawRH = humidity / 100 * (2^16-1)
  return humidity / 100.0f * 65535.0f;
}
```

2.4 i2c_hal.h

```
//=============================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//=============================================================================
// Project   :  SHT3x Sample Code (V1.1)
// File      :  i2c_hal.h (V1.1)
// Author    :  RFU
// Date      :  6-Mai-2015
// Controller:  STM32F100RB
// IDE       :  µVision V5.12.0.0
// Compiler  :  Armcc
// Brief     :  I2C hardware abstraction layer
//=============================================================================

#ifndef I2C_HAL_H
#define I2C_HAL_H

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

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

// I2C acknowledge
typedef enum{
  ACK  = 0,
  NACK = 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.

//==============================================================================
etError I2c_ReadByte(u8t *rxByte, etI2cAck ack, u8t timeout);

etError I2c_GeneralCallReset(void);

#endif
```

2.5 i2c_hal.c

```
//==============================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//==============================================================================
// Project   :  SHT3x Sample Code (V1.1)
// File      :  i2c_hal.c (V1.1)
// Author    :  RFU
// Date      :  6-Mai-2015
// Controller:  STM32F100RB
// IDE       :  µVision V5.12.0.0
// Compiler  :  Armcc
// Brief     :  I2C hardware abstraction layer
//==============================================================================

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

//-- Defines -------------------------------------------------------------------
// I2C IO-Pins                         /* -- adapt the defines for your uC -- */

// SDA on port B, bit 14
#define SDA_LOW()  (GPIOB->BSRR = 0x40000000) // set SDA to low
#define SDA_OPEN() (GPIOB->BSRR = 0x00004000) // set SDA to open-drain
#define SDA_READ   (GPIOB->IDR  & 0x4000)     // read SDA

// SCL on port B, bit 13               /* -- adapt the defines for your uC -- */
#define SCL_LOW()  (GPIOB->BSRR = 0x20000000) // set SCL to low
#define SCL_OPEN() (GPIOB->BSRR = 0x00002000) // set SCL to open-drain
#define SCL_READ   (GPIOB->IDR  & 0x2000)     // read SCL

//-- Static function prototypes ------------------------------------------------
static etError I2c_WaitWhileClockStreching(u8t timeout);

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

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

  // SDA on port B, bit 14
  // SCL on port B, bit 13
  GPIOB->CRH   &= 0xF00FFFFF;  // set open-drain output for SDA and SCL
  GPIOB->CRH   |= 0x05500000;  //
}

//------------------------------------------------------------------------------
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)
{
  etError error = NO_ERROR;
  u8t     mask;
  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
}

//-----------------------------------------------------------------------------
etError I2c_ReadByte(u8t *rxByte, etI2cAck ack, u8t timeout)
{
  etError error = NO_ERROR;
  u8t mask;
  *rxByte = 0x00;
  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);                // clock set-up time (t_SU;CLK)
    error = I2c_WaitWhileClockStreching(timeout);// wait while clock streching
    DelayMicroSeconds(3);                // SCL high time (t_HIGH)
    if(SDA_READ) *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 error;                          // return with no error
}

//-----------------------------------------------------------------------------
etError I2c_GeneralCallReset(void)
{
  etError error;

  I2c_StartCondition();
                      error = I2c_WriteByte(0x00);
  if(error == NO_ERROR) error = I2c_WriteByte(0x06);

  return error;
}

//-----------------------------------------------------------------------------
static etError I2c_WaitWhileClockStreching(u8t timeout)
{
  etError error = NO_ERROR;

  while(SCL_READ == 0)
  {
    if(timeout-- == 0) return TIMEOUT_ERROR;
    DelayMicroSeconds(1000);
  }

  return error;
}
```

2.6 system.h

```
//==============================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//==============================================================================
// Project   :  SHT3x Sample Code (V1.1)
// File      :  system.h (V1.1)
// Author    :  RFU
// Date      :  6-Mai-2015
// Controller:  STM32F100RB
// IDE       :  µVision V5.12.0.0
// 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
  TIMEOUT_ERROR  = 0x04, // timeout error
  PARM_ERROR     = 0x80, // parameter out of range 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

```c
//==============================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//==============================================================================
// Project   :  SHT3x Sample Code (V1.1)
// File      :  system.c (V1.1)
// Author    :  RFU
// Date      :  6-Mai-2015
// Controller:  STM32F100RB
// IDE       :  µVision V5.12.0.0
// 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

```c
//==============================================================================
//    S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
//==============================================================================
// Project   :  SHT3x Sample Code (V1.1)
// File      :  typedefs.h (V1.1)
// Author    :  RFU
// Date      :  6-Mai-2015
// Controller:  STM32F100RB
// IDE       :  µVision V5.12.0.0
// 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
August 2014	1	All	Initial release
Mai 2015	2	All	Added alert commands, major structural rework

Headquarters and Subsidiaries

[redacted]