

Application Note for SFM3000

Sample Code

Summary

Sensirion's SFM3000 flow sensor is based on a 5th generation CMOSens[®] mass flow sensor chip called SF05. The sensor can directly be connected to a

microcontroller. This application note contains a C++ sample code to implement the basic commands.

1. Sample Code

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. A copy of

the code may be found on the following pages of this application note.

The source code is available as zip archive on our webpage.

2. Revision history

Date	Version	Changes
November 2012	1.0	Initial release
August 2015	1.1	Minor updates
August 2015	1.2	Minor bug fix in sf05.c & sf05.h

Headquarters and Subsidiaries

SENSIRION AG
Laubisruestr. 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

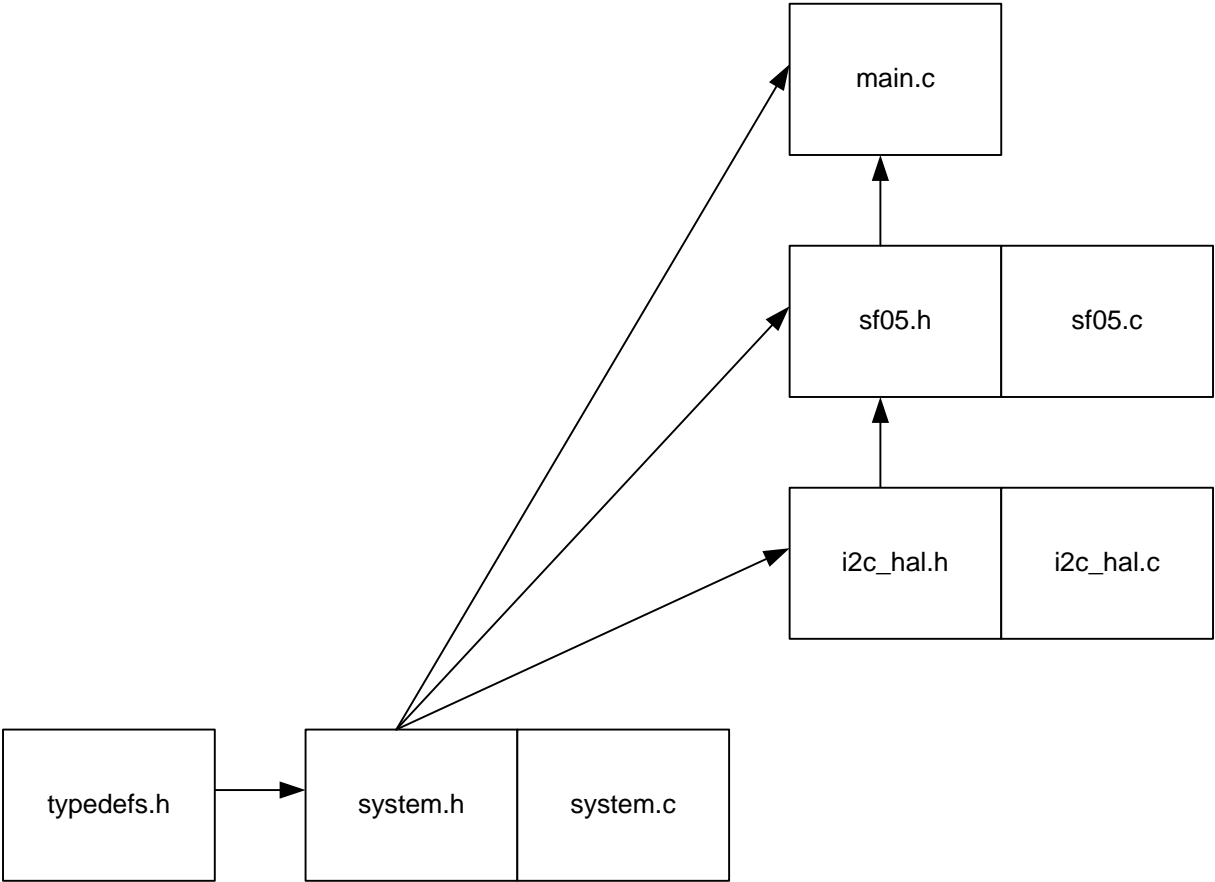
Sensirion Inc., USA
phone: +1 805 409 4900
info_us@sensirion.com
www.sensirion.com

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

Sensirion Korea Co. Ltd.
phone: +82 31 337 7700~3
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



Revision History

Revision	Date	Changes
V1.0	08.11.12 / RFU	Initial draft

```

1  //=====
2  //   S E N S I R I O N   AG, Laubisruetistr. 50, CH-8712 Staefa, Switzerland
3  //=====
4  // Project   : SF05 Sample Code (V1.1)
5  // File      : main.c (V1.1)
6  // Author    : RFU
7  // Date      : 26-Jul-2015
8  // Controller: STM32F100RB
9  // IDE       : µVision V4.60.0.0
10 // Compiler  : Armcc
11 // Brief     : This code shows how to implement the basic commands for a
12 //             flow or differential pressure sensor based on SF05 sensor chip.
13 //             Due to compatibility reasons the I2C interface is implemented
14 //             as "bit-banging" on normal I/O's. This code is written for an
15 //             easy understanding and is neither optimized for speed nor code
16 //             size.
17 //
18 // Porting to a different microcontroller (uC):
19 //   - the definitions of basic types may have to be changed   in typedefs.h
20 //   - change the port functions / definitions for your uC     in i2c_hal.h/.c
21 //   - adapt the timing of the delay function for your uC     in system.c
22 //   - adapt the SystemInit()                                in system.c
23 //   - change the uC register definition file <stm32f10x.h>   in system.h
24 //=====
25
26 //-- Includes -----
27 #include "system.h"
28 #include "sf05.h"
29
30 //-- Defines -----
31 // Offset and scale factors from datasheet (SFM3000).
32 #define OFFSET_FLOW 32000.0F // offset flow
33 #define SCALE_FLOW  140.0F  // scale factor flow
34
35 //=====
36 void Led_Init(void){
37 //=====
38     RCC->APB2ENR |= 0x00000010; // I/O port C clock enabled
39     GPIOC->CRH   &= 0xFFFFF00; // set general purpose output mode for LEDs
40     GPIOC->CRH   |= 0x00000011; //
41     GPIOC->BSRR  = 0x03000000; // LEDs off
42 }
43
44 //=====
45 void UserButton_Init(void){
46 //=====
47     RCC->APB2ENR |= 0x00000004; // I/O port A clock enabled
48     GPIOA->CRH   &= 0xFFFFF00; // set general purpose input mode for User Button
49     GPIOA->CRH   |= 0x00000004; //
50 }
51
52 //=====
53 void LedBlueOn(void){
54 //=====
55     GPIOC->BSRR = 0x00000100;
56 }
57
58 //=====
59 void LedBlueOff(void){
60 //=====
61     GPIOC->BSRR = 0x01000000;
62 }
63
64 //=====
65 void LedGreenOn(void){
66 //=====
67     GPIOC->BSRR = 0x00000200;
68 }
69
70 //=====
71 void LedGreenOff(void){
72 //=====
73     GPIOC->BSRR = 0x02000000;
74 }
75
76 //=====
77 u8t ReadUserButton(void){
78 //=====

```

```
79     return (GPIOA->IDR & 0x00000001);
80 }
81
82 //=====
83 int main(void){
84 //=====
85     etError    error;        // error code
86     u32t        serialNumber; // sensor serial number
87     ft          flow;        // measured flow value
88
89     SystemInit();
90     Led_Init();
91     UserButton_Init();
92     SF05_Init();
93
94     // read serial number from sensor
95     error = SF05_GetSerialNumber(&serialNumber);
96
97     while(1)
98     {
99         error = NO_ERROR;
100
101         if(ReadUserButton() == 0)
102             // if the user button is not pressed
103             {
104                 // read flow from sensor
105                 error = SF05_GetFlow(OFFSET_FLOW, SCALE_FLOW, &flow);
106
107                 // the blue LED lights if a weak flow is detected
108                 if(flow > 1.0) LedBlueOn();
109                 else          LedBlueOff();
110             }
111         else
112             // if the user button is pressed
113             {
114                 // green and blue LED off
115                 LedGreenOff();
116                 LedBlueOff();
117
118                 // perform a soft reset on the sensor
119                 error = SF05_SoftReset();
120
121                 // wait until button is released
122                 while(ReadUserButton() != 0);
123             }
124
125         // the green LED lights if no error occurs
126         if(!error) LedGreenOn();
127         else       LedGreenOff();
128
129         // wait 100ms
130         DelayMicroSeconds(100000);
131     }
132 }
133
```

```

1  //=====
2  //   S E N S I R I O N   AG, Laubisruetistr. 50, CH-8712 Staefa, Switzerland
3  //=====
4  // Project   : SF05 Sample Code (V1.1)
5  // File      : sf05.h (V1.2)
6  // Author    : RFU
7  // Date      : 20-Aug-2015
8  // Controller: STM32F100RB
9  // IDE       : µVision V4.60.0.0
10 // Compiler  : Armcc
11 // Brief     : Sensor Layer: Definitions of commands and functions for sensor
12 //             access.
13 //=====
14
15 #ifndef SF05_H
16 #define SF05_H
17
18 //-- Includes -----
19 #include "system.h"
20
21 //-- Defines -----
22 // CRC
23 #define POLYNOMIAL 0x131 // P(x) = x^8 + x^5 + x^4 + 1 = 100110001
24
25 //-- Enumerations -----
26 // Sensor Commands
27 typedef enum{
28     FLOW_MEASUREMENT      = 0x1000, // command: flow measurement
29     READ_SERIAL_NUMBER_HIGH = 0x31AE, // command: read serial number (bit 31:16)
30     READ_SERIAL_NUMBER_LOW  = 0x31AF, // command: read serial number (bit 15:0)
31     SOFT_RESET              = 0x2000 // command: soft reset
32 }etCommands;
33
34 //=====
35 void SF05_Init(void);
36 //=====
37 // Initializes the I2C bus for communication with the sensor.
38 //-----
39
40 //=====
41 etError SF05_WriteCommand(etCommands cmd);
42 //=====
43 // Writes command to the sensor.
44 //-----
45 // input:  cmd          command which is to be written to the sensor
46 //
47 // return: error:      ACK_ERROR = no acknowledgment from sensor
48 //                   NO_ERROR  = no error
49
50 //=====
51 etError SF05_ReadCommandResult(ul6t *result);
52 //=====
53 // Reads command results from sensor.
54 //-----
55 // input:  *result      pointer to an integer where the result will be stored
56 //
57 // return: error:      ACK_ERROR      = no acknowledgment from sensor
58 //                   CHECKSUM_ERROR = checksum mismatch
59 //                   NO_ERROR       = no error
60
61 //=====
62 etError SF05_ReadCommandResultWithTimeout(u8t maxRetries, ul6t *result);
63 //=====
64 // Reads command results from sensor. If an error occurs, then the read will be
65 // repeated after a short wait (approx. 10ms).
66 //-----
67 // input:  maxRetries    maximum number of retries
68 //         *result       pointer to an integer where the result will be stored
69 //
70 // return: error:      ACK_ERROR      = no acknowledgment from sensor
71 //                   CHECKSUM_ERROR = checksum mismatch
72 //                   NO_ERROR       = no error
73 //
74 // remark: This function is usefull for reading measurement results. If not yet
75 //         a new valid measurement was performed, an acknowledge error occurs
76 //         and the read will be automatical repeated until a valid measurement
77 //         could be read.
78

```

```

79 //=====
80 etError SF05_GetFlow(ft offset, ft scale, ft *flow);
81 //=====
82 // Gets the flow from the sensor in a predefined unit. The "flow measurement"
83 // command will be automatical written to the sensor, if it is not already set.
84 //-----
85 // input:  offset      offset flow
86 //          scale      scale factor flow
87 //          *flow      pointer to a floating point value, where the calculated
88 //                    flow will be stored
89 //
90 // return: error:      ACK_ERROR      = no acknowledgment from sensor
91 //                    CHECKSUM_ERROR = checksum mismatch
92 //                    NO_ERROR       = no error
93 //
94 // remark: The result will be converted according to the following formula:
95 //          flow in predefined unit = (measurement_result - offset) / scale
96
97 //=====
98 etError SF05_GetSerialNumber(u32t *serialNumber);
99 //=====
100 // Gets the serial number from the sensor.
101 //-----
102 // input:  *serialNumber pointer to a 32-bit integer, where the serial number
103 //          will be stored
104 //
105 // return: error:      ACK_ERROR      = no acknowledgment from sensor
106 //                    CHECKSUM_ERROR = checksum mismatch
107 //                    NO_ERROR       = no error
108
109 //=====
110 etError SF05_SoftReset(void);
111 //=====
112 // Forces a sensor reset without switching the power off and on again.
113 //-----
114 // return: error:      ACK_ERROR      = no acknowledgment from sensor
115 //                    NO_ERROR       = no error
116
117 //=====
118 etError SF05_CheckCrc(u8t data[], u8t nbrOfBytes, u8t checksum);
119 //=====
120 // Calculates checksum for n bytes of data and compares it with expected
121 // checksum.
122 //-----
123 // input:  data[]      checksum is built based on this data
124 //          nbrOfBytes  checksum is built for n bytes of data
125 //          checksum    expected checksum
126 //
127 // return: error:      CHECKSUM_ERROR = checksum does not match
128 //                    NO_ERROR       = checksum matches
129
130 #endif
131

```

```

1  //=====
2  //   S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
3  //=====
4  // Project   :   SF05 Sample Code (V1.1)
5  // File      :   sf05.c (V1.2)
6  // Author    :   RFU
7  // Date      :   20-Aug-2015
8  // Controller:   STM32F100RB
9  // IDE       :   uVision V4.60.0.0
10 // Compiler  :   Armcc
11 // Brief     :   Sensor Layer: Implementation of functions for sensor access.
12 //=====
13
14 //-- Includes -----
15 #include "sf05.h"
16 #include "i2c_hal.h"
17
18 //-- Global Variables -----
19 ul6t currentCommand = 0x0000;
20
21 //=====
22 void SF05_Init(void){
23 //=====
24     I2c_Init(); // init I2C
25 }
26
27 //=====
28 etError SF05_WriteCommand(etCommands cmd){
29 //=====
30     etError error; // error code
31
32     // write command to sensor
33     I2c_StartCondition();
34     error = I2c_WriteByte(I2C_ADR << 1 | I2C_WRITE);
35     error |= I2c_WriteByte(cmd >> 8);
36     error |= I2c_WriteByte(cmd & 0xFF);
37     I2c_StopCondition();
38
39     // if no error, store current command
40     if(error == NO_ERROR)
41         currentCommand = cmd;
42
43     return error;
44 }
45
46 //=====
47 etError SF05_ReadCommandResult(ul6t *result){
48 //=====
49     etError error; // error code
50     u8t checksum; // checksum byte
51     u8t data[2]; // read data array
52
53     // read command result & checksum from sensor
54     I2c_StartCondition();
55     error = I2c_WriteByte(I2C_ADR << 1 | I2C_READ);
56     data[0] = I2c_ReadByte(ACK);
57     data[1] = I2c_ReadByte(ACK);
58     checksum = I2c_ReadByte(NO_ACK);
59     I2c_StopCondition();
60
61     // checksum verification
62     error |= SF05_CheckCrc (data, 2, checksum);
63
64     // if no error, combine 16-bit result from the read data array
65     if(error == NO_ERROR)
66         *result = (data[0] << 8) | data[1];
67
68     return error;
69 }
70
71 //=====
72 etError SF05_ReadCommandResultWithTimeout(u8t maxRetries, ul6t *result){
73 //=====
74     etError error; //variable for error code
75
76     while(maxRetries--)
77     {
78         // try to read command result

```

```

79     error = SF05_ReadCommandResult(result);
80
81     // if read command result was successful -> exit loop
82     // it will only be successful if a new valid measurement was performed
83     if(error == NO_ERROR) break;
84
85     // if it was not successful -> wait a short time and then try it again
86     DelayMicroSeconds(10000);
87 }
88
89 return error;
90 }
91
92 //=====
93 etError SF05_GetFlow(ft offset, ft scale, ft *flow){
94 //=====
95     etError error = NO_ERROR; // error code
96     ul6t result; // read result from sensor
97
98     // write command if it is not already set
99     if(currentCommand != FLOW_MEASUREMENT)
100         error = SF05_WriteCommand(FLOW_MEASUREMENT);
101
102     // if no error, read command result
103     if(error == NO_ERROR)
104         error = SF05_ReadCommandResultWithTimeout(20, &result);
105
106     // if no error, compute the flow
107     if(error == NO_ERROR)
108         *flow = ((ft)result - offset) / scale;
109
110     return error;
111 }
112
113 //=====
114 etError SF05_GetSerialNumber(u32t *serialNumber){
115 //=====
116     etError error = NO_ERROR; // error code
117     ul6t result; // read result from sensor
118
119     // write command "read serial number (bit 31:16)"
120     error = SF05_WriteCommand(READ_SERIAL_NUMBER_HIGH);
121
122     // if no error, read command result
123     if(error == NO_ERROR)
124         error = SF05_ReadCommandResult(&result);
125
126     // if no error, copy upper 16 bits to serial number
127     if(error == NO_ERROR)
128         *serialNumber = result << 16;
129
130     // if no error, write command "read serial number (bit 15:0)"
131     if(error == NO_ERROR)
132         error = SF05_WriteCommand(READ_SERIAL_NUMBER_LOW);
133
134     // if no error, read command result
135     if(error == NO_ERROR)
136         error = SF05_ReadCommandResult(&result);
137
138     // if no error, copy lower 16 bits to serial number
139     if(error == NO_ERROR)
140         *serialNumber |= result;
141
142     return error;
143 }
144
145 //=====
146 etError SF05_SoftReset(void){
147 //=====
148     etError error; // error code
149
150     error = SF05_WriteCommand(SOFT_RESET);
151
152     return error;
153 }
154
155 //=====
156 etError SF05_CheckCrc(u8t data[], u8t nbrOfBytes, u8t checksum){

```



```
157 //=====
158     u8t bit;        // bit mask
159     u8t crc = 0;    // calculated checksum
160     u8t byteCtr;    // byte counter
161
162     // calculates 8-Bit checksum with given polynomial
163     for(byteCtr = 0; byteCtr < nbrOfBytes; byteCtr++)
164     {
165         crc ^= (data[byteCtr]);
166         for(bit = 8; bit > 0; --bit)
167         {
168             if(crc & 0x80) crc = (crc << 1) ^ POLYNOMIAL;
169             else          crc = (crc << 1);
170         }
171     }
172
173     // verify checksum
174     if(crc != checksum) return CHECKSUM_ERROR;
175     else                return NO_ERROR;
176 }
177
178
```

```

1 //=====
2 //   S E N S I R I O N   AG, Laubisruetistr. 50, CH-8712 Staefa, Switzerland
3 //=====
4 // Project   : SF05 Sample Code (V1.1)
5 // File      : i2c_hal.h (V1.0)
6 // Author    : RFU
7 // Date     : 07-Nov-2012
8 // Controller: STM32F100RB
9 // IDE      : µVision V4.60.0.0
10 // Compiler  : Armcc
11 // Brief    : I2C hardware abstraction layer
12 //=====
13
14 #ifndef I2C_HAL_H
15 #define I2C_HAL_H
16
17 //-- Includes -----
18 #include "system.h"
19
20 //-- Defines -----
21 // I2C IO-Pins
22 // SDA on port C, bit 6
23 #define SDA_LOW() (GPIOC->BSRR = 0x00400000) // set SDA to low
24 #define SDA_OPEN() (GPIOC->BSRR = 0x00000040) // set SDA to open-drain
25 #define SDA_READ (GPIOC->IDR & 0x0040) // read SDA
26
27 // SCL on port C, bit 7
28 #define SCL_LOW() (GPIOC->BSRR = 0x00800000) // set SCL to low
29 #define SCL_OPEN() (GPIOC->BSRR = 0x00000080) // set SCL to open-drain
30 #define SCL_READ (GPIOC->IDR & 0x0080) // read SCL
31
32 //-- Enumerations -----
33 // I2C header
34 typedef enum{
35     I2C_ADR = 64, // default sensor I2C address
36     I2C_WRITE = 0x00, // write bit in header
37     I2C_READ = 0x01, // read bit in header
38     I2C_RW_MASK = 0x01 // bit position of read/write bit in header
39 }etI2cHeader;
40
41 // I2C acknowledge
42 typedef enum{
43     ACK = 0,
44     NO_ACK = 1,
45 }etI2cAck;
46
47 //=====
48 void I2c_Init(void);
49 //=====
50 // Initializes the ports for I2C interface.
51 //-----
52
53 //=====
54 void I2c_StartCondition(void);
55 //=====
56 // Writes a start condition on I2C-Bus.
57 //-----
58 // remark: Timing (delay) may have to be changed for different microcontroller.
59 //
60 // SDA: _____|_____
61 //
62 // SCL: _____|____
63
64 //=====
65 void I2c_StopCondition(void);
66 //=====
67 // Writes a stop condition on I2C-Bus.
68 //-----
69 // remark: Timing (delay) may have to be changed for different microcontroller.
70 //
71 // SDA: _____|_____
72 //
73 // SCL: _____|_____
74
75 //=====
76 etError I2c_WriteByte(u8t txByte);
77 //=====
78 // Writes a byte to I2C-Bus and checks acknowledge.

```

```
79 //-----
80 // input:  txByte          transmit byte
81 //
82 // return: error:          ACK_ERROR = no acknowledgment from sensor
83 //                        NO_ERROR  = no error
84 //
85 // remark: Timing (delay) may have to be changed for different microcontroller.
86
87 //=====
88 u8t I2c_ReadByte(etI2cAck ack);
89 //=====
90 // Reads a byte on I2C-Bus.
91 //-----
92 // input:  ack              Acknowledge: ACK or NO_ACK
93 //
94 // return: rxByte
95 //
96 // remark: Timing (delay) may have to be changed for different microcontroller.
97
98 #endif
99
```

```

1  //=====
2  //   S E N S I R I O N   AG, Laubisruetistr. 50, CH-8712 Staefa, Switzerland
3  //=====
4  // Project   : SF05 Sample Code (V1.1)
5  // File      : i2c_hal.c (V1.0)
6  // Author    : RFU
7  // Date      : 07-Nov-2012
8  // Controller: STM32F100RB
9  // IDE       : µVision V4.60.0.0
10 // Compiler  : Armcc
11 // Brief     : I2C hardware abstraction layer
12 //=====
13
14 //-- Includes -----
15 #include "i2c_hal.h"
16
17 //=====
18 void I2c_Init(void){
19 //=====
20     RCC->APB2ENR |= 0x00000010; // I/O port C clock enabled
21
22     GPIOC->CRL  &= 0x00FFFFFF; // set open-drain output for SDA and SCL
23     GPIOC->CRL  |= 0x55000000; // port C, bit 6,7
24
25     SDA_OPEN(); // I2C-bus idle mode SDA released
26     SCL_OPEN(); // I2C-bus idle mode SCL released
27 }
28
29 //=====
30 void I2c_StartCondition(void){
31 //=====
32     SDA_OPEN();
33     DelayMicroSeconds(1);
34     SCL_OPEN();
35     DelayMicroSeconds(1);
36     SDA_LOW();
37     DelayMicroSeconds(10); // hold time start condition (t_HD;STA)
38     SCL_LOW();
39     DelayMicroSeconds(10);
40 }
41
42 //=====
43 void I2c_StopCondition(void){
44 //=====
45     SCL_LOW();
46     DelayMicroSeconds(1);
47     SDA_LOW();
48     DelayMicroSeconds(1);
49     SCL_OPEN();
50     DelayMicroSeconds(10); // set-up time stop condition (t_SU;STO)
51     SDA_OPEN();
52     DelayMicroSeconds(10);
53 }
54
55 //=====
56 etError I2c_WriteByte(u8t txByte){
57 //=====
58     u8t      mask;
59     etError error = NO_ERROR;
60     for(mask = 0x80; mask > 0; mask >>= 1)// shift bit for masking (8 times)
61     {
62         if((mask & txByte) == 0) SDA_LOW(); // masking txByte, write bit to SDA-Line
63         else                      SDA_OPEN();
64         DelayMicroSeconds(1); // data set-up time (t_SU;DAT)
65         SCL_OPEN(); // generate clock pulse on SCL
66         DelayMicroSeconds(5); // SCL high time (t_HIGH)
67         SCL_LOW();
68         DelayMicroSeconds(1); // data hold time(t_HD;DAT)
69     }
70     SDA_OPEN(); // release SDA-line
71     SCL_OPEN(); // clk #9 for ack
72     DelayMicroSeconds(1); // data set-up time (t_SU;DAT)
73     if(SDA_READ) error = ACK_ERROR; // check ack from i2c slave
74     SCL_LOW();
75     DelayMicroSeconds(20); // wait to see byte package on scope
76     return error; // return error code
77 }
78

```

```
79 //=====
80 u8t I2c_ReadByte(etI2cAck ack){
81 //=====
82     u8t mask;
83     u8t rxByte = NO_ERROR;
84     SDA_OPEN(); // release SDA-line
85     for(mask = 0x80; mask > 0; mask >>= 1) // shift bit for masking (8 times)
86     {
87         SCL_OPEN(); // start clock on SCL-line
88         DelayMicroSeconds(3); // SCL high time (t_HIGH)
89         if(SDA_READ) rxByte = rxByte | mask; // read bit
90         SCL_LOW();
91         DelayMicroSeconds(1); // data hold time(t_HD;DAT)
92     }
93     if(ack == ACK) SDA_LOW(); // send acknowledge if necessary
94     else SDA_OPEN();
95     DelayMicroSeconds(1); // data set-up time (t_SU;DAT)
96     SCL_OPEN(); // clk #9 for ack
97     DelayMicroSeconds(5); // SCL high time (t_HIGH)
98     SCL_LOW();
99     SDA_OPEN(); // release SDA-line
100    DelayMicroSeconds(20); // wait to see byte package on scope
101    return rxByte; // return error code
102 }
103
```

```
1  //=====
2  //   S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
3  //=====
4  // Project   :  SF05 Sample Code (V1.1)
5  // File      :  system.h (V1.0)
6  // Author    :  RFU
7  // Date      :  07-Nov-2012
8  // Controller:  STM32F100RB
9  // IDE       :  µVision V4.60.0.0
10 // Compiler  :  Armcc
11 // Brief     :  System functions, global definitions
12 //=====
13
14 #ifndef SYSTEM_H
15 #define SYSTEM_H
16
17 //-- Includes -----
18 #include <stm32f10x.h>           // controller register definitions
19 #include "typedefs.h"          // type definitions
20
21 //-- Enumerations -----
22 // Error codes
23 typedef enum{
24     NO_ERROR           = 0x00, // no error
25     ACK_ERROR          = 0x01, // no acknowledgment error
26     CHECKSUM_ERROR     = 0x02  // checksum mismatch error
27 }etError;
28
29 //=====
30 void SystemInit(void);
31 //=====
32 // Initializes the system
33 //-----
34
35 //=====
36 void DelayMicroSeconds(u32t nbrOfUs);
37 //=====
38 // Wait function for small delays.
39 //-----
40 // input:  nbrOfUs    wait x times approx. one micro second (fcpu = 8MHz)
41 // return: -
42 // remark: smallest delay is approx. 15us due to function call
43
44 #endif
45
```

```
1  //=====
2  //   S E N S I R I O N   AG,  Laubisruetistr. 50, CH-8712 Staefa, Switzerland
3  //=====
4  // Project   :  SF05 Sample Code (V1.1)
5  // File      :  system.c (V1.0)
6  // Author    :  RFU
7  // Date      :  07-Nov-2012
8  // Controller:  STM32F100RB
9  // IDE       :  µVision V4.60.0.0
10 // Compiler  :  Armcc
11 // Brief     :  System functions
12 //=====
13
14 //-- Includes -----
15 #include "system.h"
16
17 //=====
18 void SystemInit(void)
19 //=====
20 {
21     // no initialization required
22 }
23
24 //=====
25 void DelayMicroSeconds(u32t nbrOfUs)
26 //=====
27 {
28     u32t i;
29     for(i = 0; i < nbrOfUs; i++)
30     {
31         __nop(); // nop's may be added or removed for timing adjustment
32         __nop();
33         __nop();
34         __nop();
35     }
36 }
37
38
```

```

1  //=====
2  //   S E N S I R I O N   AG, Laubisruetistr. 50, CH-8712 Staefa, Switzerland
3  //=====
4  // Project   : SF05 Sample Code (V1.1)
5  // File      : typedefs.h (V1.0)
6  // Author    : RFU
7  // Date      : 07-Nov-2012
8  // Controller: STM32F100RB
9  // IDE       : µVision V4.60.0.0
10 // Compiler  : Armcc
11 // Brief     : Definitions of typedefs for good readability and portability.
12 //=====
13
14 #ifndef TYPEDEFS_H
15 #define TYPEDEFS_H
16
17 //-- Defines -----
18 //Processor endian system
19 // #define BIG_ENDIAN //e.g. Motorola (not tested at this time)
20 #define LITTLE_ENDIAN //e.g. PIC, 8051, NEC V850
21 //=====
22 // basic types: making the size of types clear
23 //=====
24 typedef unsigned char  u8t;      ///< range: 0 .. 255
25 typedef signed char    i8t;      ///< range: -128 .. +127
26
27 typedef unsigned short u16t;     ///< range: 0 .. 65535
28 typedef signed short   i16t;     ///< range: -32768 .. +32767
29
30 typedef unsigned long  u32t;     ///< range: 0 .. 4'294'967'295
31 typedef signed long    i32t;     ///< range: -2'147'483'648 .. +2'147'483'647
32
33 typedef float          ft;       ///< range: +-1.18E-38 .. +-3.39E+38
34 typedef double         dt;       ///< range:                .. +-1.79E+308
35
36 //typedef bool          bt;       ///< values: 0, 1 (real bool used)
37
38 typedef union {
39     u16t u16;          // element specifier for accessing whole u16
40     i16t i16;          // element specifier for accessing whole i16
41     struct {
42         #ifdef LITTLE_ENDIAN // Byte-order is little endian
43             u8t u8L;         // element specifier for accessing low u8
44             u8t u8H;         // element specifier for accessing high u8
45         #else                // Byte-order is big endian
46             u8t u8H;         // element specifier for accessing low u8
47             u8t u8L;         // element specifier for accessing high u8
48         #endif
49     } s16;             // element spec. for acc. struct with low or high u8
50 } nt16;
51
52 typedef union {
53     u32t u32;          // element specifier for accessing whole u32
54     i32t i32;          // element specifier for accessing whole i32
55     struct {
56         #ifdef LITTLE_ENDIAN // Byte-order is little endian
57             u16t u16L;       // element specifier for accessing low u16
58             u16t u16H;       // element specifier for accessing high u16
59         #else                // Byte-order is big endian
60             u16t u16H;       // element specifier for accessing low u16
61             u16t u16L;       // element specifier for accessing high u16
62         #endif
63     } s32;             // element spec. for acc. struct with low or high u16
64 } nt32;
65
66 #endif
67

```