C-CODE EXAMPLE FOR SCP1000-D01 PRESSURE SENSOR

1 INTRODUCTION

The objective is to set up SPI communication between VTI Technologies' digital pressure sensor component and an MCU of an application device. In this code example:

- The MCU is configured
- SCP1000-D01 is initialized and configured
- The high resolution measurement mode is activated
- Temperature and pressure information is read always when the DRDY pin is in high state

Please refer to the document "SCP1000 Product Family Specification 8260800" for further information on SCP1000 register addressing and SPI communication. This document applies to the SCP1000-D01.

2 C-CODE

The C-language software example below shows an easy way to implement communication with the SCP1000-D01 via an SPI bus. The idea is that the example code can be implemented with many micro controllers.

The example C-code is written for the ATMEL Atmega16L microprocessor. The compiler used with Atmega16L for the example code is WinAVR-20050214. The example code is tested with the hardware presented in Figure 1 below.

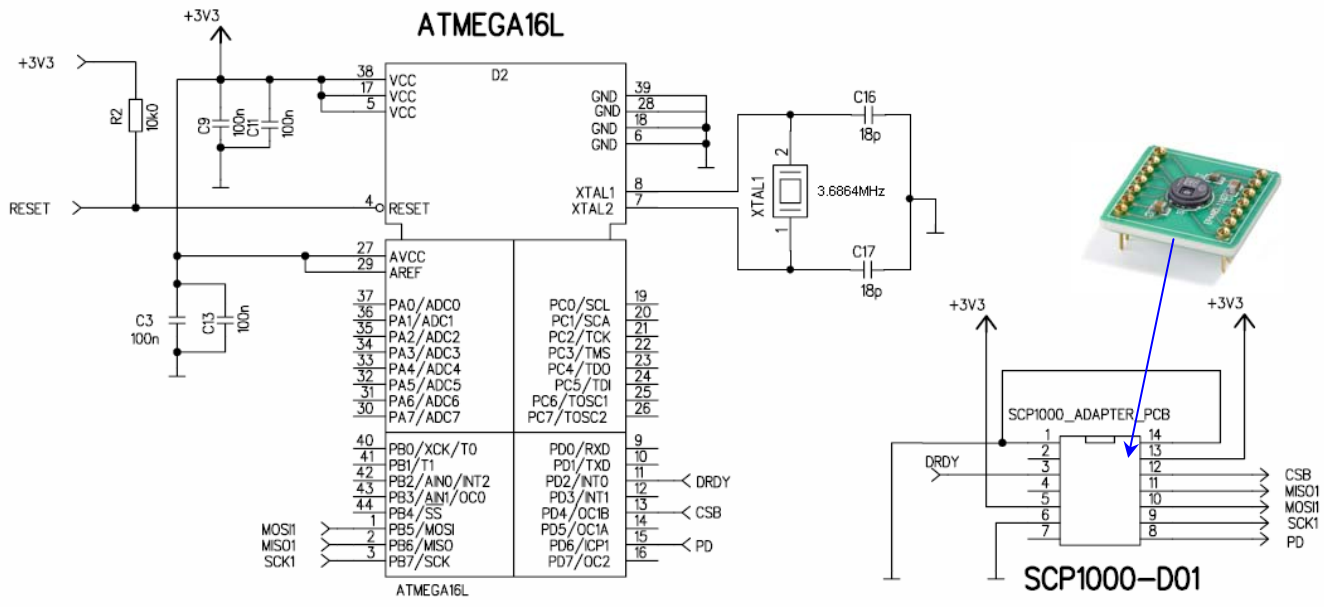

Figure 1. Circuit diagram of the hardware used in C-code testing (for reference only).

2.1 C-code example

```
/*****************************************************************
**** This C-code example is for reference only. The code below
**** is written for the SCP1000-D01 and the Atmel ATMEGA16L processor.
*****************************************************************
**** Name:     Main.c
**** Version:  1.0
*****************************************************************
**** The code:
****    - configures the MCU,
****    - initializes the SCP1000-D01
****    - activates the high resolution measurement mode
****    - reads temperature and pressure data from SCP1000-D01
****      always when the DRDY pin is '1'
*****************************************************************/

// CONSTANTS
#define XTAL 3686400L
#define BAUDRATE 230400L
#define UARTDIVISOR ((XTAL + 4 * BAUDRATE) / (16 * BAUDRATE) - 1)
#define TICKSPERMS (XTAL / 1000 / 5 - 1)

// LIBRARIES
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))

// SCP1000 PINS
#define SCP1000_PIN    PIND    // PORTD as input
#define SCP1000_PORT   PORTD   // PORTD
#define SCP1000_PD     6       // PD pin is located at PD6
#define SCP1000_DRDY   2       // DRDY pin is located at PD2
#define SCP1000_CSB    4       // CSB pin is located at PD4

// FUNCTIONS
void wait(uint16_t ms);
void fail(void);
void setup(void);
void init_scp1000(void);
void Write_Direct_Access_SPI(uint8_t address, uint8_t data);
uint16_t Read_Indirect_Access_SPI(uint8_t address);
uint16_t Read_Direct_Access_SPI(uint8_t address, uint8_t number_of_bytes);
void Write_Indirect_Access_SPI(uint8_t address, uint8_t data);

/*****************************************************************
 * WAIT, input: time to wait in [ms] (uint16_t)
 * ----
 * Execute X ms wait
 *****************************************************************/
void wait_ms(uint16_t ms)
{
        uint16_t a, b;
        for (a = ms; a > 0; a--)                    /* outer loop takes 5 ck per round */
                for (b = TICKSPERMS; b > 0; b--)    /* inner loop takes 5 ck per round */
                        asm("nop");
}
```

```
/*****************************************************************
 * FAIL
 * ----
 * If SCP1000 initialization fails, send an error message and stop execution
 *****************************************************************/
void fail(void)
{
        // <- Place for error message!
        //for(;;)              // Stop execution
        //      ;
}

/*****************************************************************
 * SETUP
 * -----
 * MCU configuration:
 *  - PortB for SPI
 *  - PortD for CSB, PD and DRDY and UART
 *  - SPI enabled and speed to Fosc/8
 *****************************************************************/
void setup(void)
{
// definition of ports, all unused pins are configured as an input
//-----------------------------------------------------------------
// PORTB SETUP ATMEGA16L
//-----------------------------------------------------------------
// |--------PORTB.7 (OUTPUT) SCK1
// ||-------PORTB.6 (INPUT)  MISO1
// |||------PORTB.5 (OUTPUT) MOSI1
// ||||-----PORTB.4 (OUTPUT) MCU slave select pin
// |||||----PORTB.3 (INPUT)
// ||||||---PORTB.2 (INPUT)
// |||||||--PORTB.1 (INPUT)
// ||||||||-PORTB.0 (INPUT)
// 10110000 = 0xB0
        PORTB    = 0x00;          //
        DDRB     = 0xB0;          // Set PORTB
//-----------------------------------------------------------------
// PORTD SETUP ATMEGA16L
//-----------------------------------------------------------------
// |--------PORTD.7 (INPUT)
// ||-------PORTD.6 (OUTPUT) PD
// |||------PORTD.5 (INPUT)
// ||||-----PORTD.4 (OUTPUT) CSB
// |||||----PORTD.3 (INPUT)
// ||||||---PORTD.2 (INPUT)  DRDY
// |||||||--PORTD.1 (OUTPUT) TxD
// ||||||||-PORTD.0 (INPUT)  RxD
// 01010010 = 0x52
        PORTD    = 0x08;          // Turn on internal pull-ups for PORTD
        DDRD     = 0xF2;          // Set PORTD
//-----------------------------------------------------------------
// Set up SPI bus
        SPCR = (1<<SPE)|(1<<MSTR)|(1<<CPHA)|(1<<CPOL)|(1<<SPR0);
//Set SPI speed to Fosc/8
        SPSR |= (1<<SPI2X);
}
```

```
/*************************************************************
 * INITIALIZE SCP1000 AND ACTIVATE MEASUREMENT MODE
 * -----------------------------------------------------------
 * 1. SCP1000 power supplies are to set up and stabilized before the SCP1000 is initialized
 * 2. Set CSB and PD high ('1') and wait 1ms
 * 3. Pull PD low ('0') and wait 60ms
 * 4. Read STATUS register (0x07) and check that STARTUP bit (LSB, bit0) is zero, if not
 *    SCP1000 start-up procedure is running --> check STATUS.STARTUP in loop for 6 times
 *    if it is not zero after 6 cycles, SCP1000 start-up has failed
 * 5. Read DATARD8 register and check that LSB (bit0) is one, if not
 *    EEPROM checksum error is detected --> SCP1000 will not give reliable pressure data
 * 6. Low noise configuration (done with indirect write)
 *    Write 0x2D to ADDPTR (0x02)
 *    Write 0x03 to DATAWR (0x01)
 *    Write 0x02 to OPERATION (0x03)
 * 7. wait 100ms
 * 8. SCP1000 is in standby mode
 * 9. Set measurement mode (High resolution in this example).
 *************************************************************/
void init_scp1000(void)
{
        sei();
        uint16_t DATA;
        uint8_t  i;

        sbi(SCP1000_PORT, SCP1000_CSB);              // #2, CSB high
        sbi(SCP1000_PORT, SCP1000_PD);               // #2, PD high
        wait_ms(1);                                  // #2, 1ms wait
        cbi(SCP1000_PORT, SCP1000_PD);               // #3, PD low
        wait_ms(60);                                 // #3, 60ms wait

        for(i = 6; i > 0 ; i--)
        {
                DATA = Read_Direct_Access_SPI(0x07, 1);// #4, read STATUS register --> LSB '0'=OK

                if(!(DATA & 0x0001))
                        break;

                wait_ms(10);
        }
        if(i == 0)
                fail();

        DATA = Read_Direct_Access_SPI(0x1F, 1);      // #5, read DATARD8 register --> LSB '1'=OK

        if(!(DATA && 0x0001))
                fail();

        Write_Indirect_Access_SPI(0x2D, 0x03);       // #6, Low noise configuration
        wait_ms(100);                                // #7, wait for 100ms

        Write_Direct_Access_SPI(0x03, 0x00);         // Reset operation mode
        wait_ms(10);                                 // Wait before change new mode
        Write_Direct_Access_SPI(0x03, 0x0A);         // #9, set SCP1000 into 'high resolution'
                                                     // measurement mode (0x0A)

        wait_ms(100);
}
```

```
/*****************************************************************
 * DIRECT WRITE, inputs: register address, register data
 * ------------
 * 1. Convert the SCP1000 register address to real address by shifting the
 *    bit pattern to left by 2 bits ([xxAAAAAA] --> [A A A A A A RW 0], A = address bit):
 *    (MSB) A A A A A A A RW 0 (LSB)
 *          |           |  | *- allways zero
 *          |           |  *--- Read/Write Bit (Read: RW=0, Write: RW=1)
 *          *-----------*------ Register address bits
 * 2. Set read/write bit to '1' at the address byte:
 *    [A A A A A A RW 0] --> [A A A A A A 1 0]
 * 3. Set CSB to low
 * 4. Send register address byte
 * 5. Send register content (data byte)
 * 6. Set CSB to HIGH
 *****************************************************************/
void Write_Direct_Access_SPI(uint8_t address, uint8_t data)
{
    address = (address << 2);                       // #1, shift the SCP1000 reg address to left by 2
    address |= 0x02;                                // #2, set write bit to one (RW=1)
    cbi(SCP1000_PORT, SCP1000_CSB);                 // #3, set CSB to low
        SPDR = address;                             // #4, write register address byte (8bits)
                                                    // to SPI line by using SPI HW

        while(!(SPSR & (1<<SPIF)))                  // Wait register address to be written
                asm("nop");

        SPDR = data;                                // #5, write data byte (8bit) to line
        while(!(SPSR & (1<<SPIF)))                  // Wait register data to be written
                asm("nop");

        sbi(SCP1000_PORT, SCP1000_CSB);             // 6#, Set CSB to HIGH
}

/*****************************************************************
 * DIRECT READ, inputs: register address, number of bytes to be read
 * ------------
 * 1. Convert the SCP1000 register address to real address by shifting the
 *    bit pattern to left by 2 bits ([xxAAAAAA] --> [A A A A A A RW 0], A = address bit):
 *    (MSB) A A A A A A A RW 0 (LSB)
 *          |           |  | *- allways zero
 *          |           |  *--- Read/Write Bit (Read: RW=0, Write: RW=1)
 *          *-----------*------ Register address bits
 * 2. Set Read/Write bit to '0' at the address byte:
 *    [A A A A A A RW 0] --> [A A A A A A 0 0]
 *    The RW bit is zero automatically after the register address is shifted to left by 2
 * 3. Set CSB to low
 * 4. Send address byte (register)
 * 5. Send SCK (clock cycles) and read data bytes
 * 6. Set CSB to HIGH
 *****************************************************************/
uint16_t Read_Direct_Access_SPI(uint8_t address, uint8_t number_of_bytes)
{
        uint16_t value;

        value = 0;

        address = (address << 2);                   // #1, shift the SCP1000 reg address to left by 2
        cbi(SCP1000_PORT, SCP1000_CSB);             // #3, set CSB to low
        //SPCR |= (1<<SPIE);

        SPDR = address;                             // #4, write register address byte (8bits)
```

```
                                                // to SPI line by using SPI HW

        while(!(SPSR & (1<<SPIF)))              // Wait register address to be written
        {
        asm("nop");
        }

        while(number_of_bytes-- > 0)
        {
                value <<= 8;
                SPDR = 0x00;                    // #5, Send SCK pulses (=send 0x00 to line)
                while(!(SPSR & (1<<SPIF)))      // Read data from SPI bus.
                        asm("nop");
                value |= SPDR;                  // Read data byte from MCU SPI register
        }
        sbi(SCP1000_PORT, SCP1000_CSB);         // 6#, Set CSB to HIGH

        return value;
}

/*************************************************************
 * INDIRECT READ, input: indirect register address
 * -------------
 * Indirect read requires three DIRECT read/write actions:
 * 1. Use DIRECT WRITE to write the register address to ADDPTR (0x02)
 * 2. Use DIRECT WRITE to write the "read indirect" command 0x01 to OPERATION (0x03)
 * 3. Wait 10ms
 * 4. Use DIRECT READ to read 2 bytes from DATARD16 (0x20), the register content
 *    is in bits [7:0], bits [15:8] should be treated as zeros
 *************************************************************/
uint16_t Read_Indirect_Access_SPI(uint8_t address)
{
        uint32_t value;
        Write_Direct_Access_SPI(0x02, address);     // #1, write reg address to ADDPTR (0x02)
        Write_Direct_Access_SPI(0x03, 0x01);        // #2, write 0x01 to OPERATION (0x03)
        wait_ms(10);                                // #3, wait 10ms
        value = Read_Direct_Access_SPI(0x20, 0x02); // #4, read two bytes from DATARD16 (0x20)
        return value;
}

/*************************************************************
 * INDIRECT WRITE, input: indirect register address, register data
 * --------------
 * Indirect write requires three DIRECT WRITE actions:
 * 1. Use DIRECT WRITE to write register address to ADDPTR (0x02)
 * 2. Use DIRECT WRITE to write data to DATAWR (0x01)
 * 3. Use DIRECT WRITE to write "indirect write" command 0x02 to OPERATION (0x03)
 * 4. Wait 50ms
 *************************************************************/
void Write_Indirect_Access_SPI(uint8_t address, uint8_t data)
{
        Write_Direct_Access_SPI(0x02, address);     // #1, write reg address to ADDPTR (0x02)
        Write_Direct_Access_SPI(0x01, data);        // #2, write reg data to DATAWR (0x01)
        Write_Direct_Access_SPI(0x03, 0x02);        // #3, write 0x02 to OPERATION (0x03)
        wait_ms(50);                                // #4, wait 50ms
}
```

```
/*************************************************************
 * MAIN
 * ----
 * 1. Setup, configure MCU
 * 2. Init_SCP1000, initialize SCP1000 and activate measurement mode
 * 3. Read SCP1000 pressure and temperature every time when new measurement
 *    results are available (e.g. DRDY pin is '1')
 *************************************************************/
int main(void)
{
        uint16_t temp;
        uint32_t pressure;

        setup();                                        // #1, configure MCU

        init_scp1000();                                 // #2, Initialize SCP1000, activate
                                                        // 'high resolution' measurement mode
        for(;;)
        {
                while(!(SCP1000_PIN & (1 << SCP1000_DRDY)))  // Wait for DRDY pin='1'
                        wait_ms(1);

                temp = Read_Direct_Access_SPI(0x21, 2);// Read 2 bytes from TEMPOUT (0x21)
                temp = temp & 0x3fff;                   // Mask bits [13:0]
                temp = temp / 20;                       // Convert temperature to [°C] by dividing
                                                        // the result by 20

                pressure = (uint32_t)((0x0007) & Read_Direct_Access_SPI(0x1F, 1));
                                                        // Read 1 byte from DATARD8 (0x1F) and
                                                        // mask bits [2:0] in order to get
                                                        // 3MSB bits for pressure information

                pressure <<= 16;                        // shift 3MSB bits to left by 16

                pressure |= (uint32_t)Read_Direct_Access_SPI(0x20, 2);
                                                        // Read 2 bytes from DATARD16 (0x20) in
                                                        // order to get 16 LSB bits for pressure
                                                        // information.

                pressure >>= 2;                         // Convert to [Pa] by dividing the 19 bit
                                                        // pressure by 4 --> the 19 bit pattern
                                                        // is shifted to right by 2


                // <- Here user can add temperature and pressure data send / display routines
                //    as needed for application. Notice that temperature is in 2's complement format.
        }

        return 0;
}

/* EOF */
```