intersema

AN502 (V1.0) April 25th 2002

Software library for MS5534 and MS5535

1 Introduction

Interfacing the MS5534 and MS5535 with a computer or a microcontroller is very easy thanks to the serial digital interface that is provided by the pressure module. Both components rely on the Sensor Interface ASIC for communication and data conversion.
This document provides C/C++ source code examples that access the MS5534 and the MS5535, thus reducing Time-To-Market and reducing development risks for our customers.

Warning: The source code provided in this document and in the additional source files is given for information to customers to help them developing their application. The reference documents are always the controlled datasheets of the products, and customers should refer to these documents instead of the source code for the specifications.

1.1 Support of C and C++

The functions described in this document have been developed in C++. However, in order to be able to use this code in C applications, we provide you with a C-compatible source code. People willing to work in C++ can very quickly transform it into the appropriate classes.

2 Communication with the Sensor Interface ASIC

The MS5534 and MS5535 are both based on the Sensor Interface ASIC which handles A-to-D conversion as well as the serial communication. Thus low-level functions are product independent (see figure 1). Future products based on the same IC might re-use the same library.
Sensor specific functions decode the sensor's coefficients (Cx) from the calibration words (Wx) and calculate the compensated pressure and temperature. The ASIC access functions rely on a few hardware and time related functions and are thus platform dependent.

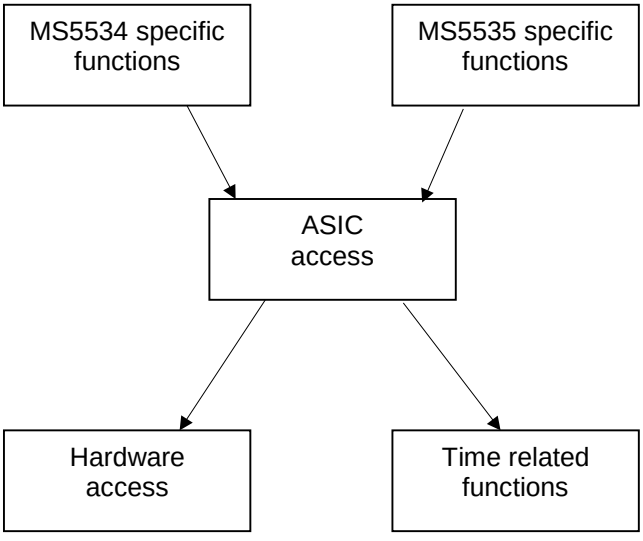

Figure 1. Software hierarchy

2.1 Hardware related functions

The MS5534 and MS5535 modules have the following pins:

- VDD
- GND
- PEN
- PV
- MCLK
- SCLK
- DIN
- DOUT

During normal operation, the software controls only SCLK, DIN and DOUT while MCLK is connected to a 32kHz oscillator. The other pins have static voltage as described in the datasheet.
The interface with the IC is a simple 3-wire interface handling both coefficients and acquisition reading. Output signals are SCLK and DIN, while the input signal is DOUT. To control and read these signals, the software described in this application note uses the following functions:

```
long sensor_controlInit   ();  // 0 = no error
void sensor_controlExit   (void);

void setSCLK  (bool state);    // set SCLK to the specified state (true/false)
bool getSCLK  (void);          // returns the current state of SCLK

void setDIN   (bool state);    // set DIN  to the specified state (true/false)
bool getDIN   (void);          // returns the current state of DIN

bool getDOUT  (void);          // returns the current state of DOUT
```

sensor_controlInit() and *sensor_controlExit()* functions are used to initialize and exit the hardware control if necessary.

The implementation of these functions is platform dependent. An example for the parallel port, working on Win32 (i.e. Windows 95/98,...NT/2000/) is given in appendix A.

2.2 Time related functions

Communication using serial interface must meet several time constraints. In particular serial data shift must not be too fast. Therefore, we use the WaitOnePulse function to put sufficient time between the two SCLK edges. Please refer to the module's datasheet for the exact value. Typically this function is implemented using a busy loop (especially on microcontroller).

```
void WaitOnePulse (void);      // Wait for a "pulse" duration
```

Note that one function of this library also uses the C run-time **time()** function. For more details, please refer to the description of the **waitOnDoutFall()** below.

2.3 Sensor Interface ASIC access functions

The sensor interface related functions give access to the module through the serial interface. They provide access to the ADC through reading D1 and D2, and also to the coefficient memory. The function prototypes are given hereafter.

```
void reset (void);       // send a reset sequence to the IC
long getW  (long index); // Read the corresponding calibration word of the IC (index [1:4])
long getD1 (long *error_pt); // Start a D1 acquisition, wait for end of conversion and
                             // return the value
long getD2 (long *error_pt); // Start a D2 acquisition, wait for end of conversion and
                             // return the value
```

These functions rely on a few sub-functions described below:

```
long waitOnDoutFall     (void);    // wait until a falling edge on DOUT is detected
long SerialGet16        (void);    // shift in (from IC to PC) a 16 bit value
void SerialSendLsbFirst (char pattern, char nbr_clock);  // shift out (from PC to IC) a
                                   // sequence of bits
```

2.3.1 waitOnDoutFall

This function make a busy loop polling on the DOUT pin. It waits until DOUT goes low. If no module is connected, the DOUT pin might remain at 1 forever. Thus in some application, it is necessary to implement a timeout that would stop the loop after a certain time. This is especially useful in some Microsoft Windows application. In embedded applications, this timeout is usually removed.
To implement the timeout function, we're using the C run-time **time()** function. The loop's duration is checked from time to time. If DOUT is not low within 1 seconds, the loop is aborted.

```
/* ========================================================================= */
/*                            waitOnDoutFall                                 */
/* ========================================================================= */
long waitOnDoutFall(void)
{
    bool     working;
    long     cnt;
    unsigned long t0;
    unsigned long t1;
    long     error;

    working = true;
    error   = 0;

    WaitOnePulse();

    t0  = (unsigned long) time(0);
    cnt = 0;

    while(working)
    {
        working = getDOUT();
        cnt++;
        WaitOnePulse();
        if (cnt>=100)
        {
            t1 = (unsigned long) time(0);
            if ((t1-t0)>1)
            {
                working = false;
                error   = 1;
            }
            cnt = 0;
        }
    };

    return(error);
}
```

2.3.2 SerialGet16

This function shifts in a 16-bit value of the Sensor Interface IC. Note that we read DOUT after the allowing rising edge of SCLK to be sure that the IC has had enough time to set the data on the DOUT pin. This function is used mainly to fetch the Wx, D1 and D2 words out of the IC.

```
/* ================================================================== */
/*                            SerialGet16                             */
/* ================================================================== */
long SerialGet16(void)
{
    char i;
    long v;

    v = 0;
    setSCLK(false);
    WaitOnePulse();

    for (i=0; i<16; i++)
    {
        setSCLK(true);
        WaitOnePulse();
        setSCLK(false);
        v = v << 1;
        if (getDOUT())
            v = v | 1;
        WaitOnePulse();
    }
    return(v);
}
```

2.3.3 SerialSendLsbFirst

This function generates a serial pattern on DIN. It generated **nbr_clock** cycles and the value of DIN is set according to the pattern. The first data transmitted is the bit 0 of pattern, the second data is bit 1 (thus LSB first). This function is used mainly to send the commands to the IC.

```
/* ================================================================== */
/*                          SerialSendLsbFirst                        */
/* ================================================================== */
void SerialSendLsbFirst(char pattern, char nbr_clock)
{
    char i;
    char c;

    setSCLK(false);
    WaitOnePulse();
    for (i=0; i<nbr_clock; i++)
    {
        c = (char) (pattern & 1);
        if (c==1)
            setDIN(true);
        else
            setDIN(false);
        WaitOnePulse();
        setSCLK(true);
        WaitOnePulse();
        setSCLK(false);
        pattern = (char) (pattern >> 1);
    }
}
```

2.3.4 reset

This function sends a reset sequence to the Sensor Interface IC.

```
/* ====================================================================== */
/*                            reset                                       */
/* ====================================================================== */
void reset(void)
{
    SerialSendLsbFirst(0x55, 8);
    SerialSendLsbFirst(0x55, 8);
    SerialSendLsbFirst(0x00, 5);
}
```

2.3.5 getW

This function read the W coefficients stored in the Sensor Interface IC. The index value for W1 is 1, 2 for W2 and so on. Note that we generate a single pulse on SCLK AFTER reading the data to be compliant with the datasheet. This pulse is often forgotten.

```
/* ====================================================================== */
/*                                getW                                    */
/* ====================================================================== */
long getW            (long index)  // 1 to 4
{
    long data;

    data = 0;
    switch(index)
    {
        case 1:
            SerialSendLsbFirst(0x57, 8);
            SerialSendLsbFirst(0x01, 5);
            data = SerialGet16();
            break;

        case 2:
            SerialSendLsbFirst(0xD7, 8);
            SerialSendLsbFirst(0x00, 5);
            data = SerialGet16();
            break;

        case 3:
            SerialSendLsbFirst(0x37, 8);
            SerialSendLsbFirst(0x01, 5);
            data = SerialGet16();
            break;

        case 4:
            SerialSendLsbFirst(0xB7, 8);
            SerialSendLsbFirst(0x00, 5);
            data = SerialGet16();
            break;
    }
    SerialSendLsbFirst(0x00, 1);  // to be compliant with the data sheet
    return(data);
}
```

2.3.6 getD1

This function starts a D1 acquisition, waits for the end of conversion and reads the result out of the IC.

```
/* ====================================================================== */
/*                                getD1                                   */
/* ====================================================================== */
long getD1            (long *error_pt)
{
    long d1;
    long error;

    SerialSendLsbFirst(0x2F, 8);
    SerialSendLsbFirst(0x00, 2);
    error = 0;
    if (getDOUT()==false)
        error = 1;              // line should be at 1 now
    SerialSendLsbFirst(0x00, 2);

    if (!error)
        error = waitOnDoutFall();

    if (!error)
        d1 = SerialGet16();
    else
        d1 = 0;

    SerialSendLsbFirst(0x00, 1);  // to be compliant with the data sheet
    if (error_pt!=0)
        *error_pt = error;
    return(d1);
}
```

2.3.7 getD2

This function starts a D2 acquisition, waits for the end of conversion and reads the result out of the IC.

```
/* ====================================================================== */
/*                                getD2                                   */
/* ====================================================================== */
long getD2            (long *error_pt)
{
    long d2;
    long error;

    SerialSendLsbFirst(0x4F, 8);
    SerialSendLsbFirst(0x00, 3); // Note the difference with getD1
    error = 0;
    if (getDOUT()==false)
        error = 1;              // line should be at 1 now
    SerialSendLsbFirst(0x00, 1);

    if (!error)
        error = waitOnDoutFall();

    if (!error)
        d2 = SerialGet16();
    else
        d2 = 0;
    if (error_pt!=0)
        *error_pt = error;
    return(d2);
}
```

3 MS5535 specific functions

The functions described here are specific for the MS5535. They either process the data read from the sensor interface IC or send commands to it.

```
long ConvertWtoC5535 (int ix, long W1, long W2, long W3, long W4);
long ConvertCtoW5535 (int ix, long C1, long C2, long C3, long C4, long C5, long C6);
void calcPT5535      (double *pressure, double *temperature, long d1_arg, long d2_arg);
```

3.1 Coefficient conversion

3.1.1 ConvertWtoC

This functions converts the W1-W4 to one of the C coefficients. The index **ix** must be in range 1 to 6

```
/* ------------------------------------------------------------------------ */
/* -------------------------- ConvertWtoC5535 ------------------------------ */
/* ------------------------------------------------------------------------ */
long ConvertWtoC5535 (int ix, long W1, long W2, long W3, long W4)
{
    long c;
    long x, y;

    c = 0;
    switch(ix)
    {
        case 1:
                    c =  (W1 >> 3) & 0x1FFF;
                    break;
        case 2:
                    x = (W1 <<10) & 0x1C00;
                    y = (W2 >> 6) & 0x03FF;
                    c =  x | y;
                    break;
        case 3:
                    c = (W3 >> 6) & 0x03FF;
                    break;
        case 4:
                    c = (W4 >> 7) & 0x01FF;
                    break;
        case 5:
                    x = (W2 << 6 )& 0x0FC0;
                    y =  W3       & 0x003F;
                    c =  x | y;
                    break;
        case 6:
                    c = W4 & 0x007F;
                    break;
    }
    return(c);
}
```

3.1.2 ConvertCtoW

This functions converts the C1-C6 to one of the W coefficients. The index **ix** must be in range 1 to 4

```
/* ---------------------------------------------------------------------- */
/* --------------------------- ConvertCtoW5535 --------------------------- */
/* ---------------------------------------------------------------------- */
long ConvertCtoW5535 (int ix, long C1, long C2, long C3, long C4, long C5, long C6)
{
    long w;

    C1 = C1 & 0x1FFF;
    C2 = C2 & 0x1FFF;
    C3 = C3 & 0x03FF;
    C4 = C4 & 0x01FF;
    C5 = C5 & 0x0FFF;
    C6 = C6 & 0x007F;

    w = 0;
    switch(ix)
    {
        case 1 :
                    w =  ( C1<<3) | (C2 >> 10);
                    break;
        case 2 :
                    w =  ((C2 & 0x3FF) << 6) | ((C5>>6) & 0x3F);
                    break;
        case 3 :
                    w =  ( C3 << 6) | (C5 & 0x3F);
                    break;
        case 4 :
                    w =  ( C4 << 7) | C6;
                    break;
    }

    w = w & 0xFFFF;
    return(w);
}
```

3.2 Pressure and Temperature calculation

The following function makes the conversion from D1/D2 to pressure and temperature. This function doesn't calculate the temperature using the second order algorithm.
The fc[] variables are **double** values of the C coefficients of the sensor. As explained in the introduction, the functions described here have been developed for C++ where fc[] are members of a Sensor5535 class. Please refer to the source code given in appendix A for a real example on how to use this code.

```
/* ---------------------------------------------------------------------- */
/* --------------------------- calcPT5535 -------------------------------- */
/* ---------------------------------------------------------------------- */
void calcPT5535    (double *pressure, double *temperature, long d1_arg, long d2_arg)
{
    double dt, off, sens;
    double fd1, fd2, x;

    d1_arg = d1_arg & 0xFFFF;
    d2_arg = d2_arg & 0xFFFF;

    fd1 = (double) d1_arg;
    fd2 = (double) d2_arg;

    dt          = -10000.0 +  fd2 - (8.0 * fc[5]);
    off         =  10000.0 +  fc[2]       + (  (   ( fc[4]-250.0) *  dt ) / 4096.0);
    sens        =   3000.0 + (fc[1] / 2.0) + (  (   ( fc[3]+200.0) *  dt ) / 8192.0);
    if (pressure!=0)
        *pressure    =   1000.0 +    (( sens * (fd1- off)) / 4096.0);
    if (temperature!=0)
        *temperature =  ( 200   +    ((  dt * ( fc[6]+100.0) ) / 2048.0)) / 10.0;
}
```

4 MS5534 specific functions

The functions described here are specific for the MS5534. They either process the data read from the sensor interface IC or send commands to it.

```
long ConvertWtoC5534 (int ix, long W1, long W2, long W3, long W4);
long ConvertCtoW5534 (int ix, long C1, long C2, long C3, long C4, long C5, long C6);
void calcPT5534      (double *pressure, double *temperature, long d1_arg, long d2_arg);
```

4.1 Coefficients conversion

4.1.1 ConvertWtoC

This functions converts the W1-W4 to one of the C coefficients. The index **ix** must be in range 1 to 6

```
/* ------------------------------------------------------------------------ */
/* -------------------------- ConvertWtoC5534 ----------------------------- */
/* ------------------------------------------------------------------------ */
long ConvertWtoC5534 (int ix, long W1, long W2, long W3, long W4)
{
    long c;
    long x, y;

    c = 0;
    switch(ix)
    {
        case 1:
                    c =  (W1 >> 1) & 0x7FFF;
                    break;
        case 2:
                    x = (W3 << 6) & 0x0FC0;
                    y =  W4       & 0x003F;
                    c = x | y;
                    break;
        case 3:
                    c = (W4 >> 6) & 0x03FF;
                    break;
        case 4:
                    c = (W3 >> 6) & 0x03FF;
                    break;
        case 5:
                    x = (W1 << 10)& 0x0400;
                    y = (W2 >> 6 )& 0x03FF;
                    c = x | y;
                    break;
        case 6:
                    c =  W2       & 0x003F;
                    break;
    }
    return(c);
}
```

4.1.2 ConvertCtoW

This functions converts the C1-C6 to one of the W coefficients. The index **ix** must be in range 1 to 4

```
/* ---------------------------------------------------------------------- */
/* --------------------------- ConvertCtoW5534 --------------------------- */
/* ---------------------------------------------------------------------- */
long ConvertCtoW5534 (int ix, long C1, long C2, long C3, long C4, long C5, long C6)
{
    long w;

    C1 = C1 & 0x7FFF;
    C2 = C2 & 0x0FFF;
    C3 = C3 & 0x03FF;
    C4 = C4 & 0x03FF;
    C5 = C5 & 0x07FF;
    C6 = C6 & 0x003F;

    w = 0;
    switch(ix)
    {
        case 1 :
                    w =  (C1<<1) | (C5 >> 10);
                    break;
        case 2 :
                    w =  ((C5 & 0x3FF) << 6) | C6;
                    break;
        case 3 :
                    w =  (C4 << 6) | (C2 >> 6);
                    break;
        case 4 :
                    w =  (C3 << 6) | (C2 & 0x003F);
                    break;
    }

    w = w & 0xFFFF;
    return(w);
}
```

4.2 Pressure and Temperature calculation

The following function makes the conversion from D1/D2 to pressure and temperature. This function doesn't calculate the temperature using the second order algorithm.
The fc[] variables are **double** values of the C coefficients of the sensor. As explained in the introduction, the functions described here have been developed for C++ where fc[] are members of a Sensor5534 class. Please refer to the source code given in appendix A for a real example on how to use this code.

```
/* ---------------------------------------------------------------------- */
/* --------------------------- calcPT5534 -------------------------------- */
/* ---------------------------------------------------------------------- */
void calcPT5534    (double *pressure, double *temperature, long d1_arg, long d2_arg)
{
    double dt, off, sens;
    double fd1, fd2, x;

    d1_arg = d1_arg & 0xFFFF;
    d2_arg = d2_arg & 0xFFFF;

    fd1 = (double) d1_arg;
    fd2 = (double) d2_arg;

    dt          =   fd2 - ((8.0 * fc[5]) + 20224.0);
    off         =   fc[2] * 4.0         + (  (    ( fc[4]-512.0) *  dt ) / 4096.0);
    sens        =   24576.0 +  fc[1]    + (  (      fc[3] *  dt ) / 1024.0);
    x           =   (( sens * (fd1- 7168.0)) / 16384.0) -off;
    if (pressure!=0)
        *pressure    =   250.0 +   x / 32;
    if (temperature!=0)
        *temperature =  20.0 +      ( (  dt * ( fc[6]+50.0) ) / 10240.0);
}
```

Appendix A : Implementation for the parallel port

The source code described in this application note can be downloaded with the file AN502_01_src.zip. This file can be found on Intersema's web site (http://www.intersema.ch)

A.2 Requirement

The ZIP file contains a project for Borland C++ Builder 5.0 and for Microsoft Visual C++ 6.0. They work with the parallel port module manufactured and distributed by Intersema. Anyway this program will also work with other modules as long as they use the same pin-out:

pin 2	SCLK
pin 3	DIN
pin 4	VDD
pin 12	DOUT
pin 25	GND

Note: The pin 4 is used for the power supply of the parallel port module. This signal is set to 1 when the *sensor_controlInit()* function is called and is set back 0 at the end of the program.

Warning: Don't forget that the parallel port is working at 5V and the MS5534 and MS5535 cannot work above 3.6V; levelshifter are therefore necessary.

A.1 Accessing the parallel port

Accessing directly the parallel port under Windows NT or 2000 is not possible since the operating system prevents applications to modify IO registers. In order to bypass this limitation, it is possible to use a device driver that will manage all access to the parallel port. The source code provided in this application note uses the NTPORT software developed by Zeal SoftStudio (http://www.zealsoftstudio.com/ntport). This piece of code allows any application to access the IO registers without restriction. A copy of the NTPORT evaluation software is included in the archive file. It is possible to purchase an unlimited version of NTPORT directly by Zeal SoftStudio.

To install the evaluation version of NTPORT on your computer, run *setup.exe* located in the *ntport_setup* directory.

A.3 Files provided in the archive

bcbport.lib	NTPORT library for Borland Builder
borlanddemo.bpf	Borland Builder project file
borlanddemo.bpr	Borland Builder project file
borlanddemo.exe	Executable generated by Borland Builder
hardware.cpp	hardware related functions (interface with NTPORT)
harware.h	hardware related functions
interface_ic.cpp	sensor interface ASIC functions
interface_ic.h	sensor interface ASIC functions
main.cpp	main
ms5534.cpp	MS5534 functions
ms5534.h	MS5534 functions
ms5535.cpp	MS5535 functions
ms5535.h	MS5535 functions
ntport.h	NTPORT interface functions
ntport.lib	NTPORT library for VC++
vcdemo.dsp	VC++ project file
vcdemo.dsw	VC++ project file
ntport_setup	directory containing NTPORT setup files

◆ FACTORY CONTACTS

Intersema Sensoric SA
Ch. Chapons-des-Prés 11
CH-2022 BEVAIX

SWITZERLAND

Tel. (032) 847 9550
Tel. Int. +41 32 847 9550
Telefax +41 32 847 9569
e-mail: knuman@intersema.ch
http://www.intersema.ch

LOCAL DISTRIBUTOR

Germany:

AMSYS GmbH
An der Fahrt 13
D-55124 Mainz
Tel. Int. +49 6131 46 98 75 55
Telefax +49 6131 46 98 75 66
http://www.amsys.de

USA:

Servoflo Cooperation
75 Allen Street
Lexington, MA 02421
Tel. Int. +1 781 862 9572
Telefax +1 781 862 9244
http://www.servoflo.com