AN502 Software library for the MS55xx sensors series

1 INTRODUCTION

Interfacing the MS55xx sensors with a computer or a microcontroller is very easy thanks to the serial digital interface that is provided by the pressure module. Components rely on the Sensor Interface ASIC for communication and data conversion.
This document provides C/C++ source code examples that access the sensor, 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.

2 COMMUNICATION WITH THE SENSOR INTERFACE ASIC

The MS55xx sensors series are 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. You find in this document some examples of source code for main types of MS55xx series. The ASIC access functions rely on a few hardware and time related functions and are thus platform dependent.

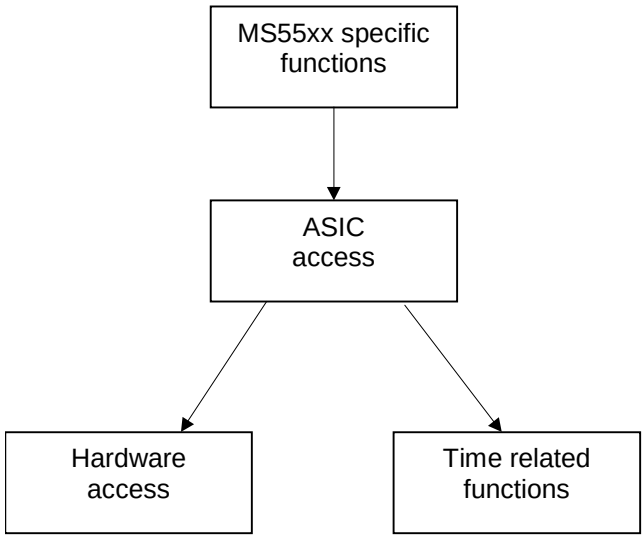

Figure 1. Software hierarchy

2.1 HARDWARE RELATED FUNCTIONS

The MS55xx 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:

```
/** initialize the microsystem */
void InitMicroSystem (void);

/** set SCLK to the specified state (0/1) */
void          setSCLK  (unsigned char state);

/** set DIN to the specified state (0/1) */
void          setDIN   (unsigned char state);

/** returns the current state of DOUT */
unsigned char getDOUT  (void);
```

hardware.h header file

The *InitMicroSystem()* functions initialize the hardware and makes it ready for operation.

- *setSCLK()* : sets the SCLK signal to the specified state
- *setDIN()* : sets the DIN signal to the specified state
- *getDOUT()* : returns the current state of the DOUT signal

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 MS55xx 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
```

hardware.h header file (continued)

2.3 SENSOR INTERFACE ASIC ACCESS FUNCTION

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.

```
/** send a reset sequence to the IC */
void           reset (void);

/**  Read the corresponding calibration word of the IC (index [1:4]) */
unsigned short getW  (unsigned char index);

/** Start a D1 acquisition, wait for end of conversion and return the value */
unsigned short getD1 (void);

/** Start a D2 acquisition, wait for end of conversion and return the value */
unsigned short getD2 (void);
```

interface_ic.h header file (continued)

The implementation of these functions is described in the interface_ic.c source file.

2.3.1 waitOnDoutFall

This function makes 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.

```
/* ================================================================== */
/*                          waitOnDoutFall                            */
/* ================================================================== */
void waitOnDoutFall(void)
{
    unsigned char working;

    working = 1;

    while(working)
        working = getDOUT();
}
```

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                               */
/* ================================================================== */
unsigned short SerialGet16(void)
{
    char           i;
    unsigned short v;

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

```
    for (i=0; i<16; i++)
    {
        setSCLK(1);
        WaitOnePulse();
        setSCLK(0);
        v = v << 1;
        if (getDOUT()==1)
            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(unsigned char pattern, unsigned char nbr_clock)
{
    unsigned char i;
    unsigned char c;

    setSCLK(0);
    WaitOnePulse();
    for (i=0; i<nbr_clock; i++)
    {
        c = pattern & 1;
        setDIN(c);
        WaitOnePulse();
        setSCLK(1);
        WaitOnePulse();
        setSCLK(0);
        pattern = 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                                   */
/* ====================================================================== */
unsigned short getW            (unsigned char index)  // 1 to 4
{
    unsigned short data;

    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                                     */
/* ======================================================================== */
unsigned short getD1            (void)
{
    unsigned short d1;

    SerialSendLsbFirst(0x2F, 8);
    SerialSendLsbFirst(0x00, 4);

    waitOnDoutFall();

    d1 = SerialGet16();

    SerialSendLsbFirst(0x00, 1);  // to be compliant with the data sheet
    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                                     */
/* ======================================================================== */
unsigned short getD2            (void)
{
    long d2;

    SerialSendLsbFirst(0x4F, 8);
    SerialSendLsbFirst(0x00, 4); // Note the difference with getD1

    waitOnDoutFall();

    d2 = SerialGet16();

    SerialSendLsbFirst(0x00, 1);  // to be compliant with the data sheet
    return(d2);
}
```

3 MS5534 / MS5540 SPECIFIC FUNCTIONS

The functions described here are specific for the MS5534 and MS5540. 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);
void calcPT5534  (double *pressure, double *temperature, long d1_arg, long d2_arg);
void calcPT2nd5534(double *pressure, double *temperature, long d1_arg, long d2_arg);
```

3.1 COEFFICIENTS CONVERSION

3.1.1 ConvertWtoC

This function 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);
}
```

3.2 PRESSURE AND TEMPERATURE CALCULATION

The following function makes the conversion from D1 & D2 to pressure and temperature. The result is stored in two global variables:

pressure
temperature

3.2.1 FIRST ORDER 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.

```
/* ------------------------------------------------------------------------ */
/* -------------------------- 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);
}
```

3.2.2 SECOND ORDER CALCULATION

```
/* ------------------------------------------------------------------------ */
/* -------------------------- calcPT2nd5534 ------------------------------- */
/* ------------------------------------------------------------------------ */
void calcPT2nd5534(double *pressure, double *temperature, long d1_arg, long d2_arg)
{
    double dt, off, sens;
    double fd1, fd2, x;
    double temp, temp2;
    double press, press2;

    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;

    press       =   2500 + x * 10 / 32.0;
```

```
    temp        =   200 + ( ( dt * ( fc[6] + 50.0 ) ) / 1024.0);

    temp2       = 0;
    press2       = 0;

    if(temp < 200)
    {
        temp2  = 11.0 * ( fc[6] +24 ) * ( 200 - temp ) * ( 200 - temp ) / 65536.0 /
16.0;
        press2 = 3.0 * temp2 * ( press - 3500 ) / 16384.0;
    }

    if(temp > 450)
    {
        temp2  = 3.0 * ( fc[6] + 24 ) * ( 450 - temp ) * ( 450 - temp ) / 655536.0 /
16.0;
    }

    temp  = temp - temp2;
    press = press - press2;

    if (pressure!=0)
        *pressure    =   press / 10.0;
    if (temperature!=0)
        *temperature =  temp / 10.0;
}
```

4 MS5535 / MS5541 SPECIFIC FUNCTIONS

The functions described here are specific for the MS5535 and MS5541. 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);
void calcPT5535  (double *pressure, double *temperature, long d1_arg, long d2_arg);
void calcPT2nd5535(double *pressure, double *temperature, long d1_arg, long d2_arg);
```

4.1 COEFFICIENTS CONVERSION

4.1.1 ConvertWtoC

This function 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);
}
```

4.2 PRESSURE AND TEMPERATURE CALCULATION

The following function makes the conversion from D1 & D2 to pressure and temperature. The result is stored in two global variables:

pressure
temperature

4.2.1 FIST ORDER 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.

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

    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;
}
```

4.2.2 SECOND ORDER CALCULATION

```
/* ------------------------------------------------------------------------ */
/* ---------------------------- calcPT5535 ---------------------------------- */
/* ------------------------------------------------------------------------ */
void calcPT2nd5535   (double *pressure, double *temperature, long d1_arg, long
d2_arg)
{
    double dt, off, sens;
    double fd1, fd2, x;
    double dt2;


    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 (temperature!=0)
    {
        if (dt<0)
            dt2 = dt - (dt/32768);
        else
            dt2 = dt - (dt/131072);

        *temperature =  ( 200   +    ((  dt2 * ( fc[6]+100.0) ) / 2048.0)) / 10.0;
        //*temperature = -50.0;
    }

    if (pressure!=0)
    {
        off      =  10000.0 +  fc[2]        + ( ( ( fc[4]-250.0) * dt2 ) / 4096.0);
        sens     =   3000.0 + (fc[1] / 2.0) + ( ( ( fc[3]+200.0) * dt2 ) / 8192.0);
        *pressure=   1000.0 +    (( sens * (fd1- off)) / 4096.0);
    }
}
```

5 MS5536 SPECIFIC FUNCTIONS

The functions described here are specific for the MS5536. They either process the data read from the sensor interface IC or send commands to it.

```
long ConvertWtoC5536 (int ix, long W1, long W2, long W3, long W4);
void calcPT5536  (double *pressure, double *temperature, long d1_arg, long d2_arg);
void calcPT2nd5536(double *pressure, double *temperature, long d1_arg, long d2_arg);
```

5.1 COEFFICIENTS CONVERSION

5.1.1 ConvertWtoC

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

```
/* ------------------------------------------------------------------------ */
/* -------------------------- ConvertWtoC5536 ----------------------------- */
/* ------------------------------------------------------------------------ */
long  ConvertWtoC5536 (int ix, long W1, long W2, long W3, long W4)
{
    long c;
    long x, y;


    W1 = W1 & 0xFFFF;
    W2 = W2 & 0xFFFF;
    W3 = W3 & 0xFFFF;
    W4 = W4 & 0xFFFF;

    c = 0;
    switch(ix)
    {
        case 1:
                    c =  W3 & 0x0FFF;
                    if (W4 & 0x8000)
                        c = c | 0x1000;
                    break;


        case 2:
                    c = W4 & 0x1FFF;
                    break;
        case 3:
                    c = (W1 >> 8) & 0xFF;
                    if (W4 & 0x2000)
                        c = c | 0x100;
                    break;
        case 4:
                    c = (W2 >> 8) & 0xFF;
                    if (W4 & 0x4000)
                        c = c | 0x100;
                    break;
        case 5:
                    x  = W1 & 0xFF;
                    y  = (W3>>4) & 0x0F00;
                    c  = x | y;
                    break;
        case 6:
                    c = W2 & 0x00FF;
                    break;
    }
    return(c);
}
```

5.2 PRESSURE AND TEMPERATURE CALCULATION

The following function makes the conversion from D1 & D2 to pressure and temperature. The result is stored in two global variables:

pressure
temperature

5.2.1 FIRST ORDER 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.

```
/* ------------------------------------------------------------------------ */
/* -------------------------- calcPT5536 ---------------------------------- */
/* ------------------------------------------------------------------------ */
void  calcPT5536(double *pressure, double *temperature, long d1_arg, long d2_arg)
{
    double dt, off, sens;
    double pressure_mmHg;
    double fd1, fd2, x;
    double ut1;

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

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

    ut1           =   4.0 * fc[5] + 15136.0;

    dt            =   fd2 - ut1;
    off           =   10381.0 + fc[2] + ( ( ( fc[4]-243.0) *  dt ) / 4096.0);
    sens          =   10179.0 + fc[1] + ( ( ( fc[3]+222.0) *  dt ) / 2048.0);
    x             =   ( sens * (fd1- off)) / 4096.0;
    pressure_mmHg = 0.02 * x;     // in mmHg

    if (pressure!=0)
        *pressure    =   pressure_mmHg;

    if (temperature!=0)
        *temperature =  20.0 +      ( (  dt * ( fc[6]+262.0) ) / 51200.0);
}
```

5.2.2 SECOND ORDER CALCULATION

```
/* ------------------------------------------------------------------------ */
/* -------------------------- calcPT5536 ---------------------------------- */
/* ------------------------------------------------------------------------ */
void  calcPT2nd5536(double *pressure, double *temperature, long d1_arg, long d2_arg)
{
    double dt, off, sens;
    double pressure_mmHg;
    double fd1, fd2, x;
    double ut1;

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

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

    ut1           =   4.0 * fc[5] + 15136.0;

    if (fd2>=ut1)
        dt  =  (fd2 - ut1) -      (fd2 - ut1)*(fd2 - ut1) / 262144.0;
    else
```

```
        dt  =  (fd2 - ut1) -  9.0*(fd2 - ut1)*(fd2 - ut1) / 262144.0;

    off           =   10381.0 +  fc[2] + ( (  ( fc[4]-243.0) *  dt ) / 4096.0);
    sens          =   10179.0 +  fc[1] + ( (  ( fc[3]+222.0) *  dt ) / 2048.0);
    x             =   ( sens * (fd1- off)) / 4096.0;
    pressure_mmHg = 0.02 * x;     // in mmHg

    if (pressure!=0)
        *pressure    =   pressure_mmHg;

    if (temperature!=0)
        *temperature = 20.0 +      ( (  dt * ( fc[6]+262.0) ) / 51200.0);
}
```

6 MS5536-60 SPECIFIC FUNCTIONS

The functions described here are specific for the MS5536-60. They either process the data read from the sensor interface IC or send commands to it.

```
long ConvertWtoC5536_60 (int ix, long W1, long W2, long W3, long W4);
void calcPT5536_60  (double *pressure, double *temperature, long d1_arg, long d2_arg);
void calcPT2nd5536_60(double *pressure, double *temperature, long d1_arg, long d2_arg);
```

6.1 COEFFICIENTS CONVERSION

6.1.1 ConvertWtoC

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

```
/* ------------------------------------------------------------------------ */
/* ------------------------- ConvertWtoC5536_60 --------------------------- */
/* ------------------------------------------------------------------------ */
long  ConvertWtoC5536_60 (int ix, long W1, long W2, long W3, long W4)
{
    long c;
    long x, y;

    W1 = W1 & 0xFFFF;
    W2 = W2 & 0xFFFF;
    W3 = W3 & 0xFFFF;
    W4 = W4 & 0xFFFF;

    c = 0;
    switch(ix)
    {
        case 1:
                    c = W1 & 0x03FF;
                    break;
        case 2:
                    c = W2 & 0x1FFF;
                    break;
        case 3:
                    x = (W1 >> 10) & 0x003F;
                    y = ((W2 >> 13) & 0x0007) << 6;
                    c = x | y;
                    break;
        case 4:
                    c = W3 & 0x03FF;
                    break;
```

```
        case 5:
                    x = (W3 >> 10 )& 0x003F;
                    y = ((W4 >> 10 ) & 0x003F) << 6;
                    c =  x | y;
                    break;
        case 6:
                    c = W4 & 0x03FF;
                    break;
    }
    return(c);
}
```

6.2 PRESSURE AND TEMPERATURE CALCULATION

The following function makes the conversion from D1 & D2 to pressure and temperature. The result is stored in two global variables:

pressure
temperature

6.2.1 FIRST ORDER 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.

```
/* ------------------------------------------------------------------------ */
/* --------------------------- calcPT5536_60 ------------------------------ */
/* ------------------------------------------------------------------------ */
void calcPT5536_60(double *pressure, double *temperature, long d1_arg, long d2_arg)
{
    double dt, off, sens;
    double pressure_mbar;
    double fd1, fd2;
    double ut1;

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

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

    ut1           = 4.0 * fc[5] + 18172.0;
    dt            = fd2 - ut1;
    off           = 10278.0 +  fc[2]        + ( ( ( fc[4]-465.0) *  dt ) / 8192.0);
    sens          = (1484.0 +  fc[1]) * 8.0 + ( ( ( fc[3]+259.0) *  dt ) / 2048.0);
    pressure_mbar = ( sens * (fd1- off)) / 4096.0;

    if (pressure!=0)
        *pressure    =   pressure_mbar / 100.0;

    if (temperature!=0)
        *temperature =  20.0 + ( (  dt * ( fc[6]+1034.0) ) / 204800.0);
}
```

REVISION HISTORY

Date	Revision	Type of changes
February 24, 2003	V1.1	Initial release
May 10, 2003	V2.0	This application note handles now only the MS5534
July 15, 2008	V3.0	Application note for MS55xx series
March 8, 2011	004	Change to MEAS logo and layout
June 20, 2011	005	Insertion of the mention TM in the logo, modification of Shenzhen zip code

FACTORY CONTACTS

NORTH AMERICA

Measurement Specialties
45738 Northport Loop West
Fremont, CA 94538
Tel: +1 800 767 1888
Fax: +1 510 498 1578

e-mail: pfg.cs.amer ch@meas-spec.com
Website: www.meas-spec.com

EUROPE

MEAS Switzerland Sàrl
Ch. Chapons-des-Prés 11
CH-2022 Bevaix
Tel: +41 32 847 9550
Fax: + 41 32 847 9569

e-mail: sales.ch@meas-spec.com
Website: www.meas-spec.com

ASIA

Measurement Specialties (China), Ltd.
No. 26 Langshan Road
Shenzhen High-Tech Park (North)
Nanshan District, Shenzhen, 518057
China
Tel: +86 755 3330 5088
Fax: +86 755 3330 5099
e-mail: pfg.cs.asia@meas-spec.com
Website: www.meas-spec.com