

HP03 programming guide

1. Brief description

The HP03 is a low power device, include a piezo-resistive pressure sensor and an ADC interface. It provide 16 bit data for pressure and temperature relate voltage 11 unique coefficients were stored on the chip, thus accurate pressure and temperature reading can be realized. IIC interface is used for communication with a microprocessor.

2. Read compensation coefficients and ADC value

1 Read compensation coefficients

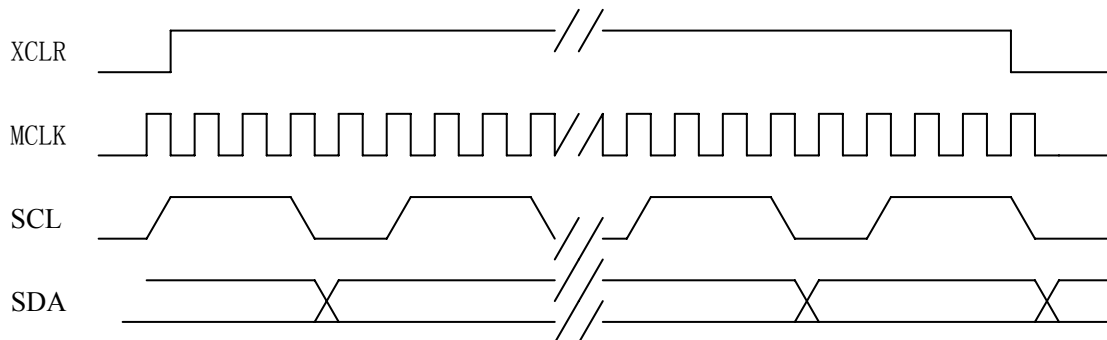
There are 11 compensation datum on the chip, the address of those coefficients are:

C1 (MSB : LSB)	(0x10 : 0x11)
C2 (MSB : LSB)	(0x12 : 0x13)
C3 (MSB : LSB)	(0x14 : 0x15)
C4 (MSB : LSB)	(0x16 : 0x17)
C5 (MSB : LSB)	(0x18 : 0x19)
C6 (MSB : LSB)	(0x1a : 0x1b)
C7 (MSB : LSB)	(0x1c : 0x1d)
A (MSB : LSB)	(0x1e)
B (MSB : LSB)	(0x1f)
C (MSB : LSB)	(0x20)
D (MSB : LSB)	(0x21)

User can read them as same as read AT24c02 chip,

2 Read AD value of the pressure and temperature

Timing diagram



In order to get the AD value, must be follow the following timing

Read pressure

S	0xEE	A	0xFF	A	0xF0	A	P	D	S	0xEE	A	0xFD	A	S	0xEF	A	MSB	A	LSB	N	P
---	------	---	------	---	------	---	---	---	---	------	---	------	---	---	------	---	-----	---	-----	---	---

Read temperature

S	0xEE	A	0xFF	A	0XD0	A	P	D	S	0xEE	A	0xFD	A	S	0xEF	A	MSB	A	LSB	N	P
---	------	---	------	---	------	---	---	---	---	------	---	------	---	---	------	---	-----	---	-----	---	---

S: start bit

P: stop bit

A: acknowledge from slave

A: acknowledge from master

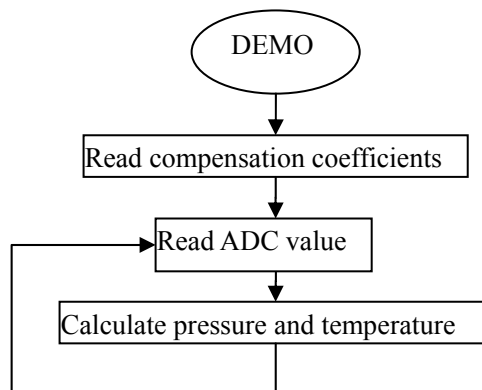
N: no acknowledge from master

D: delay for 40ms

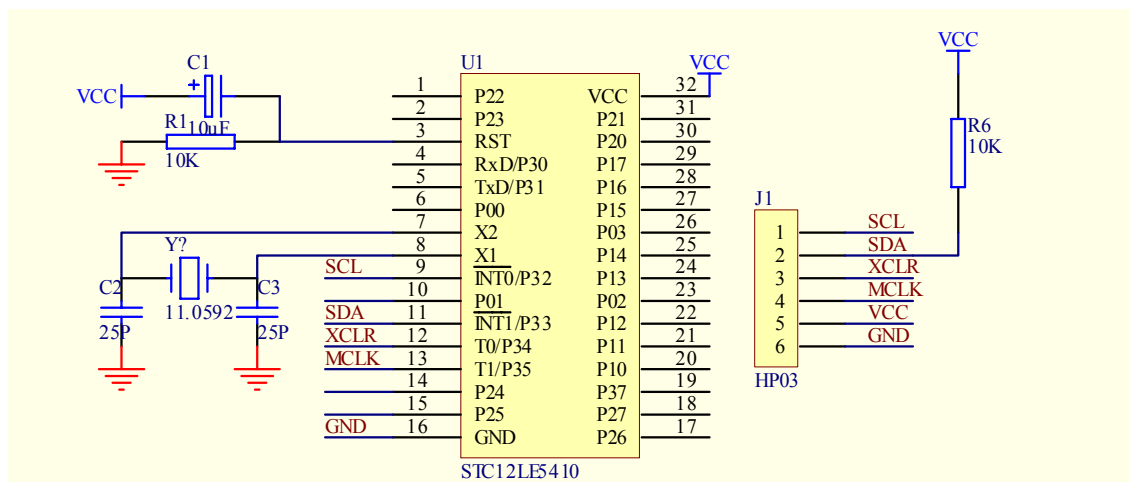
MSB: conversion result MSB

LSB: conversion result LSB.

3. Demo flow diagram



4. Example for standard 8051 core microcontroller



copyright (c) 2007

Title: HP03 simple example based on 8051 C
Current version: v1.0
Function: Calculate pressure ,altitude and temperature
Processor: STC12LE5410AD(8051 core)
Clock: 11.0592MHz Crystal
Author: Merrick
Company: Hope microelectronic Co.,Ltd.
Contact: +86-0755-86106557
E-MAIL: hopefsk@hoperf.com
Date: 2007-10-24

Connections

STC12LE5410AD SIDE	HP03 SIDE
P3.2----->	SCL
P3.3<----->	SDA
P3.4----->	XCLR
P3.5----->	MCLK

```
#include "reg52.h"  
#include <math.h>
```

```
#define MaxPress 1100 //1100Hpa  
#define DELAY10US 10
```

```
*****  
//function declaration  
*****  
void IIC_ReadCalData(void);  
void ReadTemperaturePreesureAD(void);  
unsigned int IIC_ReadTempretureAD(void);  
unsigned int IIC_ReadPressureAD(void);  
void CalculatePressTemp(void);  
void CalculateAltitude(void);
```

```
void IIC_ReadCalData(void);
unsigned char IIC_ReadByte(void);
void IIC_WriteByte( unsigned char);
void IIC_Start(void);
void IIC_Stop(void);
void IIC_ACK(void);
void IIC_NoAck(void);
void IIC_SDA_HIGH(void);
void IIC_SDA_LOW(void);
void IIC_SCL_HIGH(void);
void IIC_SCL_LOW(void);
void IIC_XCLR_LOW(void);
void IIC_XCLR_HIGH(void);
void SysDelay2ms(unsigned int);
void SysDelay(unsigned int);
```

```
/**
//symbol defined
**
```

```
sbit SCL = P3^2;
sbit SDA = P3^3;
sbit XCLR = P3^4;
sbit MCLK = P3^5;
```

```
unsigned int C1 ;
unsigned int C2 ;
unsigned int C3 ;
unsigned int C4 ;
unsigned int C5 ;
unsigned int C6 ;
unsigned int C7 ;
unsigned char AA ;
unsigned char BB ;
unsigned char CC ;
unsigned char DD ;
```

```
unsigned int D1 ;
unsigned int D2 ;
```

```
long float DUT;  
long float OFF;  
long float SENS;  
long float X;  
long float Press;  
long float Temp;  
long Altitude;
```

```
/**  
//comparison table for pressure and altitude  
/**
```

```
long code Tab_BasicAltitude[80]={-6983,-6201,-5413,-4620,-3820,-3015,-2203,-1385,-560, 270, //0.1m  
// 1100 1090 1080 1070 1060 1050 1040 1030 1020 1010 //hpa  
1108, 1953, 2805, 3664, 4224, 5403, 6284, 7172, 8068, 8972,  
// 1000 990 980 970 960 950 940 930 920 910 //hpa  
9885, 10805,11734,12671,13617,14572,15537,16510,17494,18486,  
// 900 890 880 870 860 850 840 830 820 810 //hpa  
19489,20502,21526,22560,23605,24662,25730,26809,27901,29005,  
// 800 790 780 770 760 750 740 730 720 710 //hpa  
30121,31251,32394,33551,34721,35906,37106,38322,39553,40800,  
// 700 690 680 670 660 650 640 630 620 610 //hpa  
42060,43345,44644,45961,47296,48652,50027,51423,52841,54281,  
// 600 590 580 570 560 550 540 530 520 510 //hpa  
55744,57231,58742,60280,61844,63436,65056,66707,68390,70105,  
// 500 490 480 470 460 450 440 430 420 410 //hpa  
71854,73639,75461,77323,79226,81172,83164,85204,87294,89438};  
// 400 390 380 370 360 350 340 330 320 310 //hpa
```

```
void main(void)
{
    SysDelay2ms(50);

    IIC_ReadCalData( );

    while(1)
    {
        ReadTemperaturePreesureAD( );
        CalculatePressTemp( );
        CalculateAltitude( );
        SysDelay2ms(300);
    }
}

//*****
//function: calculate power for 2
//*****
long int Get2_x(unsigned char i)
{
    long int uiData;
    uiData=2;
    i=i-1;
    while(i)
    {
        uiData <<= 1;
        i--;
    }
    return uiData;
}

//*****
//function:calculate press and temperature
//input   :D1,D2,C1---C7,AA,BB,CC,DD;
//output  :Press, unit: 0.01hpa
//        temp, unit: 0.1°C
//*****
void CalculatePressTemp(void)
{
    long float MiddleData1;
    long float MiddleData2;
    long float MiddleData3;
    long float MiddleData4;
```

```
//calculate the DUT value
if(D2<C5)
{
//DUT = D2-C5-((D2-C5)/Get2_x(7))*((D2-C5)/Get2_x(7))*BB/Get2_x(CC);
MiddleData1 = (long)D2-C5;
MiddleData2 = MiddleData1*MiddleData1/16384;
MiddleData3 = MiddleData2*BB;
MiddleData4 = Get2_x(CC);
MiddleData4 = MiddleData3/MiddleData4;
DUT = MiddleData1 - MiddleData4;
}
else
{
//DUT = D2-C5-((D2-C5)/Get2_x(7))*((D2-C5)/Get2_x(7))*AA/Get2_x(C);
MiddleData1 = D2-C5;
MiddleData2 = MiddleData1*MiddleData1/16384;
MiddleData3 = MiddleData2*AA;
MiddleData4 = Get2_x(CC);
MiddleData4 = MiddleData3/MiddleData4;
DUT = MiddleData1 - MiddleData4;
}

//calculate the OFF value
//OFF = (C2+(C4-1024)*DUT/Get2_x(14))*4;
MiddleData1 = (long)C4-1024;
MiddleData2 = Get2_x(14);
MiddleData3 = DUT*MiddleData1;
MiddleData4 = MiddleData3/MiddleData2;
MiddleData4 = (long)C2+MiddleData4;
OFF = MiddleData4*4;

//calculate the SENS value
//SENS = C1+C3*DUT/Get2_x(10);
MiddleData1 = (long)C3*DUT;
MiddleData2 = Get2_x(10);
MiddleData3 = MiddleData1/MiddleData2;
SENS = C1+MiddleData3;

//calculate the X value
//X = SENS*(D1-7168)/Get2_x(14)-OFF;
MiddleData1 = Get2_x(14);
MiddleData2 = (long)D1-7168;
MiddleData3 = MiddleData2*SENS;
```

```

MiddleData4 = MiddleData3/MiddleData1;
X = MiddleData4-OFF;

//calculate the Press value,have two decimal fraction
//Press = X*100/Get2_x(5)+C7*10;
MiddleData1 = X*100;
MiddleData2 = Get2_x(5);
MiddleData3 = MiddleData1/MiddleData2;
MiddleData4 = C7*10;
Press = MiddleData3+MiddleData4;

//calculate the Temperature value
Temp = 250+DUT*C6/Get2_x(16)-DUT/Get2_x(DD);

}

//*****
//function:calculate altitude
//input   :Press value
//output  :Altitude, unit: 0.1m
//*****

void CalculateAltitude(void)
{
char ucCount;
unsigned int  uiBasicPress;
unsigned int  uiBiasTotal;
unsigned int  uiBiasPress;
unsigned int  uiBiasAltitude;

for( ucCount=0;   ; ucCount++)
{
    uiBasicPress = MaxPress-(ucCount*10);
    if(uiBasicPress < (int)(Press/100))    break;
}

uiBiasTotal = Tab_BasicAltitude[ucCount] - Tab_BasicAltitude[ucCount-1];
uiBiasPress = Press - (long)(uiBasicPress*100);
uiBiasAltitude = (long)uiBiasTotal * uiBiasPress/1000;

Altitude = Tab_BasicAltitude[ucCount] - uiBiasAltitude;

ucCount = abs(Altitude % 10);    // four lose and five up
if(Altitude < 0)

```



```
{
    if(ucCount > 4)
        Altitude -= 10-ucCount;
    else
        Altitude += ucCount;
}

else
{
    if(ucCount > 4)
        Altitude += 10-ucCount;
    else
        Altitude -= ucCount;
}
}

//=====

void IIC_ReadCalData(void)
{
    unsigned char ucValue;

    IIC_Start();
    IIC_WriteByte(0xa0);
    IIC_WriteByte(16);
    IIC_Start();
    IIC_WriteByte(0xa1);
    ucValue = IIC_ReadByte();
    IIC_ACK();
    C1=ucValue;
    ucValue = IIC_ReadByte();
    IIC_ACK();
    C1 <<= 8;
    C1 |= ucValue;

    ucValue = IIC_ReadByte();
    IIC_ACK();
    C2=ucValue;
    ucValue = IIC_ReadByte();
    IIC_ACK();
    C2 <<= 8;
    C2 |= ucValue;

    ucValue = IIC_ReadByte();
```

```
IIC_ACK();
C3=ucValue;
ucValue = IIC_ReadByte();
IIC_ACK();
C3 <<= 8;
C3 |= ucValue;
```

```
ucValue = IIC_ReadByte();
IIC_ACK();
C4=ucValue;
ucValue = IIC_ReadByte();
IIC_ACK();
C4 <<= 8;
C4 |= ucValue;
```

```
ucValue = IIC_ReadByte();
IIC_ACK();
C5=ucValue;
ucValue = IIC_ReadByte();
IIC_ACK();
C5 <<= 8;
C5 |= ucValue;
```

```
ucValue = IIC_ReadByte();
IIC_ACK();
C6=ucValue;
ucValue = IIC_ReadByte();
IIC_ACK();
C6 <<= 8;
C6 |= ucValue;
```

```
ucValue = IIC_ReadByte();
IIC_ACK();
C7=ucValue;
ucValue = IIC_ReadByte();
IIC_ACK();
C7 <<= 8;
C7 |= ucValue;
```

```
ucValue = IIC_ReadByte();
IIC_ACK();
AA=ucValue;
ucValue = IIC_ReadByte();
IIC_ACK();
```

```
    BB= ucValue;

    ucValue = IIC_ReadByte();
    IIC_ACK();
    CC=ucValue;
    ucValue = IIC_ReadByte();
    IIC_NoAck();
    IIC_Stop();
    DD= ucValue;
}
//=====
void MCLKOn(void)
{
    TMOD = 0x12;
    TH0 = 0xf8;
    TL0 = 0xf8;
    ET0 = 1;
    EA = 1;
    PT0 = 1;
    TR0 = 1;

}
//=====
void MCLKOff(void)
{
    TR0 = 0;
    ET0 = 0;
}

//=====

void ReadTemperaturePressureAD(void)
{
    long uiSumADValue;
    uiSumADValue = 0;
    IIC_XCLR_HIGH();
    MCLKOn();
    SysDelay2ms(1);

    D1 = IIC_ReadPressureAD();
    uiSumADValue += D1;
    D1 = IIC_ReadPressureAD();
```

```
    uiSumADValue += D1;
    D1 = IIC_ReadPressureAD();
    uiSumADValue += D1;
    D1 = IIC_ReadPressureAD();
    uiSumADValue += D1;
    D1 = uiSumADValue >> 2;

    D2 = IIC_ReadTempretureAD();

    SDA=0;
    SCL=0;

    MCLKOff();
    IIC_XCLR_LOW();

}

//=====
void vect_Timer0(void) interrupt 1 using 1 //porduct 32k Hz signal
{
    MCLK = ~MCLK;
}
//=====

unsigned int IIC_ReadTempretureAD(void)
{
    unsigned char ucData;
    unsigned int wADT;

    IIC_Start();
    IIC_WriteByte(0xEE);
    IIC_WriteByte(0xFF);
    IIC_WriteByte(0xE8);
    IIC_Stop();
    SysDelay2ms(20);

    IIC_Start();
    IIC_WriteByte(0xEE);
    IIC_WriteByte(0xFD);
    IIC_Start();
    IIC_WriteByte(0xEF);

    ucData = IIC_ReadByte();
```

```
IIC_ACK();
// UartSend(CCC);
wADT = ucData;
wADT <<= 8;

ucData = IIC_ReadByte();
IIC_NoAck();
IIC_Stop();
wADT |= ucData;
// UartSend(DD);

return wADT;

}
//=====

unsigned int IIC_ReadPressureAD(void)
{
    unsigned char ucData;
    unsigned int wADp;

    IIC_Start();
    IIC_WriteByte(0xEE);
    IIC_WriteByte(0xFF);
    IIC_WriteByte(0xF0);
    IIC_Stop();
    SysDelay2ms(20);

    IIC_Start();
    IIC_WriteByte(0xEE);
    IIC_WriteByte(0xFD);
    IIC_Start();
    IIC_WriteByte(0xEF);

    ucData = IIC_ReadByte();
    IIC_ACK();
    wADp = ucData ;
    wADp <<=8;
// UartSend(AA);

    ucData = IIC_ReadByte();
    IIC_NoAck();
```

```
IIC_Stop();
wADp |= ucData;
// UartSend(BB);

return wADp;
}

//=====

unsigned char IIC_ReadByte(void)
{
    unsigned char ucValue;
    unsigned char ucIndex;

    IIC_SDA_HIGH();
    SysDelay(DELAY10US);
    for ( ucIndex = 0; ucIndex < 8; ucIndex++ )
    {
        ucValue <<= 1;

        IIC_SCL_LOW();
        SysDelay(DELAY10US);

        IIC_SCL_HIGH();
        SysDelay(DELAY10US);

        if(SDA)
            ucValue |= 1;

        SysDelay(DELAY10US);
        IIC_SCL_LOW();
        SysDelay(DELAY10US);

    }

    return ucValue;
}

//=====

void IIC_WriteByte( unsigned char ucData )
{
```

```
unsigned char i;
for( i = 0; i < 8; i++)
{
    IIC_SCL_LOW();
    SysDelay(DELAY10US);

    if((ucData & 0x80) == 0x80)
    {
        IIC_SDA_HIGH();
        SysDelay(DELAY10US);
    }
    else
    {
        IIC_SDA_LOW();
        SysDelay(DELAY10US);
    }

    IIC_SCL_HIGH();
    SysDelay(DELAY10US);
    ucData <<= 1;
    IIC_SCL_LOW();
}

IIC_SDA_HIGH();
SysDelay(DELAY10US);
IIC_SCL_LOW();
SysDelay(DELAY10US);
IIC_SCL_HIGH();
SysDelay(DELAY10US);
IIC_SCL_LOW();
SysDelay(DELAY10US);

}
//=====

void IIC_Start(void)
{

    IIC_SDA_HIGH();
    SysDelay(DELAY10US);

    IIC_SCL_HIGH();
```

```
    SysDelay(DELAY10US);

    IIC_SDA_LOW();
    SysDelay(DELAY10US);

    IIC_SCL_LOW();
    SysDelay(DELAY10US);

}
//=====

void IIC_Stop(void)
{
    IIC_SCL_LOW();
    SysDelay(DELAY10US);

    IIC_SDA_LOW();
    SysDelay(DELAY10US);

    IIC_SCL_HIGH();
    SysDelay(DELAY10US);

    IIC_SDA_HIGH();
    SysDelay(DELAY10US);
}
//=====

void IIC_ACK(void)
{
    IIC_SDA_LOW();
    SysDelay(DELAY10US);

    IIC_SCL_HIGH();
    SysDelay(DELAY10US);

    IIC_SCL_LOW();
    SysDelay(DELAY10US);
}
//=====

void IIC_NoAck(void)
{
```



```
IIC_SDA_HIGH();  
SysDelay(DELAY10US);
```

```
IIC_SCL_HIGH();  
SysDelay(DELAY10US);
```

```
IIC_SCL_LOW();  
SysDelay(DELAY10US);
```

```
}
```

```
//=====
```

```
void IIC_SDA_HIGH(void)  
{  
    SDA=1;  
}
```

```
//=====
```

```
void IIC_SDA_LOW(void)  
{  
    SDA=0;  
}
```

```
//=====
```

```
void IIC_SCL_HIGH(void)  
{  
    SCL=1;  
}
```

```
//=====
```

```
void IIC_SCL_LOW(void)  
{  
    SCL=0;  
}
```

```
//=====
```

```
void IIC_XCLR_LOW(void)
{
    XCLR=0;
}

//=====

void IIC_XCLR_HIGH(void)
{
    XCLR=1;
}

//=====

void SysDelay2ms( unsigned int t)
{
    unsigned int i;
    while(t--)
        {for (i = 0; i < 1250; i++);
         for (i = 0; i < 1500; i++);
         }
}

//=====

void SysDelay(unsigned int t)
{
    while(t--);
}
```

Notice:

1. Supply voltage : 2.7V---3.3V
2. MCLK frequency : 30K---35K。 The up edge and down edge should be steep, the edge more steeper the power more lower.
3. Don't be static electricity destroy ,don't operation with power.
4. With constant temperature to solder.
5. After solder, wait for some time ,let temperature tranquilization. For the best wait for 24 hours.

HOPE MICROELECTRONICS CO.,LTD

4/F, Block B3, East Industrial Area,
Huaqiaocheng, Shenzhen, Guangdong,
China

Tel: 86-755-86096602

Fax: 86-755-86096587

Email: sales@hoperf.com

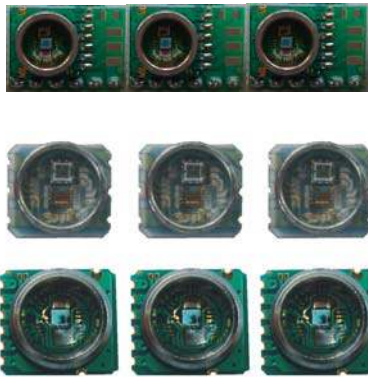
Website: <http://www.hoperf.com>

<http://www.hoperf.cn>

<http://hoperf.en.alibaba.com>

This document may contain preliminary information and is subject to change by Hope Microelectronics without notice. Hope Microelectronics assumes no responsibility or liability for any use of the information contained herein. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of Hope Microelectronics or third parties. The products described in this document are not intended for use in implantation or other direct life support applications where malfunction may result in the direct physical harm or injury to persons. NO WARRANTIES OF ANY KIND, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, ARE OFFERED IN THIS DOCUMENT.

©2006, HOPE MICROELECTRONICS CO.,LTD. All rights reserved.



- . Integrated pressure sensor
- . 300-1100hpa absolute Pressure Range
- . 16 Bit Σ - Δ ADC
- . 11 coefficients for software compensation stored on chip
- . I²C Serial Interface
- . One system clock line (32768Hz)
- . One hardware controlled reset line
- . Low voltage, low power consumption

Description

The HP03S pressure module includes a piezo-resistive pressure sensor and an ADC interface. It provides 16 bit word data for pressure and temperature related voltage. With the help of a highly accurate calibration of the sensor, 11 unique coefficients were stored on the chip, thus accurate pressure and temperature reading can be realized. HP03S is a low power, low voltage device with automatic power down switching. I²C Serial Interface is used for communications with a microprocessor. Sensor packaging options are DIP or SMD (with metal cap)

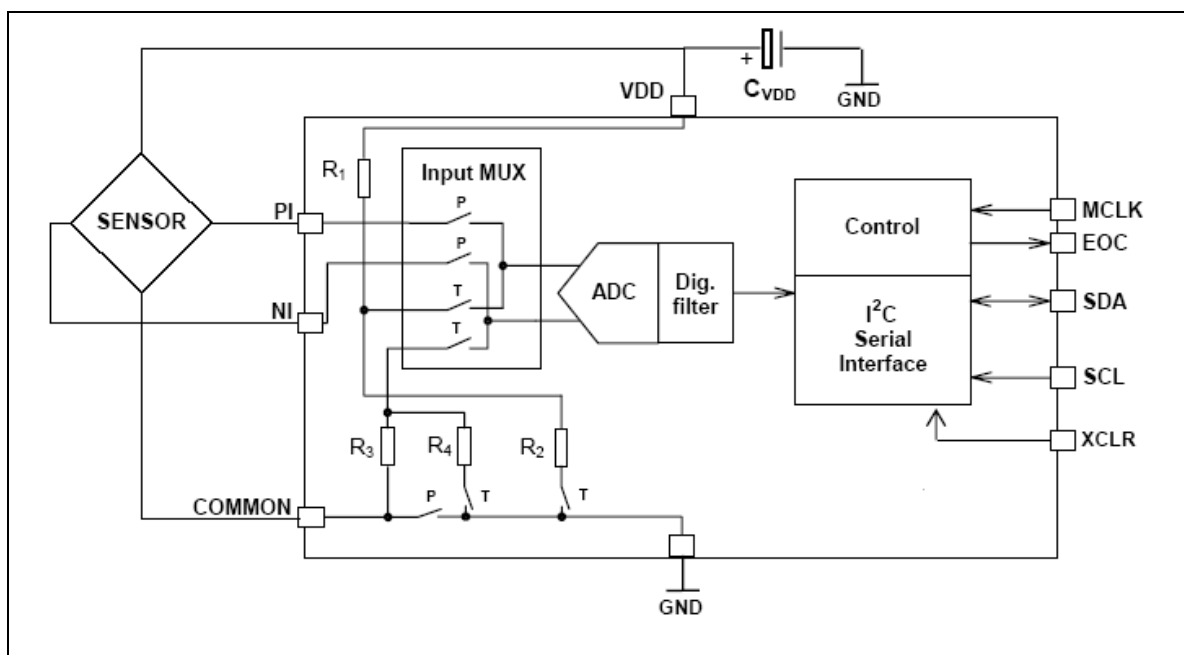
Features

- . 14 Bit ADC resolution
- . Supply voltage 2.2v-3.6v
- . -40°C to + 85°C operating range
- . No external components required

Applications

- . Pressure measurement and control systems
- . Mobile altimeter/barometer systems
- . Weather forecast products
- . Adventure or multi-mode watches
- . GPS receivers

Block Diagram



PIN Description

Pin Name	Pin Number	Type	Function
VSS	6	G	power ground
VDD	5	P	power VCC
MCLK	4	I	master clock(32k) input
XCLR	3	I	ADC reset input (keep low when system is in idle state)
SDA	2	I/O	I ² C data input and output
SCL	1	I	I ² C clock input

* XCLR is to reset the AD converter (active low). XCLR should be set to high only during AD conversion phase(reading D1,D2), at all other states, such as reading calibration factors, this pin should be kept low.

* The quality of the MCLK signal can significantly influence the current consumption of the pressure module. To obtain minimum current, remember to supply good quality MCLK signal

Absolute Maximum Ratings

Parameter	Symbol	Min	Max	Unit
Supply Voltage	VDD	-0.3	4	V
Over pressure	P		15	Bar(abs)
Storage Temperature	Tstg	-40	125	°C

Recommended Operating Conditions

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Supply Voltage	VDD		2.2	3	3.6	V
Supply Current	I	VDD=3V				V
during conversion				500		μA
Stand by				1		μA
Operating Pressure Range	P		300		1100	hpa (abs)
Operating Temperature Range	T		-40	25	85	°C
MCLK	T		30	32768	35	KHz
Duty Cycle of MCLK			40%	50%	60%	%
Serial Date Rate	SCL				500	KHz

Pressure and Temperature Output Characteristics

With the calibration data provided by the HP03 system, it should be able to reach the following characteristics:

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Absolute Pressure Accuracy	HP03SA	750-1100 @ 0~50	-1.5		1.5	hpa
Absolute Pressure Accuracy	HP03SB	750-1100 @ 0~50	-3.0		3.0	hpa
Absolute Pressure Accuracy	HP03SA	750-1100 @ -20~60	-3.0		3.0	hpa
Absolute Pressure Accuracy	HP03SB	750-1100 @ -20~60	-5.0		5.0	hpa
Long Term Stability		12 month		2		hpa

HP03S

2010-2-3

Version: 1.3

VDD Dependency		2.4~3.6	-1.5	0	1.5	hpa
Temperature Accuracy		0~+50	-1.0		1.0	°C
Temperature Accuracy		-20~+60	-2.0		2.0	°C

Pressure and Temperature Measurement

The main function of HP03 system is to convert the uncompensated pressure and temperature signal from a pressure sensor. After the conversion, the following two values can be obtained:

- . measured temperature “D2”
- . measured pressure “D1”

As the sensor is strongly temperature dependent, it is necessary to compensate for these effects. Therefore 10 sensor-specific coefficients are stored on the HP03 at our manufacturing facility, and they allow an accurate software compensation in the application.

The 7 coefficients are:

- . Sensitivity coefficient “C1”
- . Offset coefficient “C2”
- . Temperature Coefficient of Sensitivity “C3”
- . Temperature Coefficient of Offset “C4”
- . Reference Temperature “C5”
- . Temperature Coefficient of Temperature “C6”
- . Offset Fine Tuning “C7”

4 sensor parameter

- . Sensor Specific Parameter “A,B,C,D”

Note: Make sure to pull low XCLR before start to Read these coefficients or the data read out is probably incorrect

Parameter		Range (Hex:Dec)	
C1	0x100 -- 0xFFFF	:	256 -- 65535
C2	0x00 -- 0x1FFF	;	0 -- 8191
C3	0x00 -- 0x400	;	0 -- 3000
C4	0x00 -- 0x1000	;	0 -- 4096
C5	0x1000 -- 0xFFFF	;	4096 -- 65535
C6	0x00 -- 0x4000	;	0 -- 16384
C7	0x960 -- 0xA28	;	2400 -- 2600
C, D	0x01 -- 0x0F	;	1 -- 15
A, B	0x01 -- 0x3F	;	1 -- 63
D1	0x00 -- 0xFFFF	;	0 -- 65535
D2	0x00 -- 0xFFFF	;	0 -- 65535

Pressure and Temperature Calculation:

Step 1: (get temperature value)

$D2 \geq C5 \quad dUT = D2 - C5 - ((D2 - C5) / 2^7) * ((D2 - C5) / 2^7) * A / 2^C$ $D2 < C5 \quad dUT = D2 - C5 - ((D2 - C5) / 2^7) * ((D2 - C5) / 2^7) * B / 2^C$
--

Step 2: (calculate offset, sensitivity and final pressure value)

$OFF = (C2 + (C4 - 1024) * dUT / 2^{14}) * 4$
$SENS = C1 + C3 * dUT / 2^{10}$
$X = SENS * (D1 - 7168) / 2^{14} - OFF$
$P = X * 10 / 2^5 + C7$

- For altitude measurement system, recommend to use $P = X * 100 / 2^5 + C7 * 10$
- So that better altitude resolution can be achieved

Step 3: (calculate temperature)

$T = 250 + dUT * C6 / 2^{16} - dUT / 2^D$

Example:

C1=29908
 C2=3724
 C3=312
 C4=441
 C5=9191
 C6=3990
 C7=2500
 A=1
 B=4
 C=4
 D=9

D1=30036
 D2=4107

$dUT = (4107 - 9191) - ((4107 - 9191) * (4107 - 9191) / 128^2) * 4 / 2^4 = -5478$

$OFF = (3724 + (441 - 1024) * (-5478) / 2^{14}) * 4 = 15675$

$SENS = 29908 + 312 * (-5478) / 2^{10} = 28238$

$X = 28238 * (30036 - 7168) / 2^{14} - 15675 = 23738$

HP03S

2010-2-3

Version: 1.3

$$P = 23738 * 10 / 2^5 + 2500 = 9918 = 991.8\text{hpa}$$

$$T = 250 + (-5478) * 3990 / 2^16 - (-5478/2^9) = -72 = -7.2^{\circ}\text{C}$$

Serial Interface

The I²C interface is used for accessing calibration data as well as reading measurement result from AD conversion.

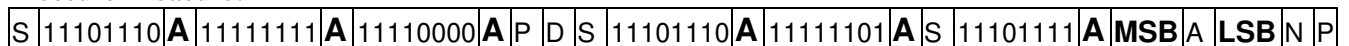
The EEPROM and ADC is sharing the same I²C bus but with different chip address assigned. The EEPROM chip address is set to 0xA1(in the case of read), write operation is not allowed. For AD part, the chip address is set to 0xEE. So this module used two different addresses for calibration data and AD converting data accessing. Calibration EEPROM data read operation is fully compatible to 24C02. Bus drive timing should be referred to the specification of this part as well.

Coefficient EEPROM ADDRESS

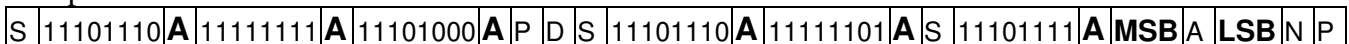
C1(MSB:LSB)	(16:17)
C2(MSB:LSB)	(18:19)
C3(MSB:LSB)	(20:21)
C4(MSB:LSB)	(22:23)
C5(MSB:LSB)	(24:25)
C6(MSB:LSB)	(26:27)
C7(MSB:LSB)	(28:29)
A	(30)
B	(31)
C	(32)
D	(33)

AD chip address is set to 0xEE(device write address), 0xEF(device read address). In order to get the AD value D1 and D2, you have to follow the following timing sequence:

Pressure Measure:



Temperature Measure:



- S: start condition
- P: stop condition
- A** (bold) : acknowledge from slave
- A : acknowledge from master
- N: no acknowledge from master (send out bit 1 instead)
- D : delay for 40ms minimum
- MSB: conversion result MSB
- LSB: conversion result LSB.

Remark:

Before start an AD conversion cycle, remember to pull high for XCLR pin so that the system is no longer in the reset state.

HP03S

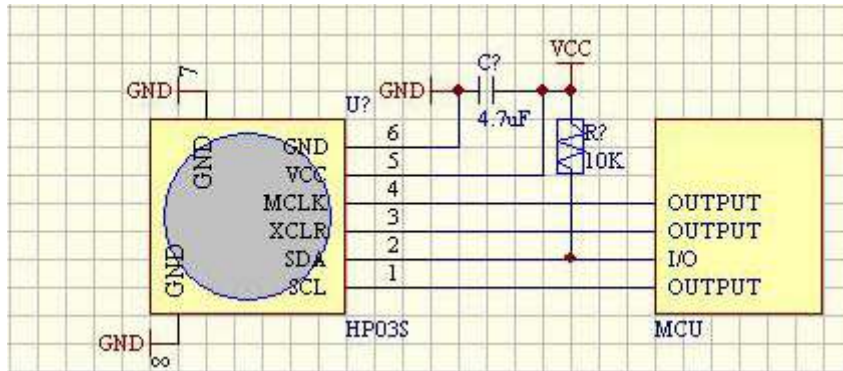
2010-2-3

Version: 1.3

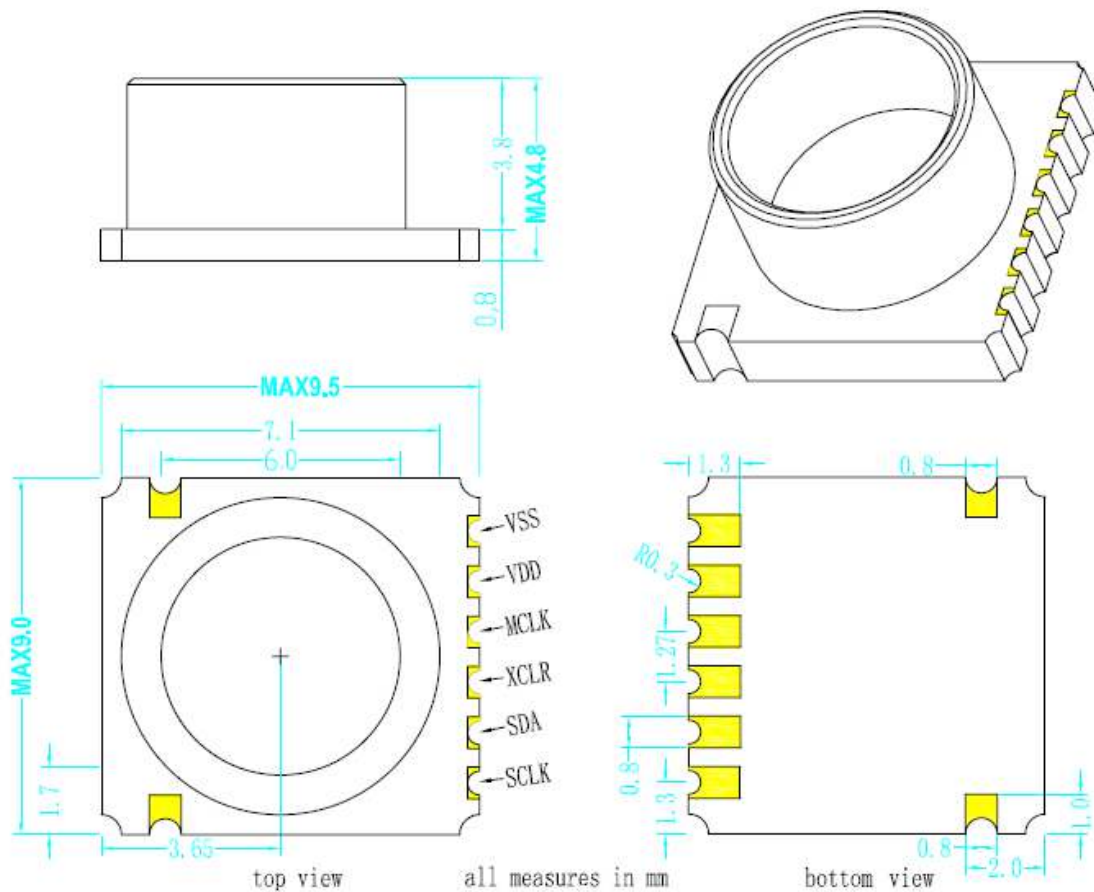
All data read from the module is in hex format.

After first power on, the first read data should be disregarded, and only the second value should be used. This can assure that any unstable data after reset can be filtered out.

Typical Application Circuit Diagram:



Mechanical Dimension



Important Notices

Never unplug the module when power is on.

Do not use this product as safety or emergency stop device or in any application where failure of this product could lead in personal injury. Failure to comply with these instructions could result with death or serious injury.

Should buyer purchase or use HOPE RF products for any such unintended or unauthorized application, buyer should indemnify and hold HOPE RF and its officers, employees, affiliates and distributors harmless against all claims, costs, damages and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury associated with such unintended or unauthorized use, even if such claim alleges that HOPE RF was negligent regarding the design or manufacturing of the part.

Hope RF reserves the right, without further notice, to change the product specification and/or information in this document and to improve reliability, functions and design.