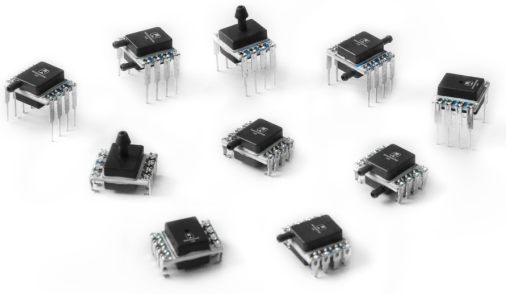
WPS08/04高分辨率高精度系列压力传感器

产品介绍

WPS08/04 高分辨率高精度系列是由深圳市沃感科技有限公司自主研发的压阻式高精度高性能硅陶瓷压力传感器，内置24位高分辨率ASIC 对传感器的偏移、温度效应、灵敏度及非线性度进行校准和温度补偿，可在多种压力测量范围和温度范围提供IIC通信数字信号输出。此系列传感器可用于差压、表压和绝压的测量,可直接安装在标准印刷电路板上使用。此系列压力传感器适用于无腐蚀性、非离子气体（例如空气和其它干燥气体）。所有产品遵循 ISO9001标准设计制造并经过严格的生产校准，出厂检验确保产品的一致性和可靠性。



特点

- IIC通信数字信号输出
- 补偿温度：-40~+85℃
- 倒钩状压力接口或无端口
- 精度：±0.1%FSS
- 绝压、差压和表压类型
- 分辨率： 24 位
- 总误差带（TEB）：最好±0.5%
- 供电方式：3.3V 供电

应用

工业自动化　泄露测试　医疗器械　风道静压　楼宇空调　肺活量计

产品规格

额定参数

参数	最小值	最大值	单位
电源电压	-0.3	3.6	Vdc
数字接口时钟频率:	100	400	kHz
ESD 敏感度（人体模式）		3	kV
存储温度	-40	125	°C
过载压力	2 倍满量程		
爆破压力	3 倍满量程		
焊接时间及温度	波峰焊	250 °C 条件下最长 15 秒	
	回流焊	250 °C 条件下最长 4 秒	

深圳市光明区玉塘街道同观路十九号路10号九洲工业区1栋502
TEL:+86-755-83439588 FAX:+86-755-83439588
WEB:www.woosens.com

工作参数

参数	最小值	典型值	最大值	单位
电源电压VDD：	3.0	3.3	3.6	Vdc
电源电流：	-	-	2.0	mA
待机电流（25°C ）		-	0.1	mA
补偿温度范围	-40	-	85	°C
启动时间（从加电到数据准备就绪）	-	2.5	-	ms
低电平电压	-	-	0.2	V_{DD}
高电平电压	0.8	-	-	V_{DD}
负载电阻	1	5	50	kOhm
精度	-0.1	-	0.1	%FSS BFSL
分辨率	-	24	-	位
默认通讯地址		0X78		

注意：

- 该传感器没有反向极性保护。将错误的引脚与电源连接或者接地可能会导致电气故障。
- 补偿温度范围是指传感器可以在特定的性能限制下产生与压力成比例的输出的温度范围。
- 工作温度范围是指传感器可以产生与压力成比例的输出的温度范围，但不一定在特定性能限制范围之内。
- 精度：相对适用于在 25°C 时的压力范围内所测输出的最佳直线 (BFSL) 的最大输出偏差。包括所有因压力非线性、压力迟滞性和不可重复性造成的误差。
- 总误差带（TEB)：相对整个补偿温度和压力范围内理想传递函数的最大偏差。包括所有因零点、满量程、压力非线性、压力迟滞性、不可重复性、热零点偏移、热量程偏移和热迟滞性造成的误差。
- 满量程 (FSS) 是指在压力范围最大限制值 (Pmax.) 和最小限制值 (Pmin.) 处测得的输出信号之间的代数差。
- 过载压力：可安全施加到产品的最大压力，使产品在压力返回到工作压力范围时规格保持不变。施加过高的压力可能会对产品造成永久损坏。除非另有规定，否则这适用于工作温度范围内任何温度下的所有可用压力口。
- 爆破压力：可施加到产品的任意压力口而不造成压力媒介脱离的最大压力。在经受任何超过爆破压力的压力之后， 产品将不能正常工作。

环境规格

参数	特性
湿度： 仅干燥气体	0% 到 95% RH
寿命	至少为 100 万次循环

- 寿命可能因传感器使用的特定应用而有所变化。

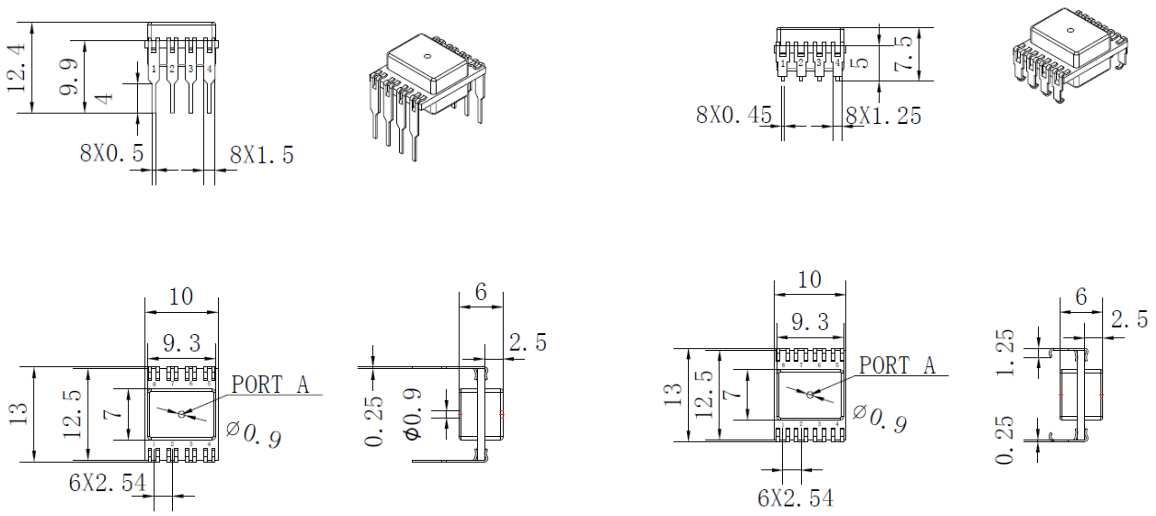
脚位定义

输出类型	引脚 1	引脚 2	引脚 3	引脚 4	引脚 5	引脚 6	引脚 7	引脚 8
I2C	GND	V_{DD}	SDA	SCL	NC	NC	NC	NC

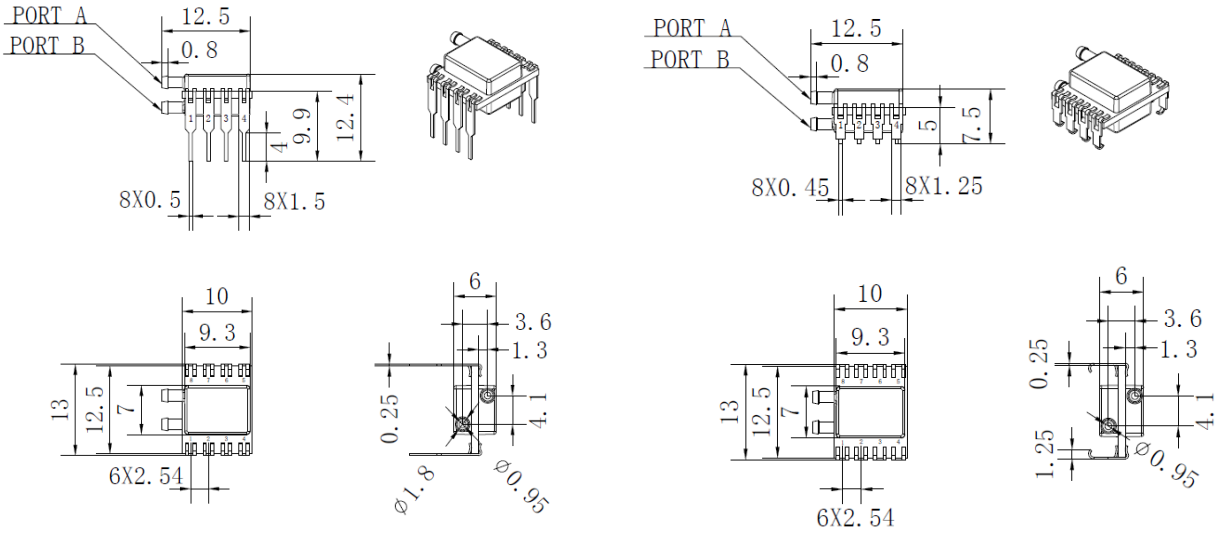
结构参数

单位：mm

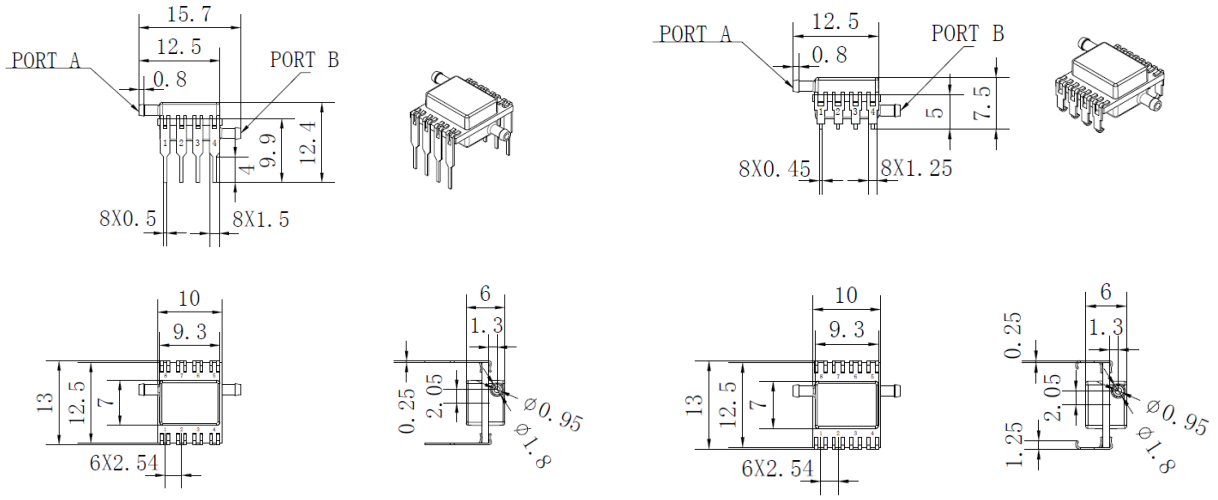
NN：无端口



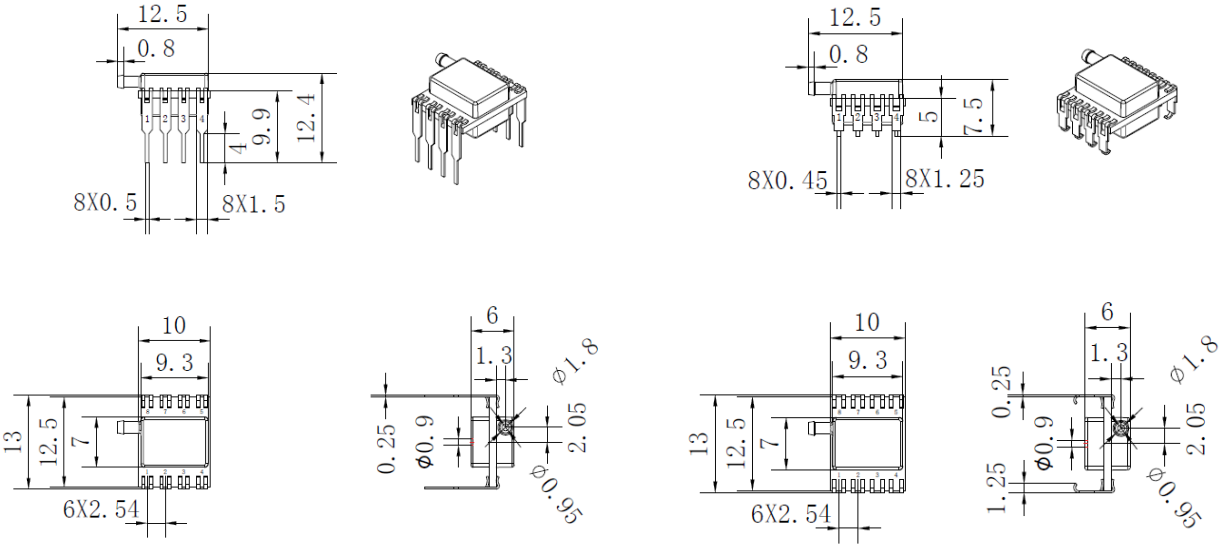
RR：双径向带刺端口，同侧



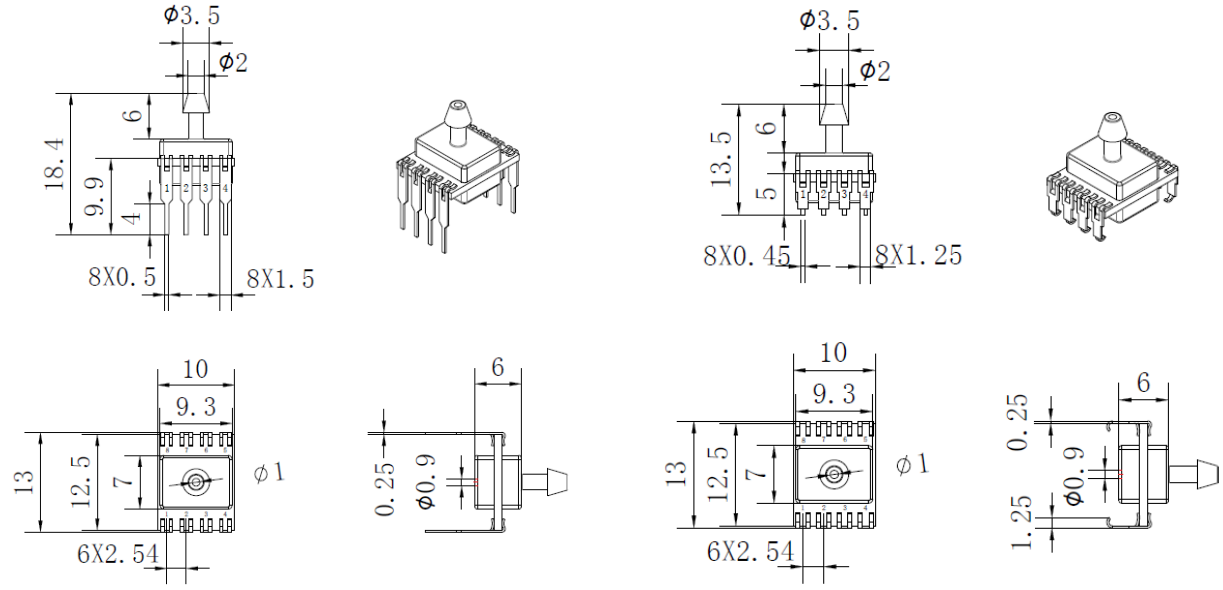
DR：双径向带刺端口，对侧



RN：单径向带刺端口



AN：单轴向带刺端口



订购信息

例如: WPS 08D 006K G 05 K NN 3 H

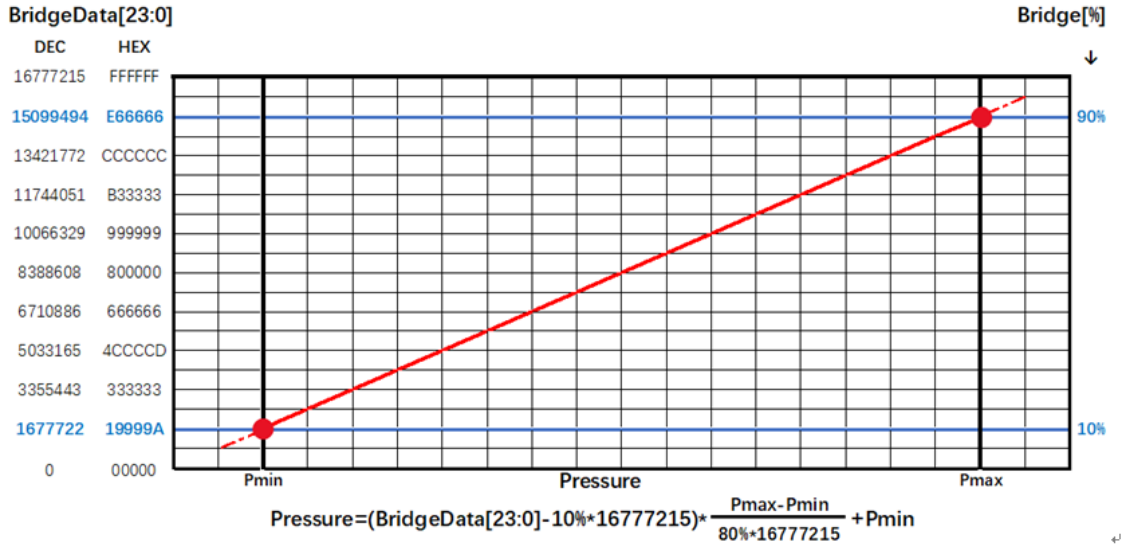
(1) (2) (3) (4) (5) (6) (7) (8) (9)

序号	具体意义	详细描述
1	产品识别码	WPS
2	封装	08 D: 8 PIN DIP（双列直插） 08 M: 8 PIN SMT（表面贴装）
3	压力范围	004K(4KPA) 006K(6KPA) 010K(10KPA) 016K(16KPA) 025K(25KPA) 040K(40KPA) 050K(50KPA) 060K(60KPA) 100K(100KPA) 160K(160KPA) 250K(250KPA) 400K(400KPA) 600K(600KPA) 700K(700KPA) 001G(1MPA) 注：当产品的量程在此表内未体现出来时按以下示例进行扩充 KPA 的产品均为---K 示例：005K 代表5KPA PA 的产品均 为---P 示例：100P 代表 100PA
4	压力类型	G：表压 D：差压 A：绝压
5	TEB范围	05：±0.5%FS 07：±0.75%FS 10：±1.0%FS 20：±2.0%FS
6	输出类型	K： I2C输出24位
7	压力端口	NN：无端口 AN：单轴向带刺端口 RN：单径向带刺端口 RR：双径向带刺端口，同侧 DR：双径向带刺端口，对侧
8	供电电压	3：3.3VDC
9	补偿温度	E： -40 C°~85C° H： -20 C°~85C° M： 0 C°~85C° L： 0 C°~50C°

压力和温度输出对应公式

I2C 输出（分辨率24位）

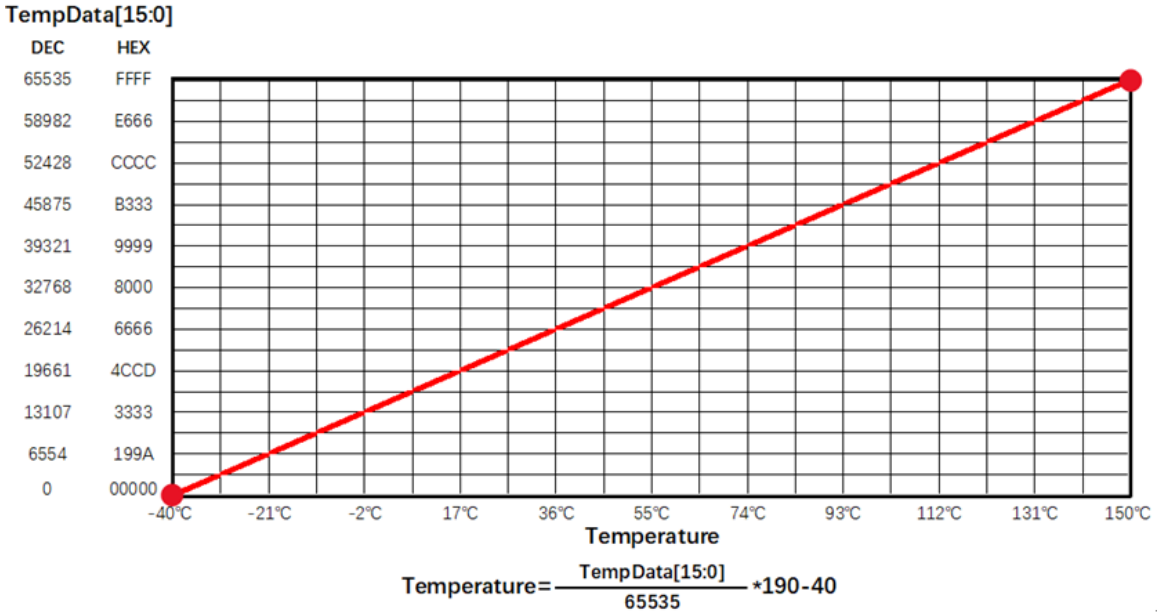
压力转换方程



高百分比的传感器输出

输出百分比	数字计数（十进制）
0	0
10	1677722
50	8388608
90	15099494
100	16777216

温度转换方程



I2C 输出（24位分辨率）

使用 0xAC 命令利用传感器内部校准算法获取校准值的说明

发送 0xAC 命令获取校准值的步骤如下：

一、发写命令

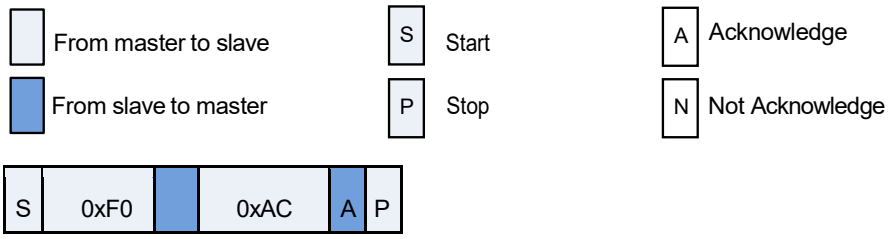


图 1 写命令

写命令中的 0xF0 表示默认的 7bits I2C 设备地址为 0x78，最后 1bit 为 0 表示主设备写操作。

二、等待

发送完写命令后需要等待一段时间再发送读命令，因为内部完成整个测量需要一段时间。等待的时长取决于OTP(Address: 0x14)的[13:11]电桥过采样率和 OTP(Address: 0x14)的[15:14]温度过采样率的设置。对照附录的表 1 和表 2，等待的时长=tP+tT。 等待的时长不需要计算，可以通过不断的读 IIC 状态字的方式判断出是否采集已经完成。

三、读数

要保证写指令和读指令的时间间隔大于测量的时长才能够读出校准数据，读数格式如图 2 所示，读命令中的 0xF1 表示默认的 7bits I2C 设备地址为 0x78， 最后 1bit 为 1 表示主设备读操作。读到的校准数据共 6 个字节，依次为 1 字节状态字，3 字节电桥校准值，2 字节温度校准值。

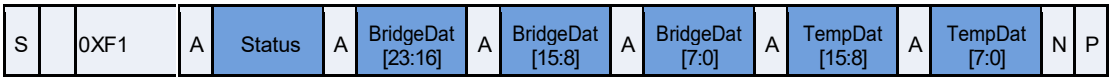


图 2 I2C 读出 5 字节校准后的压力和温度值

四、换算

读到校准数据后，需要将以 AD 值形式表示的无符号数迆行简单的换算。 为方便理解我们假设读到的校准数据为：

0x04 0x9B 0xB0 0xC5 0x56 0xAA

0x04 为状态字 Bit5 为 1 表明最近一次 I2C 忙，需要等待一段时间。如果 Bit5 为 0 表明设备非忙，可以读取数据。关于状态字各比特的详细描述请参见附录。

0x9B 0xB0 0xC5 三个字节为电桥校准值

0x56 0xAA 两个字节为温度校准值

电桥校准值换算：将 0x9B 0xB0 0xC5 转换为十迆制数为 10203333,

本次计算假设校准时使用的量程为 20Kpa-120Kpa，对应的 AD 输出为 1677722~15099494（10%AD~90%AD）

根据输入输出关系校准公式得到：

实际压力值=（120-20）/（15099494-1677722）*（10203333-1677722）+20=83.5208 Kpa

温度校准值换算：将 0x56 0xAA 转换为十进制数为 22186，由于读取到的校准数据是以百分比形式表示的，这个百分比在数值上等于我们换算得到的十进制数与16bits 无符号数的最大值（65535）之比，所以在换算百分比时可进行如下计算

22186/65536*100%=33.85%

温度的校准范围规定为-40℃—150℃ 所以校准值=（150—（-40））*33.85%—40=24.32℃

附录

表 1 压力过采样率和测量时间对照表

OSR_Pressure[13:11]（二进制）	对应的过采样率	测量时间 tP(ms)
000	32768	203
001	16384	105
010	8192	56
011	4096	31
100	2048	19
101	1024	13
110	512	10

表 2 温度过采样率和测量时间对照表

OSR_Temperature[15:14]（二进制）	对应的过采样率	测量时间 tT(ms)
00	2048	19
01	4096	31
10	8192	56
11	16384	105

表 3 Status 字节比特位描述

比特位	意 义	描 述
Bit7	保留	固定为 0
Bit6	上电指示 (Power indication)	1 设备上电（VDDB on）；0 设备掉电
Bit5	忙闲指示(Busy indication)	1 设备忙，表明最近一次 I2C 命令所要求读取的数据还未有效。如果设备忙，新的命令将不被处理。0 表明最近一次 I2C 命令所要求读 取的数据已经准备好被读取
Bit4	保留	固定为 0
Bit[3]	工作状态 (Mode Status)	0 NOR mode 1 CMD mode
Bit2	存储器数据完整性指示 (Memory integrity/error flag)	0 表示 OTP 存储器数据完整性测试 (CRC) 通过， 1 表示完整性测试失败。对数据完整性的测试只在上电过程中(POR)计算一次，所以被写入的新 CRC 值只能在接下来的 POR 之后使用。
Bit1	保留	固定为 0
Bit0	保留	固定为 0

应用示例

```
/* ********************************************************************** *
 * WPS_IIC.c
 * Date: 20XX/XX/XX
 * Revision: 1.0.0
 *
 * Usage: IIC read and write interface
 * ***********************************************************************/
#include "WPS_IIC.h"

//IIC clock line sbit
SCL = P1 ^ 1;

//IIC data line sbit
SDA = P1 ^ 0;

//Set the input and output mode of IIC data pin
#define  Set_SDA_INPUT() \
    P1MDOUT  &=  0xFE;  \
    P1 |= 0X01
#define Set_SDA_OUTPUT() P1MDOUT |=
0x01;
////Delay function needs to be defined void
DelayUs(unsigned char i) {
}

  //Start signal
  void
  Start(void) {
   SDA = 1;
   DelayUs(2);
   SCL = 1;
   DelayUs(2);
   SDA = 0;
   DelayUs(2);
   SCL = 0;
}

    //Stop signal
    void Stop(void)
    {
```

```
    Set_SDA_OUTPUT(
    ); SDA = 0;
    DelayUs(2);
    SCL = 1;
    DelayUs(2);
    SDA = 1;
    DelayUs(2);
}
//Read ACK signal
unsigned char Check_ACK(void)
{
     unsigned char ack;
     Set_SDA_INPUT();
    SCL = 1;
    DelayUs(2);
    ack  =  SDA;
    SCL = 0;
    Set_SDA_OUTPUT(
    ); return ack;
}
//Send ACK signal
void Send_ACK(void)
{
Set_SDA_OUTPUT(
); SDA = 0;
DelayUs(2);
SCL = 1; DelayUs(2);
SCL = 0; DelayUs(2);
}
//Send one byte
void SendByte(unsigned char
byte) {
unsigned char i = 0;
Set_SDA_OUTPUT(
); do
{
if (byte & 0x80)
{
            SDA = 1;
}
else
```

```
                {
                        SDA = 0;
                }
                DelayUs(2);
                SCL = 1;
                DelayUs(2);
                byte <<= 1;
                i++;
                SCL = 0;
}
    } while (i < 8);
    SCL = 0;
  //Receive one byte
  unsigned char
  ReceiveByte(void) {
  unsigned char i = 0, tmp = 0;
  Set_SDA_INPUT();
  do
  {
                tmp <<= 1;
                SCL = 1;
                DelayUs(2);
                if (SDA)
                {
                        tmp |= 1;
                }
                SCL = 0;
                DelayUs(2)
                ; i++;
     } while (i < 8);
     return tmp;
  }
  //Write a byte of data through IIC
  uint8 BSP_IIC_Write(uint8 address, uint8 *buf, uint8
  count) {
     unsigned char timeout,
     ack; address &= 0xFE;
     Start();
     DelayUs(2);
     SendByte(address);
     Set_SDA_INPUT();
     DelayUs(2);
     timeout = 0;
```

```
    do
    {
            ack =
            Check_ACK();
            timeout++;
            if (timeout == 10) {
                    Stop();
                    return 1;
            }
} while (ack);
while (count)
 {
          SendByte(*buf);
          Set_SDA_INPUT(
          ); DelayUs(2);
            timeout = 0;
           do
           {
  ack = Check_ACK();
  timeout++;
  if (timeout == 10)
  {
              return 2;
                        }
            } while (0);
            buf++;
            count--;
    }
    Stop();
    return 0;
}
  //Read a byte of data through IIC
  uint8 BSP_IIC_Read(uint8 address, uint8 *buf, uint8
  count) {
  unsigned char timeout,
  ack; address |= 0x01;
  Start();
  SendByte(address);
  Set_SDA_INPUT();
  DelayUs(2);
  timeout = 0;
  do
  {
```

```
            ack =
            Check_ACK();
            timeout++;
            if (timeout == 4)
            {
                    Stop();
                    return 1;
            }
 } while (ack);
 DelayUs(2);
while (count)
{
            *buf = ReceiveByte();
            if (count != 1)
                    Send_ACK();
                buf++;
                count--;
      }
       Stop();
      return 0;
 }
  /* ********************************************************************************* *
       * WPS_IIC.h
        * Date: 20XX/XX/XX
       * Revision: 1.0.0
 *
* Usage: IIC read and write interface
 * ********************************************************************************/#ifndef
 WPS_IIC_H_
 #define    WPS_IIC_H_
       uint8 BSP_IIC_Write(uint8 IIC_Address, uint8 *buffer, uint8
      count); uint8 BSP_IIC_Read(uint8 IIC_Address, uint8 *buffer,
                                  uint8 count);
#endif
/* ********************************************************************************* *
* WPS.c
 * Date: 20XX/XX/XX
* Revision: 1.0.0 *
* Usage: Sensor Driver file for WPS
* ********************************************************************************/
#include "WPS.h"
#include "WPS_IIC.h"
```

```
// Define the upper and lower limits of the calibration pressure
#define PMIN 20000.0  //Full range pressure for example 20Kpa
#define PMAX 120000.0  //Zero Point Pressure Value, for example 120Kpa
#define DMIN 3355443.0  //AD value corresponding to pressure zero, for example 20%AD=2^24*0.2
#define DMAX 13421772.0  //AD Value Corresponding to Full Pressure Range, for example 80%AD=2^24*0.8

  //The 7-bit IIC address of the JHM1200 is 0x78
  uint8 Device_Address = 0x78 << 1;

  //Delay function needs to be defined
  void DelayMs(uint8 count)
  {
  }

  //Read the status of IIC and judge whether IIC is busy
  uint8 WPS_IsBusy(void)
  {
    uint8 status;
    BSP_IIC_Read(Device_Address, &status,
    1); status = (status >> 5) & 0x01;
    return status;
  }


 /**
  * @brief Using the 0xAC command to calculate the actual pressure and temperature using the WPS internal
algorithm
  * @note  Send 0xAC, read IIC status until IIC is not busy
  * @note  The returned data is a total of six bytes, in order: status word, three-byte pressure value, two-byte
temperature value
  * @note  The returned three-byte pressure value is proportional to the 24-bit maximum value 16777216.
According to this ratio,

          the actual pressure value is again converted according to the calibration range.
  * @note  The returned two-byte temperature value is proportional to the 16-bit maximum value 65536.
According to this ratio,
          the actual pressure value is again converted according to the calibration range.
  * @note  Zero pressure point and full pressure point of calibration pressure correspond to 20kpa and
120Kpa, respectively
  * @note  The zero point of the calibration temperature is -40°C and the full point is 150°C
  * @note  The pressure actual value is calculated according to the span pressure unit is Pa, temperature
actual value temp unit is 0.01°C
  */
 void
 WPS_get_cal(void) {
```

```
    uint8 buffer[6] = {0};
    uint32 Dtest = 0;
    uint16 temp_raw = 0;
    double pressure = 0.0, temp = 0.0;

    //Send 0xAC command and read the returned six-byte data
    buffer[0] = 0xAC;
    BSP_IIC_Write(Device_Address, buffer, 1);
                if
    (WPS_IsBusy())
    DelayMs(5);
                {
    while (1)
    {

                        DelayMs(1);
                }
                else
                        break;
       }
       BSP_IIC_Read(Device_Address, buffer, 6);

     //The returned pressure and temperature values are converted into actual values according to the
  calibration range
     Dtest = ((uint32)buffer[1] << 16) | ((uint16)buffer[2] << 8) | buffer[3];
     temp_raw = ((uint16)buffer[4] << 8) | (buffer[5] << 0);
     pressure = (PMAX-PMIN)/(DMAX-DMIN)*(Dtest-DMIN)+PMIN; temp =
     (double)temp_raw / 65536;
     temp = temp * 19000 - 4000;
  }
 /* ************************************************************************** *

* File : WPS.h
 *
 * Date : 20XX/XX/XX
 *
* Revision : 1.0.0 *

* Usage: Sensor Driver for WPS sensor
 * ************************************************************************/#ifndef
 WPS_H
 #define    WPS_H__
                    __

 void WPS_get_cal(void);

 #endif
```