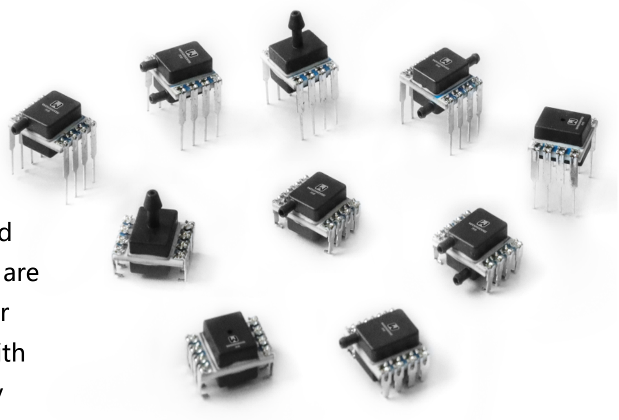
WPS0804 Series Pressure Sensor

Product introduction

WPS0804 Serise Basic Board Mount Pressure Sensors are high-precision high-performance piezoresistive silicon pressure sensor independently developed by Shenzhen Woosens Technology Co., Ltd., with a built-in dedicated ASIC for calibration and temperature compensation of the sensor's offset, temperature effect, sensitivity and nonlinearity. Provides digital or analog signal output in a variety of pressure measurement ranges and temperature ranges.This series of sensors can be used for differential or gauge pressure measurement and can be directly mounted on standard printed circuit boards.The WPS0804 series pressure sensors are suitable for use with non-corrosive, non-ionic gases such as air and other dry gases. All products are designed and manufactured in accordance with ISO9001 standards and undergo strict production calibration,and factory inspections ensure product consistency and reliability.



Ihdwxuhv

· I2C/SPI digital output	· Accuracy: ±0.25%FSS	· Total Error Band(TEB)：Optimum±1.0%
·Operating temperature -20~+85℃	· Differential /Gauge type	· Power supply：3.3V / 5V
· Barb shaped / No pressure port	· Sleep mode	· 24 or 14 bit resolution optional

Application

Industrial automation | Leak testing | Medical equipment | HAVC

Product specification

Absolute Maximum Ratings

SPI Output

Parameter	Minimum	Maximum	Unit
Supply voltage	-0.3	6	Vdc
Digital interface clock frequency:	50	800	kHz
ESD susceptibility (human body model)		4	kV
Storage temperature	-40	125	°C
Over load pressure	2 times full scale		
Burst pressure	3 times full scale		
Soldering time and temperature	Peak reflow temperature	15 s max. at 250°C	
	lead solder temperature	4 s max. at 250°C	

I2C Output

Parameter	Minimum	Maximum	Unit
Supply voltage	-0.3	3.6	Vdc
Digital interface clock frequency	100	400	MHz
ESD susceptibility (human body model)	-	3	kV
Storage temperature	-40	125	°C
Over load pressure	2 times full scale		
Burst pressure	3 times full scale		
Soldering time and temperature	Peak reflow temperature lead solder temperature	15 s max. at 250° C 4 s max. at 250° C	

Analog Output

Parameter	Minimum	Maximum	Uit
Analog supply voltage	-0.3	6	Vdc
Analog power ground	0	0	V
Analog and digital IO PIN voltage	-0.3	VDD+0.3	V
ESD susceptibility (human body model)	-	3	kV
Storage temperature	-40	125	°C
Over load pressure	2 times full scale		
Burst pressure	3 times full scale		
Soldering time and temperature	Peak reflow temperature lead solder temperature	15 s max. at 250° C 4 s max. at 250° C	

Operating Parameter

SPI Output

Parameter	Minimum	Typical	Maximum	Unit
Power supply: 3.3Vdc 5.0 Vdc	 3.0 4.75	 3.3 5.0	 3.6 5.25	Vdc
Supply current: 3.3 Vdc 5.0 Vdc	 - -	 1.6 2.0	 2.1 3	mA
Opreating temperture	-20	-	85	°C
Startup time (from power-up to data-ready)	-	2.8	7.3	ms
Response time	-	0.46	-	ms
Low level voltage	-	-	0.2	VDD
High level voltage	0.8	-	-	VDD
load resistance	1	4.7	-	kOhm
Accuracy	-0.25	-	0.25	%FSS BFSL
Resolution ratio	-	14	-	bit

I2C Output (Rosolution ratio 24bit)

Parameter	Minimum	Typical	Maximum	Unit
Power supply: 3.3 Vdc	3.0	3.3	3.6	Vdc
Supply current: 3.3 Vdc	-	-	2.0	mA
Standby current (25°C)		-	0.1	mA
Opreating temperture	-20	-	85	°C
Startup time (from power-up to data-ready)	-	2.5	-	ms
Measuring Frequency	5	-	100	Hz
Low level voltage	-	-	0.2	VDD
High level voltage	0.8	-	-	VDD
load resistance	1	4.7	-	kOhm
Accuracy	-0.25	-	0.25	%FSS BFSL
Resolution ratio	-	24	-	bit
Default communication address	0X78			

I2C Output (Rosolution ratio 14bit)

Parameter	Minimum	Typical	Maximum	Unit
Power supply: 3.3 Vdc 5.0 Vdc 3.3 Vdc or 5.0 Vdc base on customer choice	3.0 4.75	3.3 5.0	3.6 5.25	Vdc
Supply current: 3.3 Vdc 5.0 Vdc	- -	1.6 2.0	2.1 3	mA
Opreating temperture	-40	-	125	°C
Startup time (from power-up to data-ready)	-	2.8	7.3	ms
Response time	-	0.46	-	ms
Low level voltage	-	-	0.2	Vsupply
High level voltage	0.8	-	-	Vsupply
load resistance	1	-	-	kOhm
Accuracy	-0.25	-	0.25	%FSS BFSL
Resolution ratio	-	14	-	bit
Default communication address	0X28			

Analog Output

Parameter	Minimum	Typical	Maximum	Unit
Power supply: 5 Vdc 3.3 Vdc	 4.0 3.0	 5.0 3.3	 5.5 3.6	Vdc
Average operating current: Minimum update rate Maximum update rate	 - -	 0.6 1.2	 0.8 1.8	mA
Power-on reset level	1.7	-	2.7	V
Opreating temperture	-20	-	85	°C
Startup time (from power-up to data-ready)	-	-	12	ms
Sampling frequency	-	1	-	KHZ
Low level voltage	-	-	0.2	V_{DD}
High level voltage	0.8	1	-	V_{DD}
load capacitor	0	1	15	nF
Accuracy	-0.25	-	0.25	%FSS BFSL

Note:

- The sensor is not reverse polarity protected. Connecting the wrong pin to power or ground may cause failure
- The compensated temperature range is the temperature range over which the sensor can produce an output proportional to pressure within specific performance limits.
- Operating temperature range is the temperature range over which the sensor can produce an output proportional to pressure, but not necessarily within specific performance limits.
- Accuracy: Maximum output deviation from a Best Fit Straight Line (BFSL) applied to the measured output over the pressure range at 25°C. Includes all errors due to pressure nonlinearity, pressure hysteresis, and non-repeatability.
- Total Error Band(TEB): The maximum deviation from the ideal transfer function over the entire compensated temperature and pressure range. Includes all errors due to zero, span, pressure nonlinearity, pressure hysteresis, non-repeatability, thermal zero offset, thermal span offset, and thermal hysteresis.
- Full Scale Span (FSS) is the algebraic difference between the output signal measured at the pressure maximum limit (Pmax.) and the pressure minimum limit (Pmin.) of the pressure range.
- Overpressure: The maximum pressure that can be safely applied to the product such that the product remains in specification when the pressure returns to the operating pressure range. Applying excessive pressure may cause permanent damage to the product. Unless otherwise specified, this applies to all available pressure ports at any temperature within the operating temperature range.
- Burst pressure: The maximum pressure that can be applied to any pressure port of the product without causing the pressure medium to escape. The product will not function properly after being subjected to any pressure in excess of burst pressure.

Environmental Specifications

Characteristic	Parameter
Humidity： Dry Gases	0% 到 95% RH
Life	1 million pressure cycles minimum

• Life may vary depending on specific application

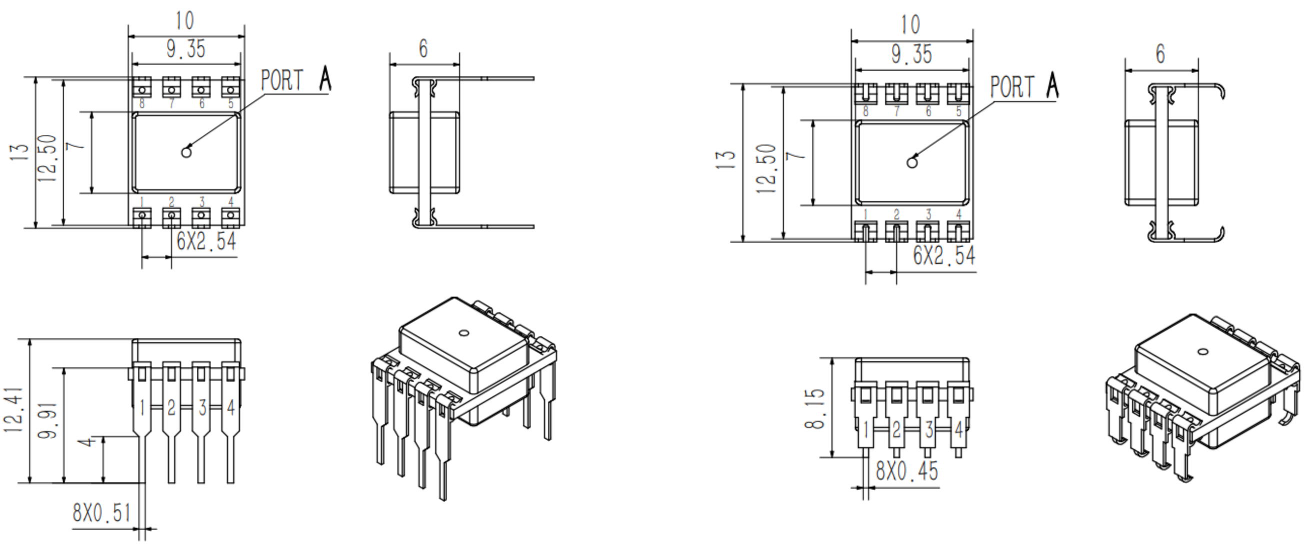
Pinouts

Outputs Type	PIN 1	PIN 2	PIN 3	PIN 4	PIN 5	PIN 6	PIN 7	PIN 8
I2C	GND	VDD	SDA	SCL	INT	NC	NC	NC
SPI	GND	VDD	MISO	SCLK	SS	NC	NC	NC
Analog	NC	VDD	Signal	GND	NC	NC	NC	NC

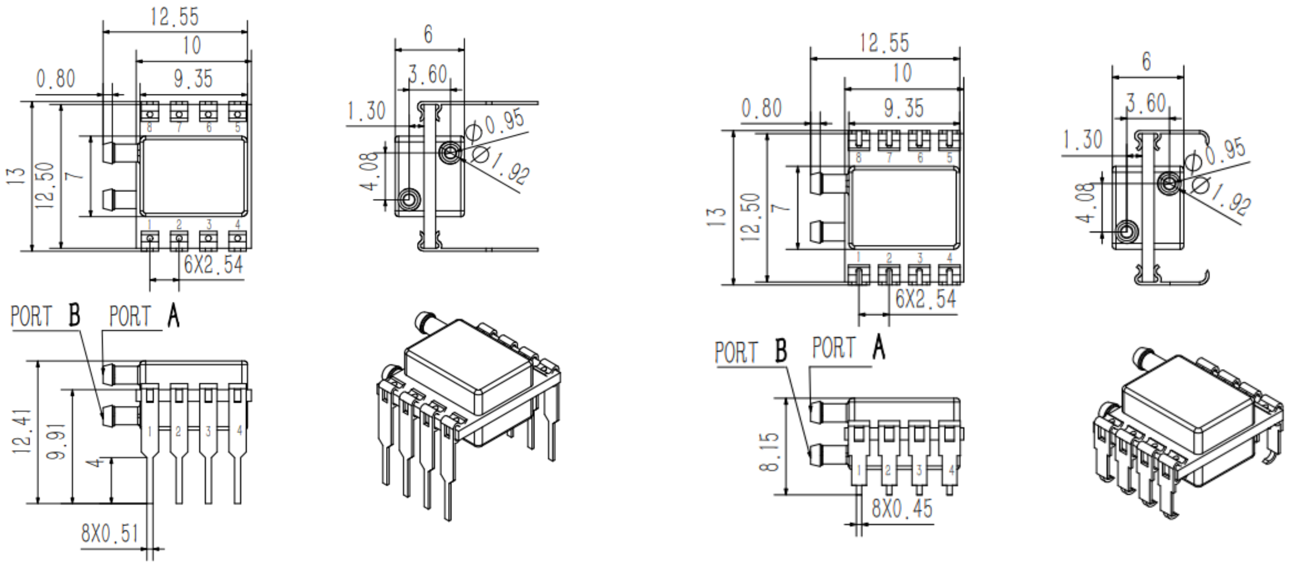
Structure Parameter

Units： mm

NN： No Port

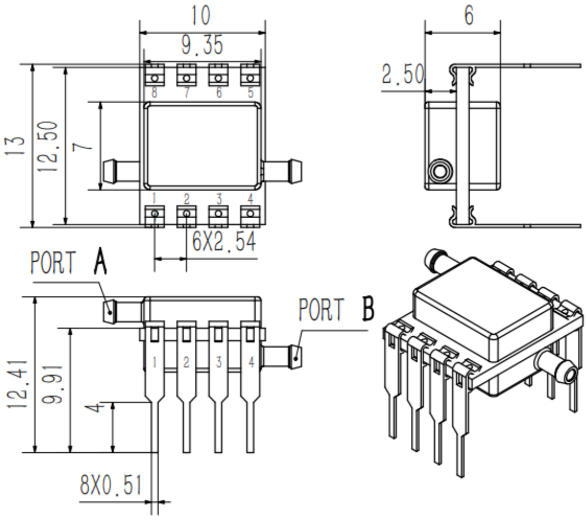


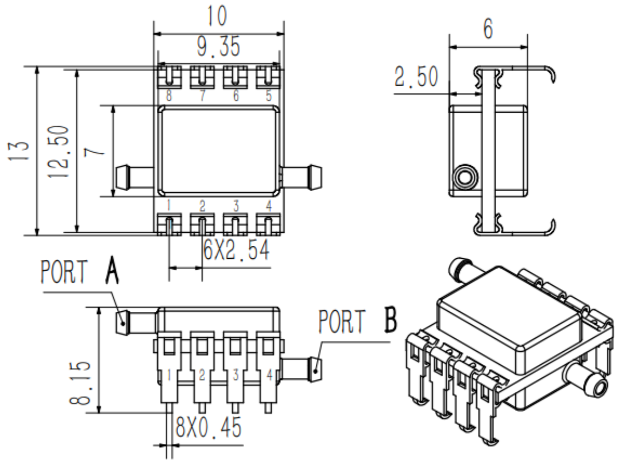
RR： Dual radial barbed ports, Ipsilateral



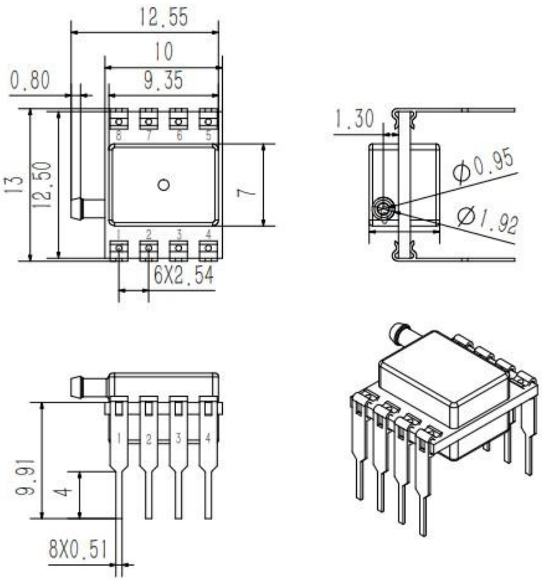
Room 502, Building 1, Jiuzhou Industrial Park, No. 10, Tongguan Road,
No.19 Road, Yutang Street, Guangming District, Shenzhen, P. R. China.
TEL:+86-755-83439588
www.woosens.com

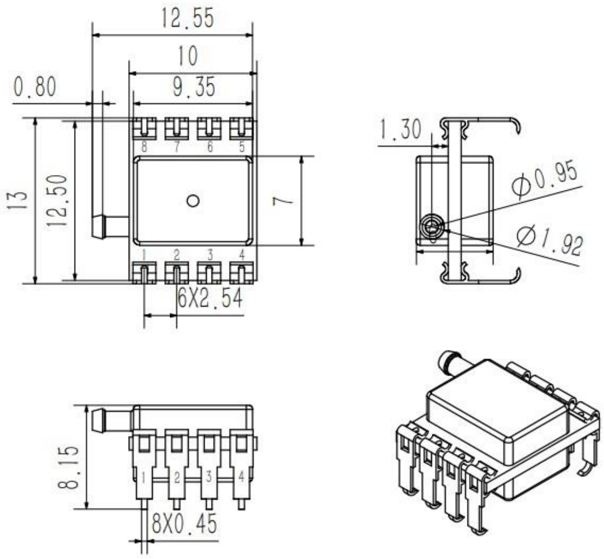
DR：Dual radial barbed ports, Contralateral



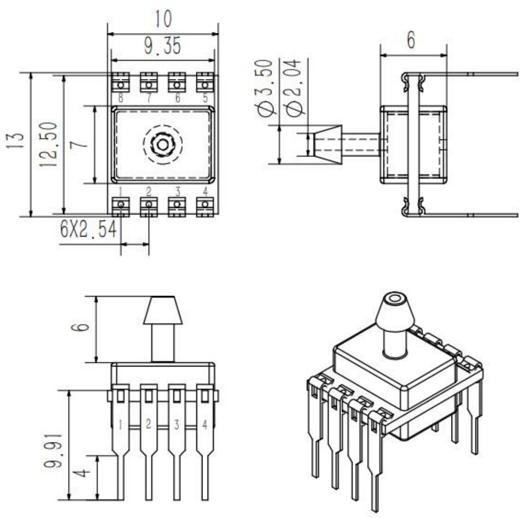


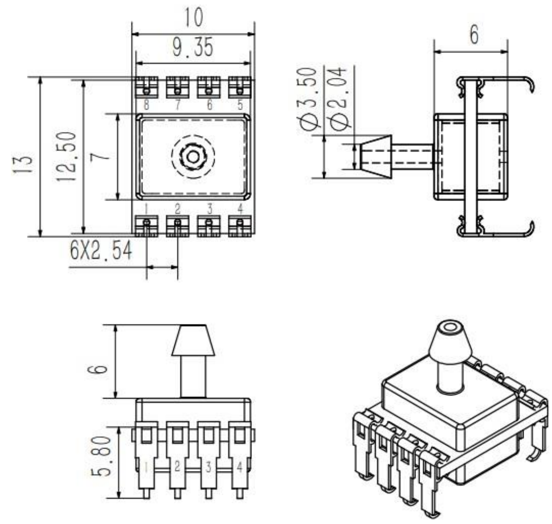
RN：Single radial barbed port





AN：Uniaxial Barbed Port





Ordering information

For example: WPS 08D 006K G 05 A NN 3 H

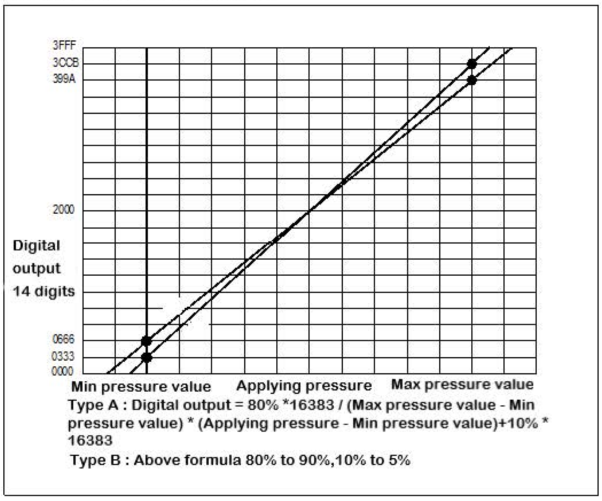
(1) (2) (3) (4) (5) (6) (7) (8) (9)

Series number	Significance	Description
1	Product series	WPS
2	Package	04 S: 4 PIN SIP 08 D: 8 PIN DIP 08 M: 8 PIN SMT
3	Pressure range	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) Note： When the range of the product is not reflected in this table, expand it according to the following example: For example： 005K as 5KPA For example： 010P as 10PA
4	Pressure Type	G： Gauge D： Differential
5	TEB Range	10： ±1.0%FS 15： ±1.5%FS 20： ±2.0%FS
6	Output Type	A: Analog C: I2C 14 bits resolution D： SPI I： I2C 24 bits
7	Pressure Port	NN： No Port AN： Uniaxial Barbed Port RN： Single radial barbed port RR： Dual radial barbed ports, Ipsilateral DR： Dual radial barbed ports, Contralateral
8	Power supply	3： 3.3VDC 5： 5.0VDC
9	Compensation Temperature	H： -20 C°~85C° M： 0 C°~85C° L： 0 C°~50C°

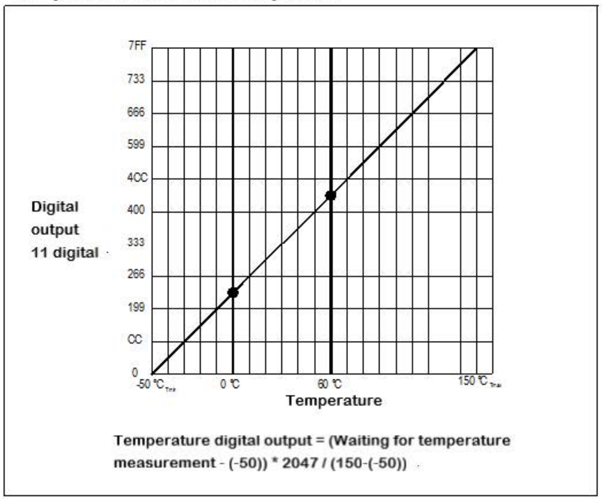
Pressure and Temperature Transfer Function

I2C/SPI Output（ 14 bits resolution ）

Pressure conversion equation



Temperature Conversion equation



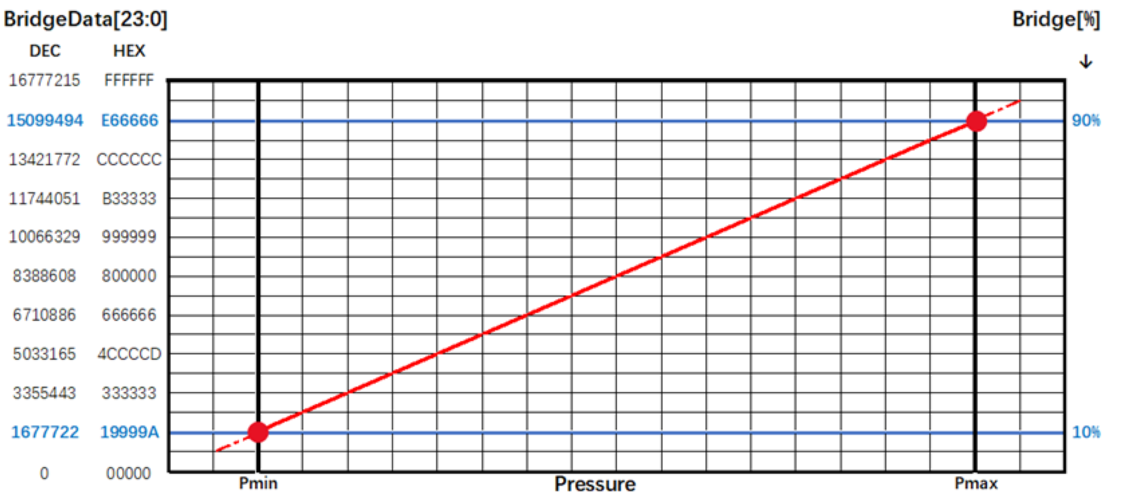
Unless otherwise specified, the products are all pressure transfer function TYPE-A sensors.

Sensor Output at Significant Percentages

% Output	Digtal counts (Decimal)
0	0
10	1638
50	8192
90	14745
100	16383

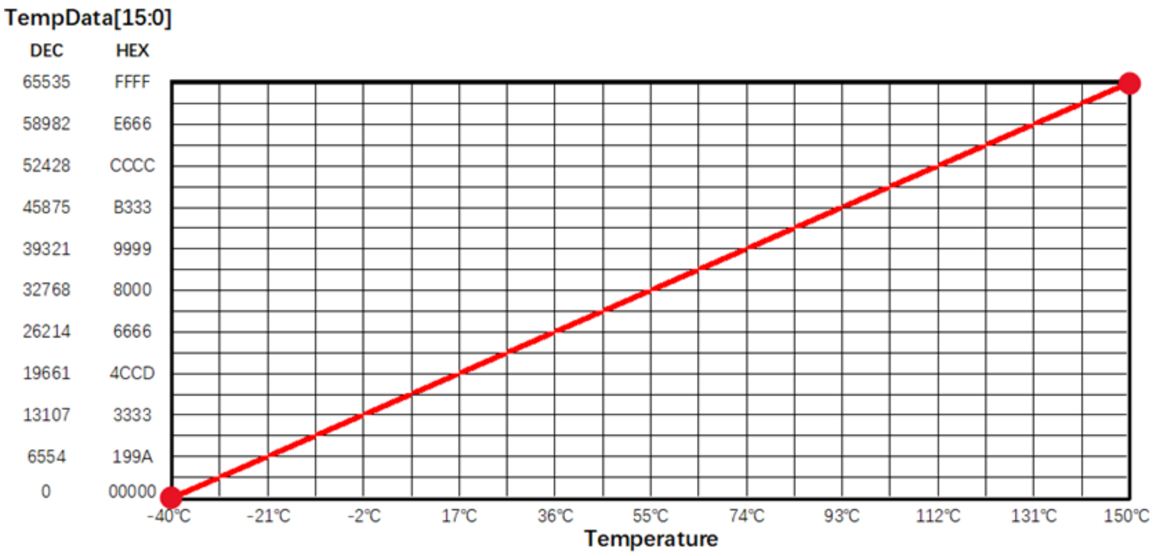
I2C Output（24bits resolution）

Pressure conversion equation



$$Pressure=(BridgeData[23:0]-10\%*16777215)*\frac{Pmax-Pmin}{80\%*16777215}+Pmin$$

Temperature conversion equation



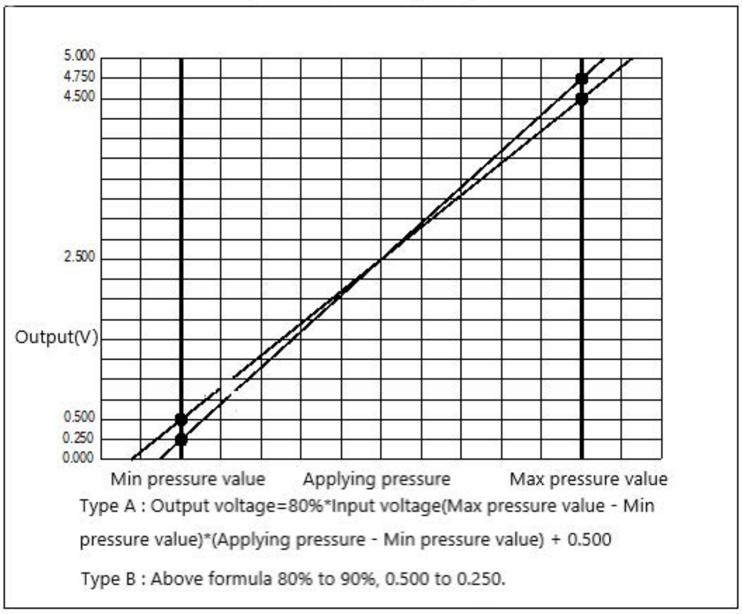
$$Temperature = \frac{TempData[15:0]}{65535} * 190 - 40$$

Sensor Output at Significant Percentages

% Output	Digtal counts (Decimal)
0	0
10	1677722
50	8388608
90	15099494
100	16777216

Analog output

Pressure conversion equation，voltage input 5V

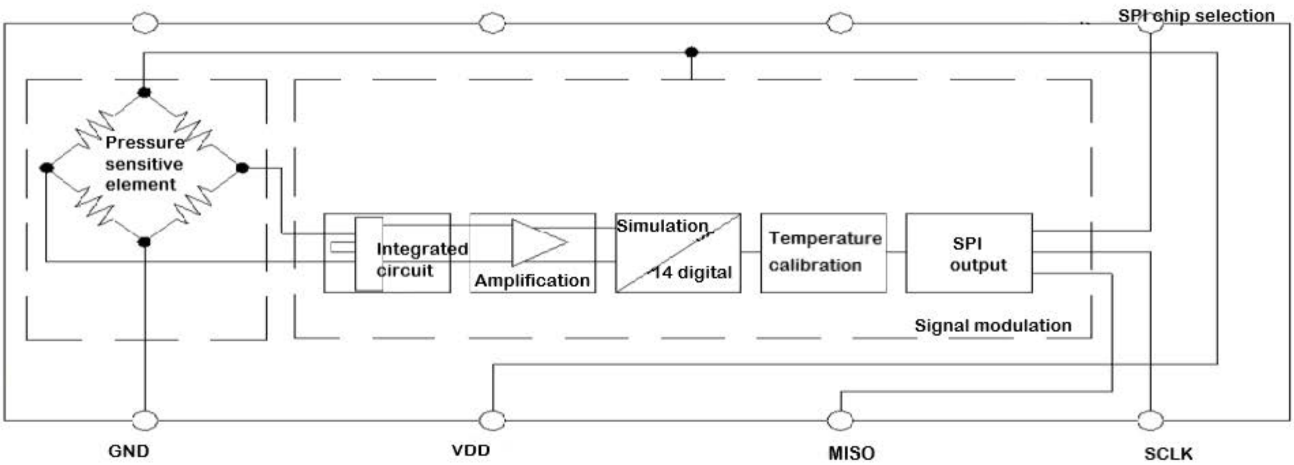


Sensor Output at Significant Percentages

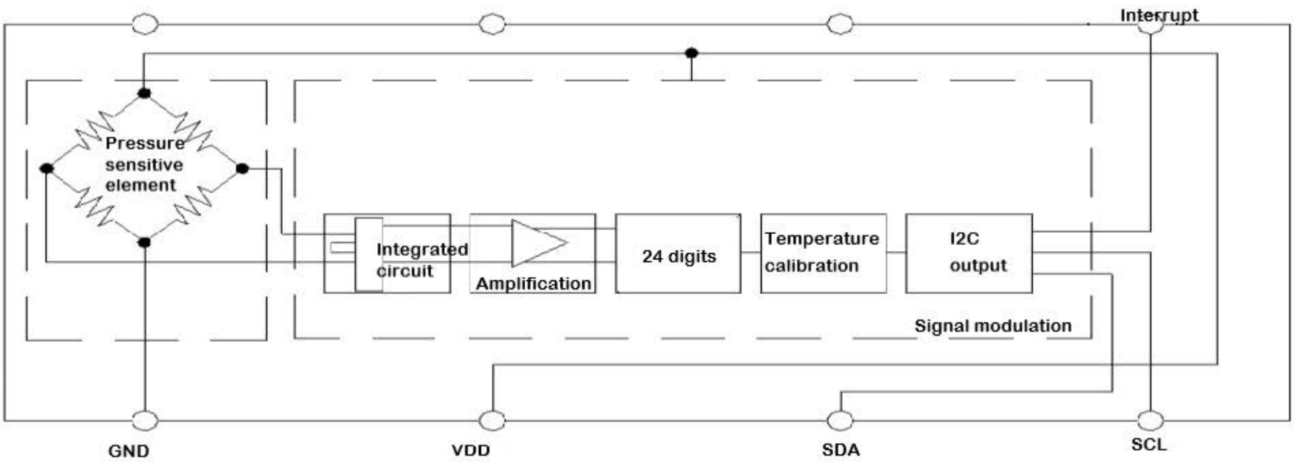
% Output	Analog Amplify（5V)
0	0
5	0.25
10	0.5
50	2.5
90	4.5
95	4.75
100	5

Equivalent Circuit

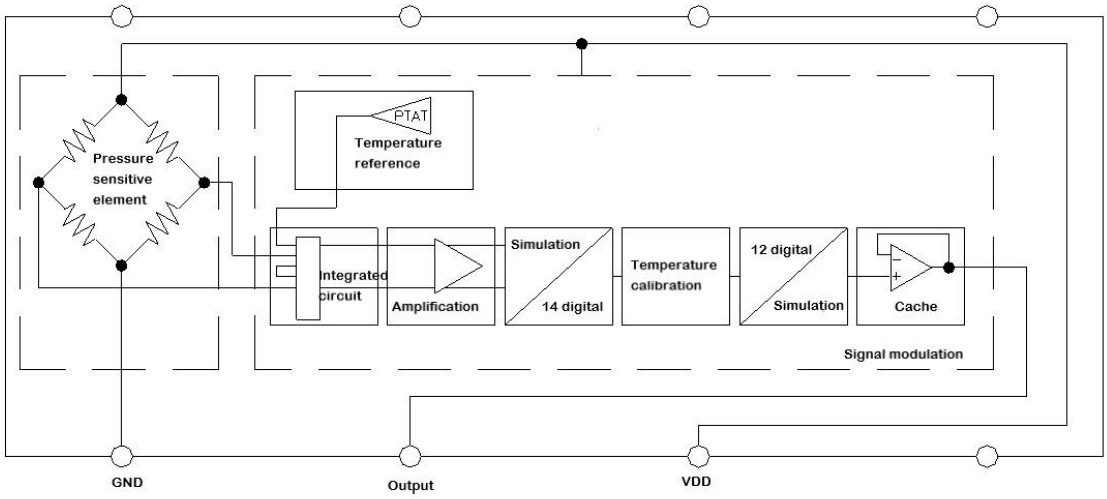
SPI Output



I2C Output



Analog output



Note:

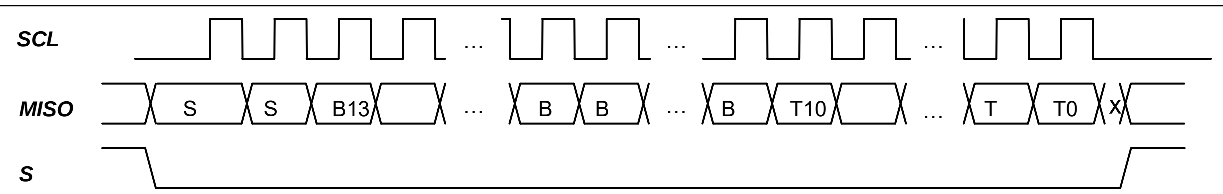
- It is recommended to place the pressure port A of the sensor downwards so that particles in the system cannot easily enter and stay inside the pressure sensor.
- Specifications are subject to change without notice.
- More information, Please contact Woosens sales .

SPI Communication

SPI Read_DF(Data Reading)

For simplicity of explanation and illustration, the following sections describe only the falling edge SPI polarity.
The SPI interface will have data changes after the falling edge of SCLK.Here we take MISO as an example to introduce the rise of SCLK.
The entire output packet is 4 bytes (32 bits).The high-bridge data bytes are sorted first, and the low-bridge data bytes are sorted last.Then send the 11-bit corrected temperature (T[10:0]):First the T[10:3] bytes, then the [T[2:0], xxxxx) bytes.The last 5 bits of the last byte are unknown,should be blocked in the application program.
If the user only needs the correct pressure value, the read can be terminated after the second byte.
If you also need corrected temperature, but only need 8-bit resolution, you can terminate the read after 3dbyte has been read.

SPI Output Packet with Falling Edge SPI_Polarity



Packet = [{S(1:0),B(13:8)}, {B(7:0)}, {T(10:3)},{T(2:0),xxxxx}] Where
S(1:0) = Status bits of packet (normal, command, busy, diagnostic) B(13:8) = Upper 6 bits of 14-bit bridge data.
B(7:0) = Lower 8 bits of 14-bit bridge data.
T(10:3) = Corrected temperature data (if application does not require corrected temperature, terminate read early)

Application Example:

C code example for SPI with Read_DF4 command

ReadWithSPI.c

```
/*
ReadWithSPI.c reads the digital output simply at any time and be assured the data is no older than the selected response time specification by checking the status of the 2 MSBs of the bridge high byte data */


/*PB0 = SCLK*/
/*PB1 = MISO*/
/*PB2 = SS*/
#include —iom164p.h‖
#define DF2        2
#define DF3        3
#define DF4        4
unsigned char bufptr[4];
void Init(void)
{
/* P0 = SCLK – output */
/* P1 = MISO – input */
/* P2 = SS – output */
/* P7, P6, P5, P4, P3, P2, P1, P0 */
/* O O O O O O I O */
/* 1 1 1 1 1 1 1 1 */
DDRB = 0xfd;
PORTB = 0xfc;
}
voidBitDelay(void)
{
char delay; delay = 0x03;
do

{
```

```
while(--delay) ;
_NOP();
return;
}
unsigned char GetOneByte (void)
{
unsigned char data=0;
  unsigned char i;
 for (i=0; i<8; i++)
{
 BitDelay();
  SCLK=1;
  BitDelay();
data=data<<1;
if (PINB & 0x02)
  data=data | 1;
  SCLK=0;
BitDelay()
}
return (data);
}
unsigned char ReadSA191D(unsigned char DF_Command)
{
  unsingned char i;
  SCLk=0;
  SS=0;
 BitDelay();
for (i=0; i<(DF_Command); i++)
{
   bufptr[i] = GetOneByte ();           /* 1 byte of read sequence */

}
SS=1;
BitDelay();
}
void main (void)
```

```
{
float Pressure, Temperature; unsigned
int Dpressure,Dtemperature;

 float P1= 819.15;          /* P1= 5% * 16383 – B type*/

 float P2= 15563.85;       /* P2= 95% *16383 – B type*/
 float Pmax= 2.0;
 float Pmin= -2.0;
 Init();
 do
 {
   ReadSA191D (DF4);         /*Read_DF4 command – data fetch 4 bytes */
   If((bufptr [0] & 0xc0)==0)              /*test status of the 2 MSBs of the bridge high byte of data*/
 {
     Dpressure= ((unsigned int) (bufptr [0] & 0x3f) <<8) + (bufptr [1]);
     Dtemperature= (((unsigned int) bufptr [2]) <<3) + bufptr [3];
     Pressure= (((float) Dpressure)-P1) * (Pmax-Pmin) / P2+Pmin;
    Temperature= ((float) Dtemperature) * 200 / 2047-50;
```

I2C Output （24bit resolution)

Instructions for Obtaining Calibration Values Using the 0xAC Command and Using the Sensor's Internal Calibration Algorithm..

The steps to send the 0xAC command to get the calibration value are as follows:

1. Send write commands

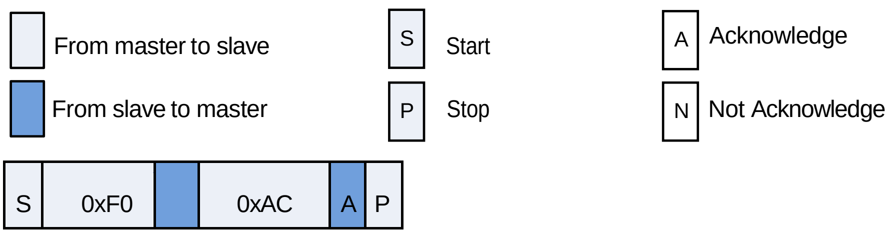


Chart 1 Write Commands

0xF0 in the write command means the default 7bits I2C device address is 0x78, and the last 1bit is 0 means the master device writes.

2. Wait

After sending the write command, you need to wait for a while before sending the read command, because it takes a while for the entire measurement to be completed internally. The waiting time depends on the setting of [13:11] bridge oversampling rate of OTP (Address: 0x14) and [15:14] temperature oversampling rate of OTP (Address: 0x14). Comparing with Table 1 and Table 2 in the appendix, the waiting time = tP + tT. The waiting time does not need to be calculated, and it can be judged whether the acquisition has been completed by continuously reading the IIC status word.

3、Read

To ensure that the time interval between the write command and the read command is greater than the measured duration, the calibration data can be read out. The reading format is shown in Figure 2. The 0xF1 in the read command means the default 7bits I2C device address is 0x78, and the last 1bit is 1 means the master Device read operation. The read calibration data consists of 6 bytes, which are 1 byte status word, 3 bytes bridge calibration value, and 2 bytes temperature calibration value.

S		0XF1	A	Status	A	BridgeDat [23:16]	A	BridgeDat [15:8]	A	BridgeDat [7:0]	A	TempDat [15:8]	A	TempDat [7:0]	N	P

Chart **2** I2C reads 5 bytes of calibrated pressure and temperature values

4、Conversion

After reading the calibration data, it is necessary to perform a simple conversion to the unsigned number in the form of AD value. For ease of understanding, we assume that the calibration data read is: 0x04 0x9B 0xB0 0xC5 0x56 0xAA

0x04 is the status word Bit5 is 1 to indicate that the latest I2C is busy and needs to wait for a period of time. If Bit5 is 0, it means the device is not busy, and data can be read. Please refer to the appendix for a detailed description of each bit of the status word.

0x9B 0xB0 0xC5 Three bytes are bridge calibration value

0x56 0xAA Two bytes are temperature calibration value

Bridge calibration value conversion: convert 0x9B 0xB0 0xC5 to decimal number as 10203333,

This calculation assumes that the calibration range is 20Kpa-120Kpa, and the corresponding AD output is 1677722~15099494 (10%AD~90%AD)

According to the calibration formula of the input-output relationship, we get:

Actual pressure value= (120-20) / (15099494-1677722) * (10203333-1677722) +20=83.5208 Kpa

Temperature calibration value conversion: Convert 0x56 0xAA to a decimal number as 22186. Since the read calibration data is expressed as a percentage, this percentage is numerically equal to the maximum value of the decimal number and 16bits unsigned number we converted (65535), so the following calculation can be done when converting the percentage

22186/65536*100%=33.85%

The calibration range of temperature is specified as -40℃—150℃, so the calibration value=(150—(-40))*33.85%—40=24.32℃

Appendix:

Table **1** Pressure Oversampling Rate and Measurement Time Comparison Table

OSR_Pressure[13:11]（Binary）	Corresponding oversampling rate	Measuring time **tP(ms)**
000	32768	203
001	16384	105
010	8192	56
011	4096	31
100	2048	19
101	1024	13
110	512	10

Table 2　Temperature Oversampling Rate and Measurement Time Comparison Table

OSR_Temperature[15:14]（Binary）	Corresponding oversampling rate	Measuring time **tT(ms)**
00	2048	19
01	4096	31
10	8192	56
11	16384	105

Table 3 **Status** Byte bit description:

Bit	Means	Description
Bit7	Reserve	Fixed at **0**
Bit6	Power indication	**1 VDDB on；0** VDDB off
Bit5	Busy indication	**1** Busy，Indicates that the data required to be read by the last I2C command is not yet valid。 **0** Indicates that the data requested by the last I2C command is ready to be read.
Bit4	Reserve	Fixed at **0**
Bit[3]	Mode Status	**0 NOR mode** **1 CMD mode**
Bit2	(Memory integrity/error flag)	**0** Indicates that the OTP memory data integrity test (CRC) passed， **1** Indicates that the integrity test failed. The test for data integrity is only calculated once during power-up (POR), so the new CRC value that is written can only be used after the following POR.
Bit1	Reserver	Fixed at **0**
Bit0	Reserve	Fixed at **0**

Application Example

```
/* ******************************************************************************* *

 * 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);

DelayMs(5);
while (1)
{
        if (WPS_IsBusy())
        {
                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
```

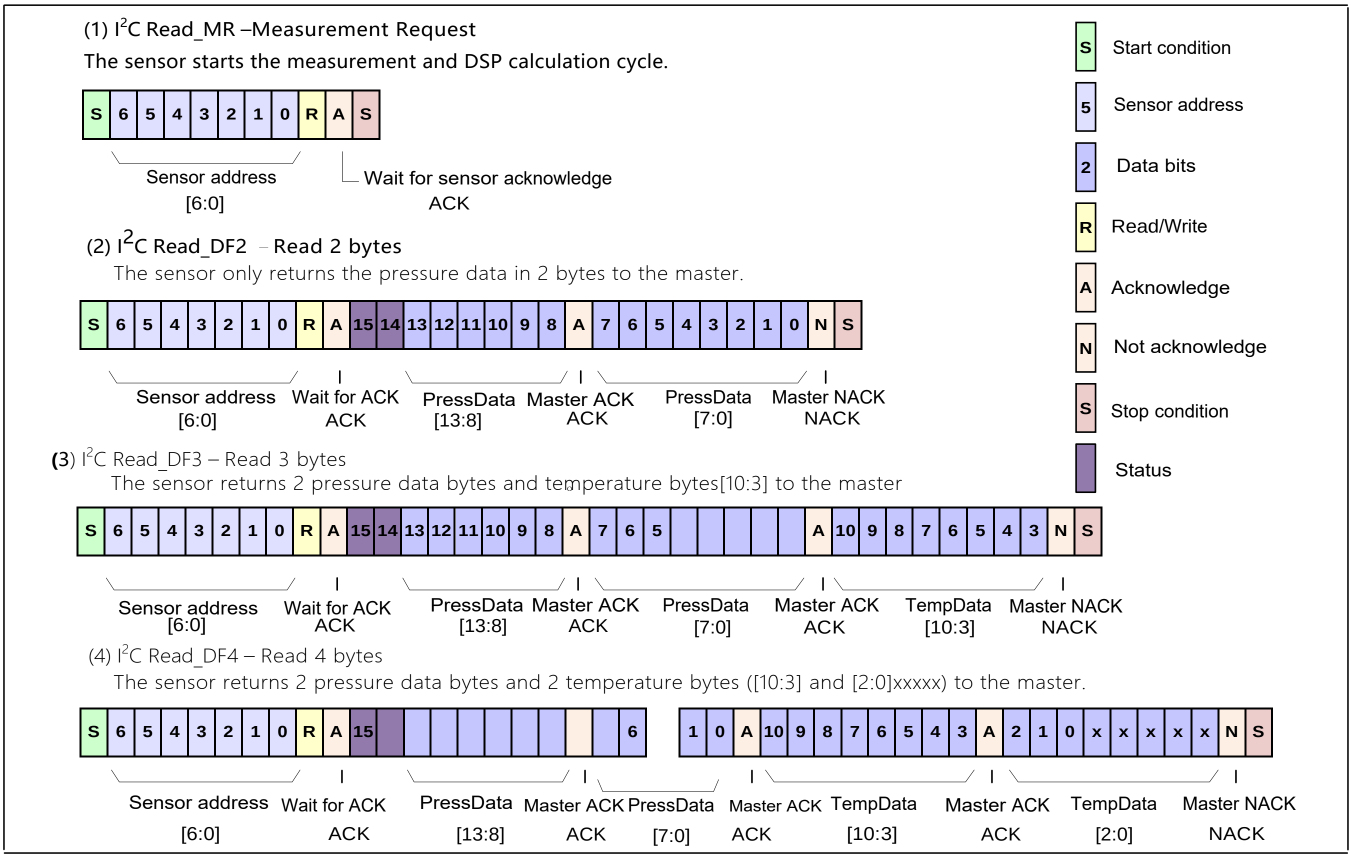
I2C Communication（14bits resolution)

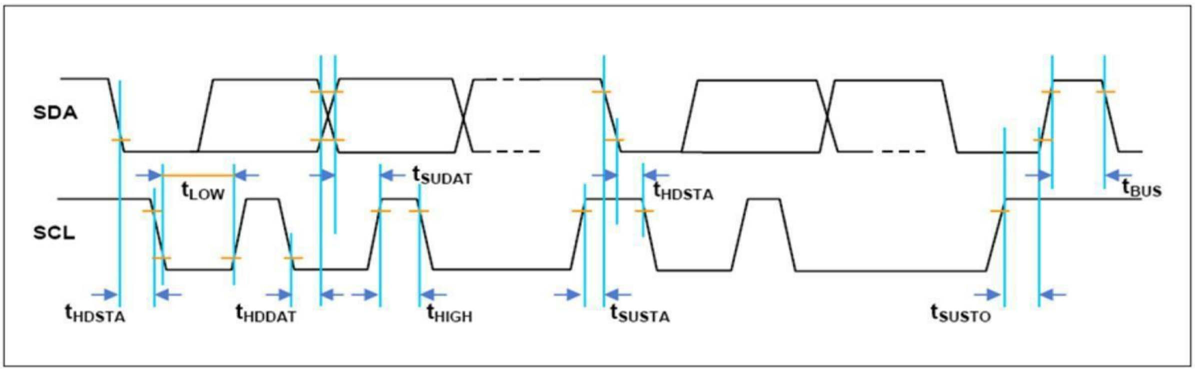
Read Operation

- The Master sends a 7-bit I2C address with the 8th bit as 1 (read). The sensor as a slave will send an acknowledgement (ACK) to indicate success.
- The sensor has 4 I2C read commands: READ_MR, READ_DF2, READ_DF3, READ_DF4. The figure below shows the structure of the 3 measurement packets in the four I2C read commands, which are explained further below.
- For the READ_DF3 data acquisition command (3 bytes of data acquisition), the sensor returns 3 bytes: two bytes of data, two status bits as Most Significant Bits (MSBs), and then one byte of temperature data (8 bits accuracy). After receiving the required number of data bytes, the master sends NACK and stop condition to terminate the read operation.
- For the READ_DF4 command, the master delays sending a not-acknowledge (NACK) and continues to read an additional last byte to obtain a fully corrected 11-bit temperature measurement. In this case, the last 5 bits of the last byte of the packet are undefined and should be screened out in the application.
- If temperature correction is not required, use the READ_DF2 command. The master terminates the read operation after two bytes of data.

I2C READ:

For data acquisition commands, the number of data bytes returned by the sensor is determined by the master sending NACK and stop condition.





Parameter	Abbreviation	MIN	TYP	MAX	UNITS
SCL clock frequency	FSCL	100		400	kHz
Start condition hold time relative to SCL edge	tHDSTA	0.1			µs
Minimum SCL clock low width	tLOW	0.6			µs
Minimum SCL clock high width	tHIGH	0.6			µs
Start condition setup time relative to SCL edge	tSUSTA	0.1			µs
Data hold time on SDA relative to SCL edge	tHIDDAT	0.0			µs
Data setup time on SDA relative to SCL edge	tSUDAT	0.1			µs
Stop condition setup time on SCL	tSUSTO	0.1			µs
Bus free time between stop condition and start condition	tBUS	2.0			µs

Differences between Sensor I2C protocol and general I2C protocol:

- Start-Stop Condition - No change on the CLK line (no clock pulse in between) will cause a communication error for the next communication, even if the next start-up condition is correct and there is a clock pulse. An additional START condition must be sent to restore normal communication.
- RESTART CONDITION - A falling edge of SDA while the CLK line is high during a data transfer can also cause communication to fail, and an additional start condition must be sent for proper communication.
- The start condition does not allow the first SCL edge to rise while the SDA edge is falling. If using an I2C address with the first bit 0, SDA must be held low from the start condition to the first bit.

Diagnostic Functions - Status Bits

- The sensor has diagnostic functions to ensure stable system operation. Diagnostic status is indicated by the status transfer of the high byte data of the 2 Most Significant Bits (MSBs).

Status Bits (2 Most Significant Bits (MSBs) output packets)	Description
00	Operation and packets are fine
01	Device in command mode (not for normal operation)
10*	Outdated data: data that has been acquired since the last measurement period
11	Presence of diagnostic condition

Note*: If data retrieval is performed before or during the first measurement after a power-on reset, "stale" is returned, but the data is actually invalid because the first measurement has not been completed

- One of the following faults is displayed when the two most significant bits (MSBs) are 11;
  - Invalid EEPROM signature
  - Positive or negative loss of resistance bridge
  - Resistor bridge input short circuit
  - Resistor bridge losses
- All diagnostics are detected in the next measurement cycle and reported in subsequent data acquisitions. Once a diagnostic is reported, the diagnostic status bits will not change unless the cause of the diagnostic is fixed and a power-on reset is performed.

Sleep Mode

- In sleep mode, after the command window, the sensor will be powered down until the master sends a Read_MR command, Read_MR will wake up the sensor and start a measurement cycle. If the command is Read_MR, the part performs temperature, auto-zero (AZ), and bridge measurements, then goes to DSP correction calculations, the rms value is written to the digital output register, and the sensor shuts down again.
- After a measurement sequence, before the next measurement can be performed, the host must send a Read_DF command which will fetch 2, 3 or 4 bytes of data without waking up the sensor.

When the Read_DF is executed, the returned packet will be the last measurement with the status bit set to "valid. After the Read_DF is complete, the status bit will be set to "stale".The next Read_MR will wake up the part again and start a new measurement cycle. If a Read_DF is sent while the measurement cycle is still in progress, the packet's status bit will be read as "stale".

Note: The I2C™Read_MR function can also be done using the I2C™Read_DF2 or Read_DF3 commands, and the ignored "stale" data will be restored.

I2C C code example using Read_DF4 command

On power-up, PORTB is initialized to all inputs with the internal pull-ups turned off, the external pull-ups pull the SDA and SCL lines high and the PORTB output latch bits SCL and SDA are initialized to zero. Routines WriteSDA and WriteSCL toggle their respective data direction bit depending on the value of parameter “state”. When state is a “1” the port pin is configured as input (external pull-ups pull high). When state is a “0” the port pin is configured as an output and the latch drives the pin low. WriteSDA and WriteSCL are very simple routines that could be incorporated into their respective calling routines to further reduce the code size.

General Calling Sequence for the Routines

```
SendStartBit();          /*start*/
SendByte(byt e);         /*send address or command MSB
GetOneByte();            first*/ /*read one byte from serial
SendStop();              stream */ /*stop*/
```

PORTB on the ATmega164P is used to communicate with SA18D transducer. Bit assignments are as follows:

```
I2C.c
/*PB0 =SDA*/
/*PB1 = SCL*/

#include “i2c.h”
void WriteSCL(unsigned char state)
{
 if (state)
     DDRB &= 0xfd;          /* input ... pullup will pull high or Slave will drive low */
 else
     DDRB |= 0x02;                  /* output ... port latch will drive low */
}

void WriteSDA(unsigned char state)
```

```
 {
    if (state)
     DDRB &= 0xfe;              /* input ... pullup will pull high or Slave will drive low */

  else
     DDRB |= 0x01;                      /* output ... port latch will drive low */

 }

 unsigned char SetSCLHigh(void)

 {

   WriteSCL(1);                         /* release SCL*/

   /* set up timer counter 0 for timeout */

   t0_timed_out = FALSE;                /* will be set after approximately 34 us */

   TCNT0 = 0;                  /* clear counter */

   TCCR0 = 1;                  /* ck/1 .. enable counting */

   /* wait till SCL goes to a 1 */

   while (! (PINB & 0x02) && !t0_timed_out) ;

   TCCR0 = 0;                  /* stop the counter clock */

   return(t0_timed_out);

}

void BitDelay(void)

{

  char delay;

  delay = 0x03;

  do

  {
   _NOP();
  } while (--delay);
}
```

/* Routine SendStopBit generates an TWI stop bit assumes SCL is low stop bit is a 0 to 1 transition on SDA while SCL is high

```
               _____________

            /

SCL  ___/
                    _______

                   /
```

```
   SDA _________/
   */
void SendStopBit(void)
{
   WriteSDA(0);
   BitDelay();
   SetSCLHigh(
   ); BitDelay();
   WriteSDA(1);
   BitDelay();
}
/* Routine SendStartBit generates an start bit start bit is a 1 to 0 transition on SDA while SCL is high
              ____________
             /
SCL ___/
       __________
                      \
SDA    \_____
*/
void SendStartBit(void)
 {
   WriteSDA(1);
   BitDelay();
   SetSCLHigh()
   ; BitDelay();
   WriteSDA(0);
   BitDelay();
   WriteSCL(0);
   BitDelay();
}
unsigned char SendByte(unsigned char byte)
{
   unsigned char i;
   unsigned char error;
   for (i = 0; i < 8; i++)
```

```
{
   WriteSDA(byte & 0x80);              /* if > 0 SDA will be a
   byte = byte << 1;             1 */  /* send each bit */
  BitDelay();
  SetSCLHigh();
  BitDelay();
  WriteSCL(0);
  BitDelay();
  }
  /* now for an ack */
  /* Master generates clock pulse for ACK */
  WriteSDA(1);                           /* release SDA ... listen for ACK */
  BitDelay();
  SetSCLHigh                          /* ACK should be stable ... data not allowed to change when SCL is
(); high */
   /* SDA at 0 ?*/
   error = (PINB & 0x01);              /* ack didn't happen if bit 0 = 1 */
   WriteSCL(0);
   BitDelay();
   return(error);
}
unsigned char GetOneByte(unsigned char lastbyte)
{
  /* lastbyte ==1 for last byte */
 unsingned char i;
 unsigned char data;
  DDRB &=0xfe;    /* release SDA ... listen for slave output */
  data=0;
  for (i=0; i<8;i++)
  {
    SetSCLHigh() ;                    /* Slave output should be stable ... data not allowed to change when
SCL is high  */
```

```
BitDelay();
data=data<<1;
if (PINB &0x01)
data=data |1;
WriteSCL(0);
BitDelay();
}
/*send ACK*/
WriteSDA (lastbyte); /* no ack on last byte ... lastbyte = 1 for the lastbyte */
BitDelay(); SetSCLHigh();
BitDelay();
WriteSCL(0);
BitDelay();
WriteSDA(1) ;
BitDelay();
return (data);
}
```

ReadWithPollingI2C.c

```
/*
ReadWithPollingI2C.c reads the digital output simply at any time and be assured the data is no older than the selected response time specification by checking the status of the 2 MSBs of the bridge high byte data
*/
#include "i2c.h"
extern unsigned char GetOneByte(unsigned char lastbyte);
extern unsigned char SendByte(unsigned char byte);
extern void SendStartBit(void);
extern void SendStopBit(void);
extern void BitDelay(void);
extern unsigned char SetSCLHigh(void);
extern void WriteSDA(unsigned char state);
extern void WriteSCL(unsigned char state);
```

```
unsigned char SA181DO_Address;

unsigned char bufptr[4];

void Init (void)


{

   __disable_interrupt();

   /* P0 = SDA - bidirectional */ /

   * P1 = SCL - output */

   /* P7, P6, P5, P4, P3, P2, P1,

   P0 */ /* O O O O O O O O */ /*

   1 1 1 1 1 1 1 1 */

   DDRB = 0xff;

   PORTB = 0xfc;

  /*setup SA181DO device address*/

   SA181DO_Address=0x28;

   /*

   The factory setting for I2C slave address is 0x28, 0x36 or 0x46 depending on the interface type
selected from the ordering information.

   For this sample code, 0x28 is used for Slave address of SA181DO.

    */

 }

 unsigned char ReadSA181DO(unsigned char DF_Command)

 {
     unsigned char i;
  unsigned char error;
  SendStartBit();
  if (SendByte((SA181DO_Address<<1) +re {ad))                    /*send salve address byte*/

  {
   return (1);    /*check error*/
   }

   for (i=0; i< (DF_Command-1); i++)

   {

   bufptr[i] =GetOneByte (0);                     /* 1 byte of read sequence */
```

```
  }
  bufptr[DF_Command-1] = GetOneByte(1);            /* 1 signals last byte of read sequence */

 SendStopBit();

 return (0);

}

 void main (void)
{
   float Pressure,Temperature;

     unsigned int Dpressure, Dtemperature;

     float P1=819.15;                  /* P1= 5% * 16383 – A type*/

     float P2=15563.85;                /* P2= 95% * 16383 – A type*/

     float Pmax=2.0;

     float Pmin=-2.0

Init();

do

 {

 ReadSA181DO (DF4);    /*Read_DF4 command – data fetch 4 bytes */

 If ((bufptr [0] & 0xc0) ==0x00)/*test status of the 2 MSBs of the bridge high byte of data*/

 {

 Dpressure= ((unsigned int) (bufptr [0] & 0x3f) <<8) + (bufptr [1]);

 Dtemperature= (((unsigned int) bufptr [2]) <<3) + bufptr [3];



 Pressure= (((float) Dpressure)-P1) * (Pmax-Pmin) / P2+Pmin;

 Temperature= ((float) Dtemperature) * 200 / 2047 -50;

   }

  }

  while(1);
```

```
}  /* main */
```

I2C.h

```
#include "iom164p.h"

#define    DF2    2
#define    DF3    3
#define    DF4    4

#define write 0
#define read 1
```