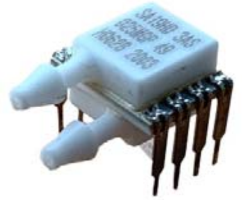
AT19系列

硅陶瓷系列
数字输出
模拟放大输出



产品概述

AT19系列硅陶瓷系列为压阻硅压力传感器，可提供指定满量程压力范围和温度范围读取压力的数字输出。AT19系列通过使用板载专用集成电路(ASIC)针对传感器偏移、灵敏度、温度效应和非线性进行了充分校准和温度补偿。经校准 的压力输出值会在1kHz 左右更新。AT19系列在 0℃到60℃的温度范围内进行校 准。该传感器可在3.3Vdc或5.0Vdc的单电源条件下工作。这些传感器测量绝压、差压和表压。绝压型号的传感器具备内部真空参照以及与绝压成比例的输出值。差压型号的传感器允许向感应模片的任意一侧加压。表压型号的传感器以大气压力为参考，提供与大气压力变化成比例的输出值。AT19 压力传感器适用于无腐蚀性、非离子气体（例如空气和其他干燥气体）。提供的选件可延伸这些传感器的性能，使其适用于无腐蚀性、非离子的液体。所有产品均遵循 ISO 9001 标准设计和制造。

用途

- 呼吸机、麻醉机
- 肺活量计
- 雾化器
- 医院室内气压控制
- VAV 调节系统
- 风道静压
- HVAC 滤清器堵塞检测
- HVAC(暖通空调)变送器

产品特点

- 多样化的封装：AT19 系列压力传感器为硅压阻精密压力传感器，采用模块化设计，具有多种封装类型可供选择（侧面供气，DIP型，SMT型），可满足客户不同安装环境的需要。
- 工作电压较低，能耗极小，供电电压可以为3.3V或5V输入
- 业界领先的长期稳定性：通过压力敏感芯片的优选和封装工艺的技术处理，作为微压力传感器与业内其它传感器相比表现出色，具有优异的长期稳定性。
- 内部诊断功能可增强系统的可靠性
- 还提供了模拟输出选项
- 绝压、差压和表压类型
- 达到 0.25% FSS BFSL（满刻度量程最佳直线）的极高精确度
- 总误差带为 1% 的满刻度量程最大值
- 所有这些产品都同样具备业界领先的性能规格
- 与 I 2C 或 SPI 兼容的 14 位数字输出，压力输出14位，可以同时监控温度，温度输出11位
- 数字输出或模拟放大输出提供10%到90%输出或5%到95%输出。
- 在 0 C到 60 C 的温度范围内进行精密ASIC 调节和温度补偿
- 符合 RoHS 标准
- 压力口特点：直径3.175毫米的倒钩状压力口可以稳固的和2.38毫米内经的压力管牢固连接测试压力。
- 客户定制：精度、总偏差和温度补偿范围以及信号输出方式等可根据客户需求定制，非标准产品请联系工厂。

标准压力范围量程（英寸水柱，PA）

1PSI	表压，差压	模拟放大和数字输出
2PSI	表压，差压	模拟放大和数字输出
5PSI	表压，差压	模拟放大和数字输出
15PSI	表压，差压，绝压	模拟放大和数字输出
30PSI	表压，差压，绝压	模拟放大和数字输出
50PSI	表压，差压，绝压	模拟放大和数字输出
100PSI	表压，差压，绝压	模拟放大和数字输出
150PSI	表压，差压，绝压	模拟放大和数字输出
300PSI	表压，差压，绝压	模拟放大和数字输出

100pa	表压，差压	模拟放大和数字输出
1Kpa	表压，差压	模拟放大和数字输出
10Kpa	表压，差压	模拟放大和数字输出
30Kpa	表压，差压，绝压	模拟放大和数字输出
100Kpa	表压，差压，绝压	模拟放大和数字输出
500Kpa	表压，差压，绝压	模拟放大和数字输出
1000Kpa	表压，差压，绝压	模拟放大和数字输出
2000Kpa	表压，差压，绝压	模拟放大和数字输出

额定值

参数	最小值	最大值	单位
电源电压 (Vsupply)	-0.3	6.0	Vdc
任意引脚上的电压	-0.3	Vsupply +0.3	V
数字接口时钟频率：I²C SPI	100 50	400 800	Khz
ESD 敏感度（人体模式）	4		Kv
存储温度	-40	125	℃
焊接时间和温度 铅焊料温度（SIP、DIP） 回流峰值温度 (SMT)	最多5秒，在250°C时 最多15秒，在250°C时		

性能规格

参数	最小值	典型值	最大值	单位
电源电压 (Vsupply) 3.3 5.0 *3.3 Vdc* 或*5.0 Vdc*具体取决于所选型号	 3.0 4.75	 3.3 [2] 5.0 [2]	 3.6 5.25	 Vdc Vdc
电源电流 3.3 Vdc 电源 5.0 Vdc 电源		 2.1 3		 mA mA
补偿温度范围 [3]	0	-	60	℃
工作温度范围[4]	-40	-	125	℃
启动时间（从加电到数据准备就绪）	-	2.8	7.3	ms
响应时间	-	0.46	-	ms
SPI和 I^2C 低电平	-	-	0.2	V_{supply}
SPI和 I^2C 高电平	0.8	-	-	V_{supply}
SDA/MISO, SCL/SCLK, SS上拉电阻	1	-	-	Kohm
精度[5]	-	-	±0.25	%FSS [7]
位置灵敏度	-	-	±0.15	%FSS
综合偏差[6]	-1%	-	1%	%FSS
过载压力[9]		3		倍
爆破压力[10]		5		倍
输出分辨率	12	-	-	位

环境规格

参数	特性
湿度：仅干燥气体	0% 到 95% RH，非冷凝
振动	MIL-STD-202F，方法 214，条件 F（20.7 g 随机）
冲击	MIL-STD-202F，方法 213B，条件 F
寿命[8]	100 万次循环
回流焊	J-STD-020D.1 MSL湿度灵敏度级别1

被测介质接触材料

盖子	PPS	PPS
基材	氧化铝陶瓷	氧化铝陶瓷
粘合剂	环氧树脂、硅树脂	环氧树脂、硅树脂
电子组件	陶瓷、玻璃、焊料、硅	硅、玻璃、金、焊料

备注：

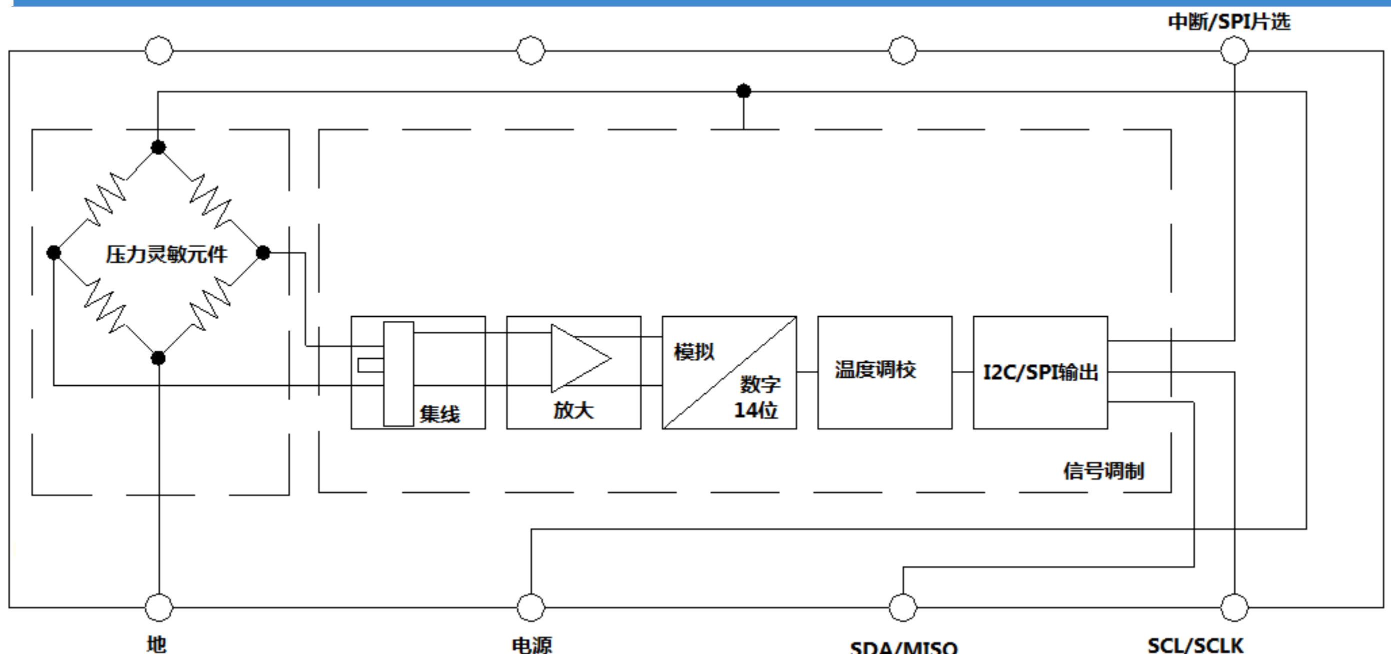
1. **额定值**是设备在不损坏的前提下所能承受的最大极限。
2. 该传感器不受反向极性保护。将错误的引脚与电源连接或者接地可能会导致电气故障。
3. 补偿温度范围是指传感器可以在特定的性能限制下产生与压力成比例的输出的温度范围.
4. 工作温度范围是指传感器可以产生与压力成比例的输出的温度范围，但不一定在特定性能限制范围之内。
5. 精度：相对适用于在 25°C 时的压力范围内所测输出的最佳直线 (BFSL) 的最大输出偏差。包括所有因压力非线性、压力滞后和不重复性造成的误差。
6. 综合偏差：相对整个补偿温度和压力范围内理想传递函数的最大偏差。包括所有因偏置、满刻度量程、压力非线性、压力滞后、可重复性、偏置热效应、量程热效应和热滞后造成的误差。
7. 满刻度量程 (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。
8. 寿命可能因传感器使用的特定应用而有所变化。
9. 过压：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。
10. 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。
11. 客户定制请联系良品实业业务人员。

注意：

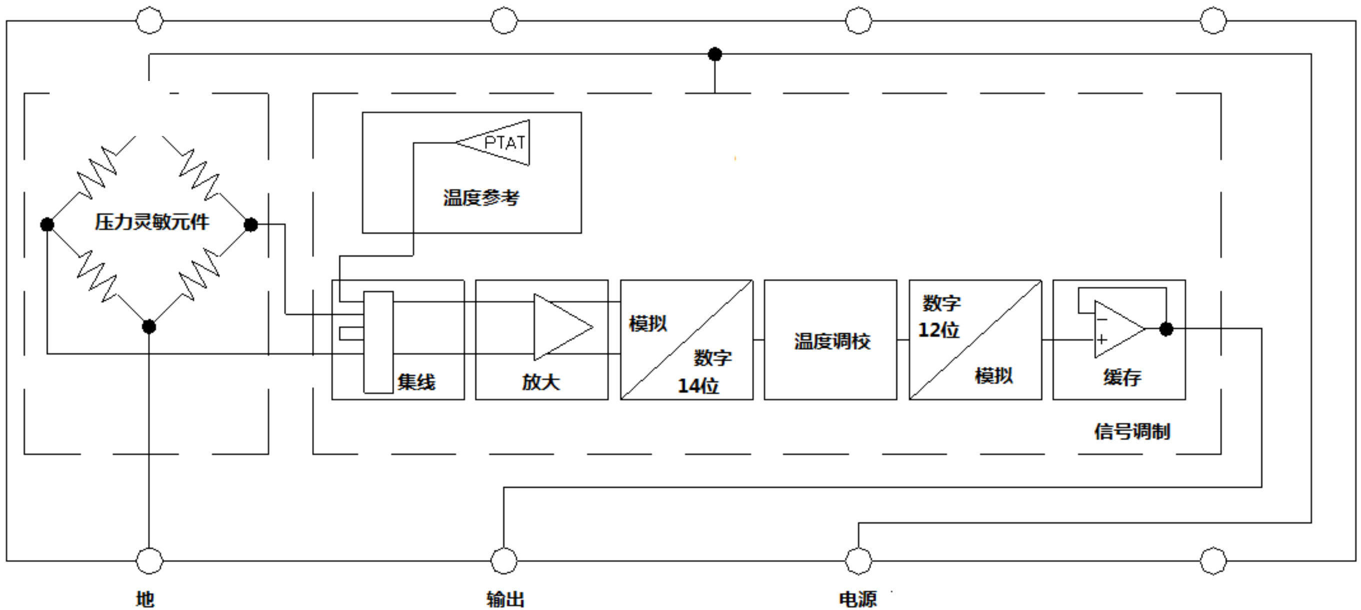
产品损坏

- 确保液体介质仅用于压力口A；压力口B 与液体不相容。
- 确保液体介质不含颗粒。所有 AT19传感器均为终端密封设备。颗粒会在传感器内积聚，造成设备损坏或影响传感器输出。
- 建议将传感器的压力口A 朝下放置，这样系统中的颗粒就不容易进入并停留在压力传感器内。
- 确保液体介质在干燥时不会产生残留物；传感器内的堆积物可能会影响传感器输出。清洗终端密封的传感器十分困难，并且无法有效地去除残留物。
- 确保液体介质与接液材料相容。不相容的液体介质会降低传感器的性能，并可能导致传感器故障。
- 不遵循这些说明可能会导致产品损坏。

等效电路



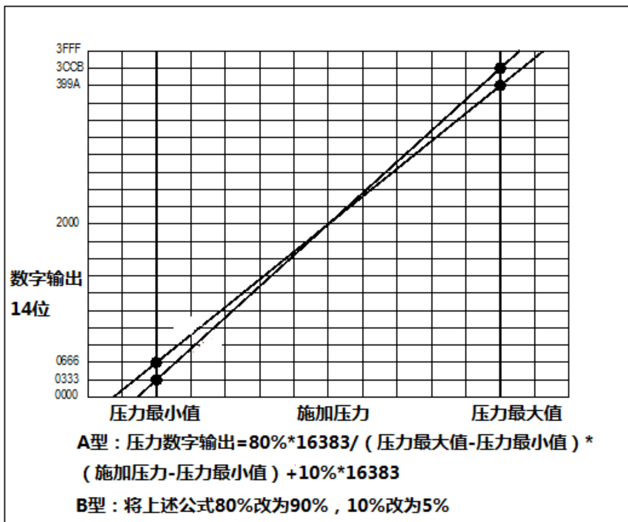
AT19 数字输出等效电路



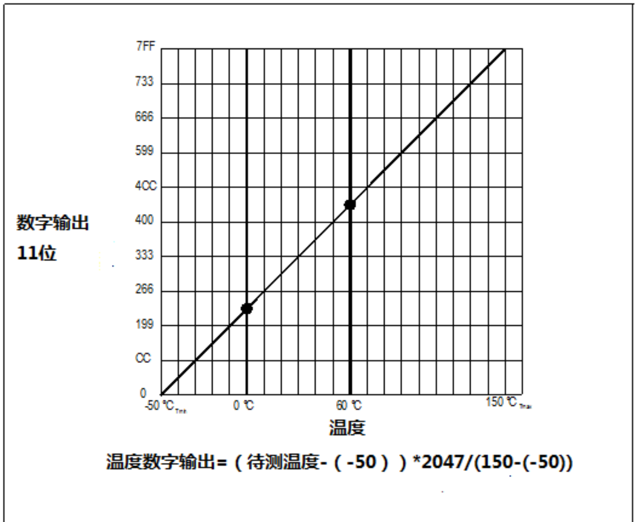
AT19 模拟放大输出等效电路

压力和温度输出对应公式

压力转换方程

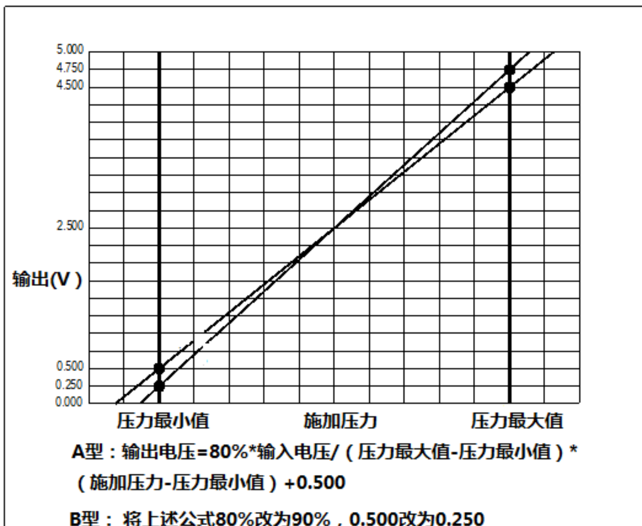


温度转换方程

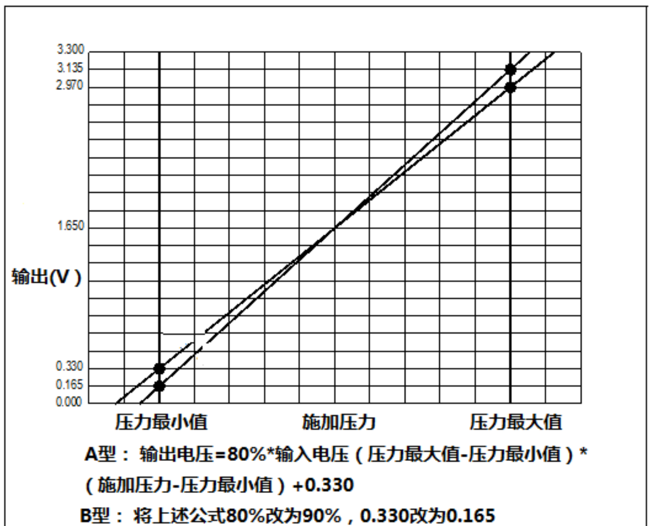


数字输出

压力转换方程，电压输入5V



压力转换方程，电压输入 3.3V



模拟放大输出 RESSURE AND TEMPERATURE RANSFER(ANALOG OUTPUT)

压力类型	说明
差压	输出与施加到各压力口的压力差成比例
表压	输出与施加压力和大气（环境）压力之间的差值成比例
绝压	输出与施加压力和真空压力之间的差值成比例

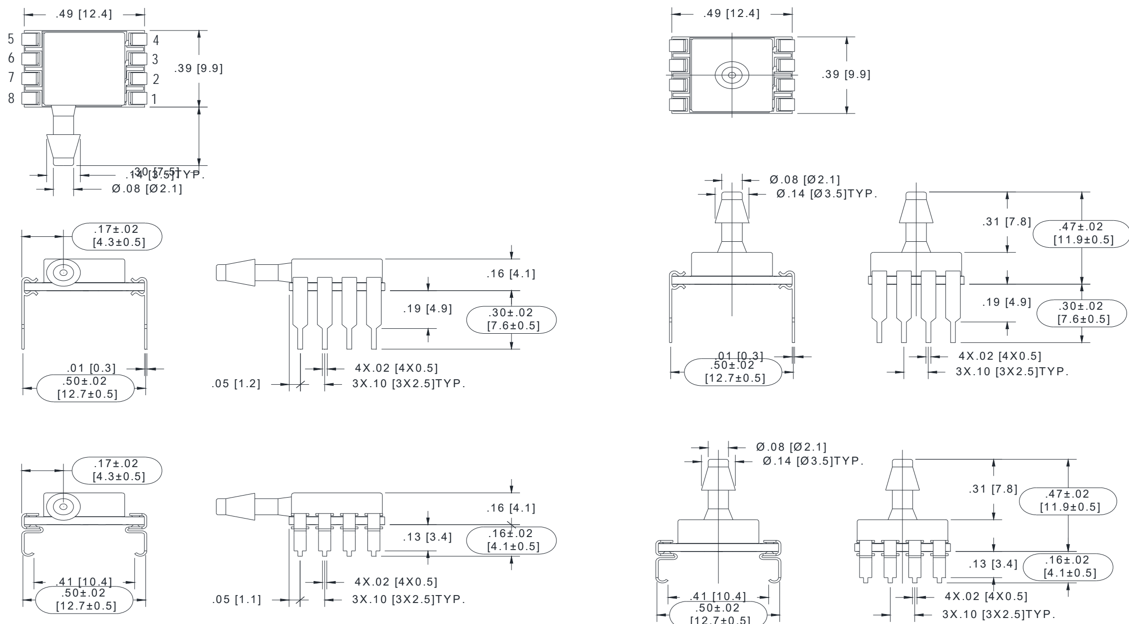
输出百分比

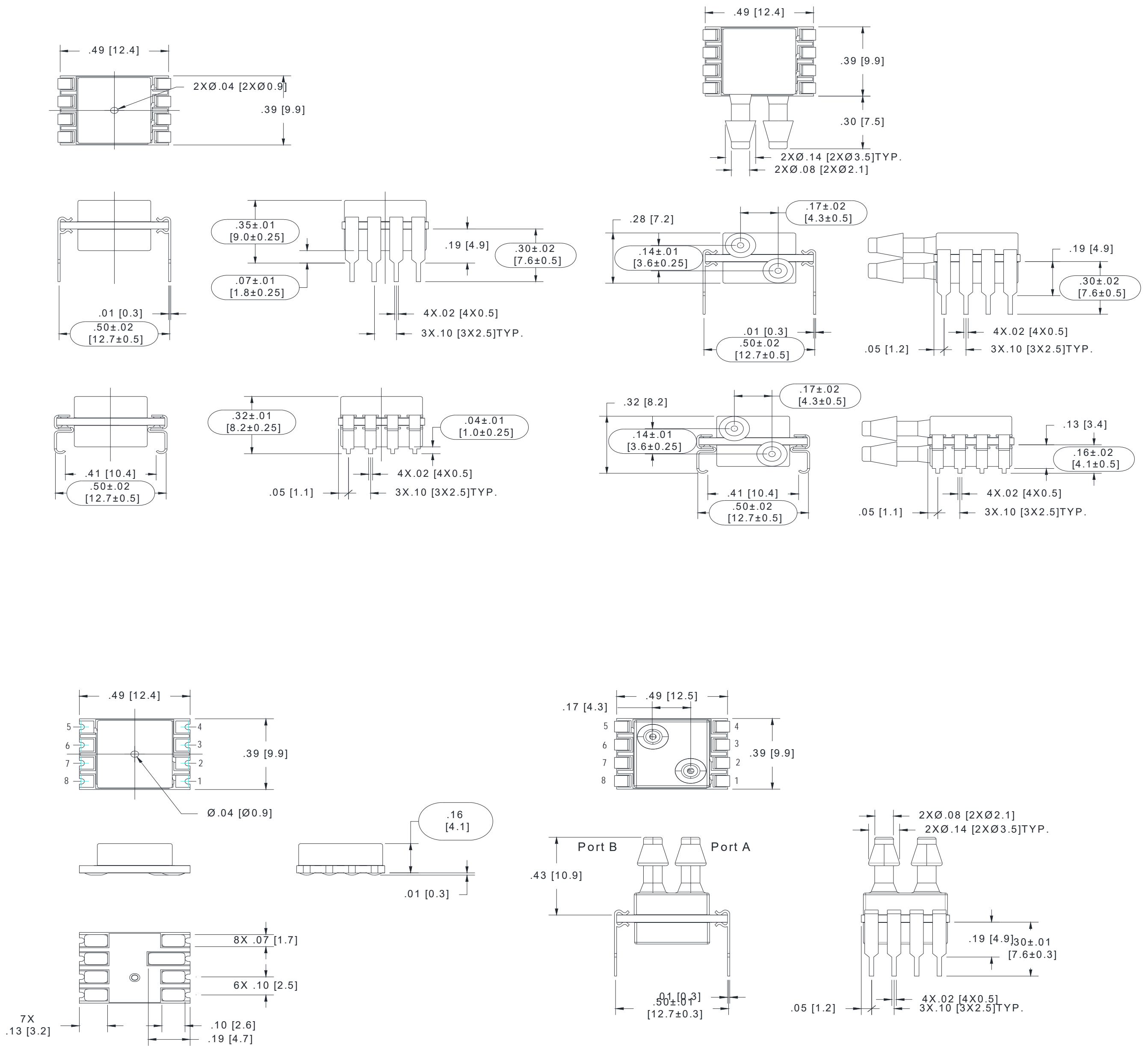
输出百分比（%）	数字计数（十进制）	模拟放大（5V）	模拟放大（3.3V）
0	0	0	0
5	819	0.25	0.165
10	1638	0.5	0.33
50	8182	2.5	1.65
90	14746	4.5	2.97
95	15565	4.75	3.135
100	16383	5	3.3

脚位定义

输出类型/脚位	Pin 1	Pin 2	Pin 3	Pin 4	Pin 5	Pin 6	Pin 7	Pin 8
模拟输出	空	电源	信号	地	空	空	空	空
数字输出	地	电源	SDA/MISO	SCL/SCLK	INS/SS	空	空	空

尺寸（毫米）





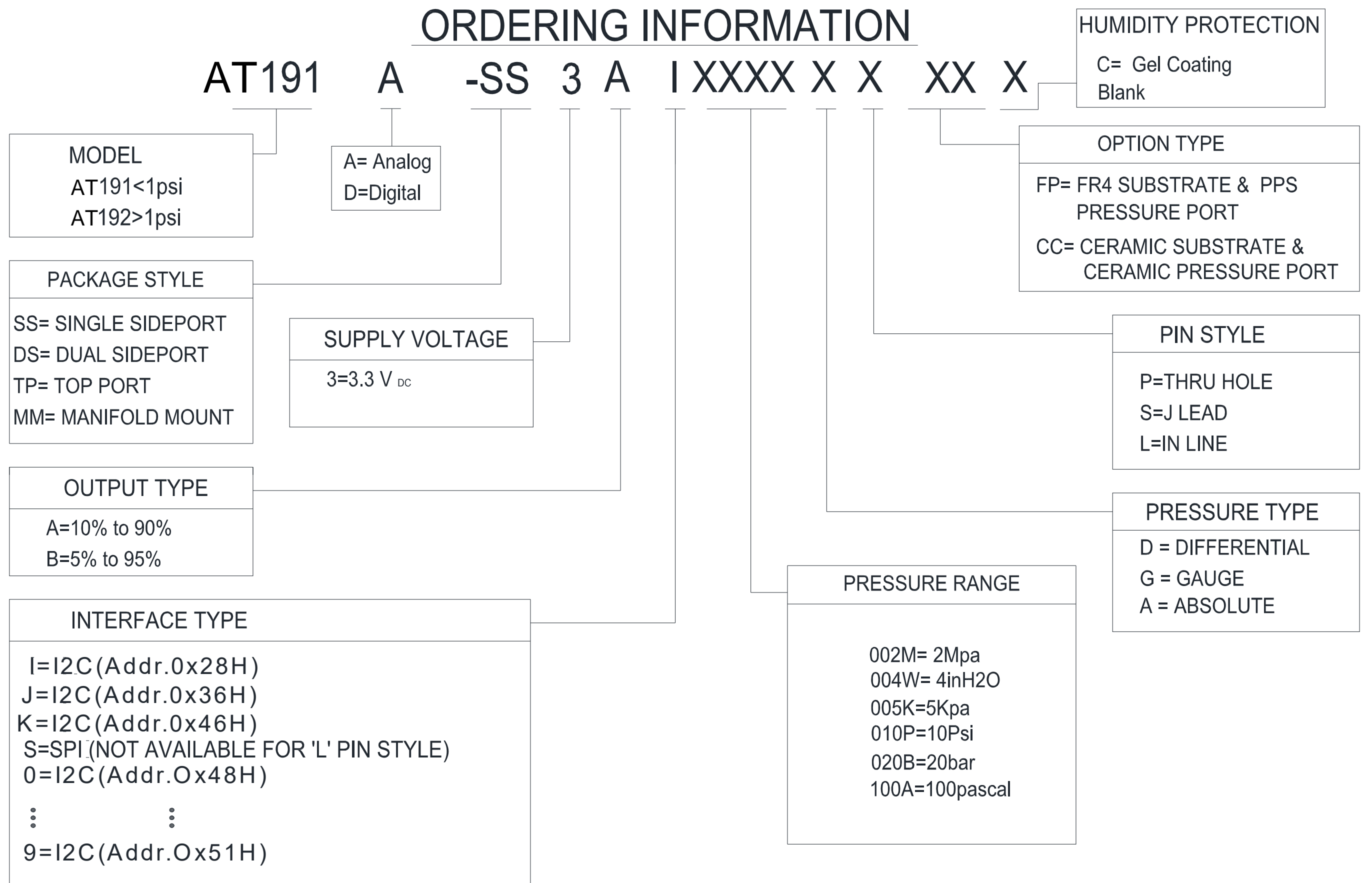
备注

1：差压时，当压力口A的压力大于压力口B时，输出大于输入电压或者16383的50%. 当压力口A的压力等于压力口B时，输出为输入电压或者16383的50%

2：表压时，当压力口A的压力大于压力口B时，输出对于A型，大于输入电压或者16383的10%，对于B型，大于输入电压或者16383的5%

3：压力口A在调试的时候始终加的是正压。

订购信息



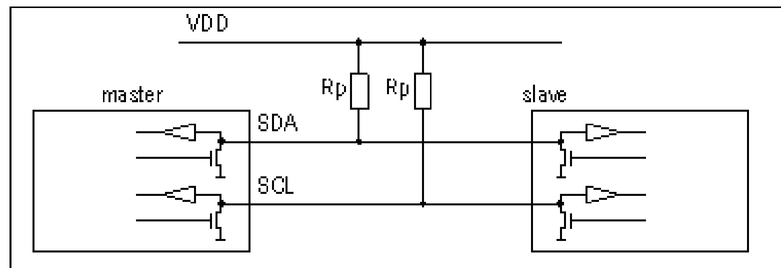
1. I2C Interface Specification

HM19 series is optimized in terms of sensor readout and power consumption when the factory setting for I2C. For detailed specifications of the I2C protocol, see The I2C Bus Specification.

1.1 Interface Connection-External

Bi-directional bus lines are implemented by the devices (master and slave) using open-drain output stages and a pull-up resistor connected to the positive supply voltage. The recommended pull-up resistor value depends on the system setup (capacitance of the circuit or cable and bus clock frequency). In most cases, 4.7kΩ is a reasonable choice. The capacitive loads on SDA and SCL line have to be the same. It is important to avoid asymmetric capacitive loads.

I^2C Transmission start Condition



Both bus lines, SDA and SCL, are bi-directional and therefore require an external pull-up resistor.

1.2 I2C Address

The I2C address consists of a 7-digit binary value. The factory setting for I2C slave address is 0x28, 0x36 or0x46 depending on the interface type selected from the ordering information. The address is always followed by a write bit (0) or read bit (1).The default hexadecimal I2C header for read access to the sensor is therefore 0x51, 0x6D, 0x8D respectively, based on the ordering information.

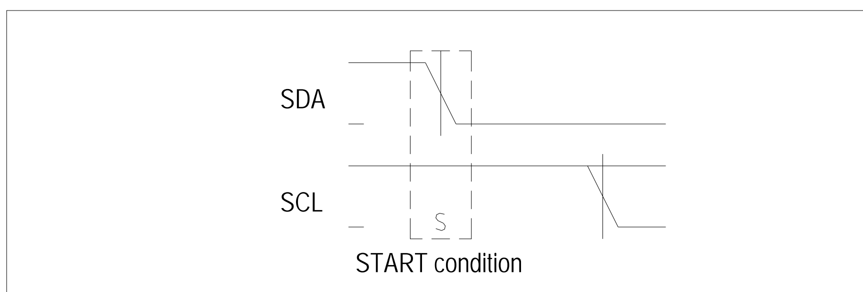
1.3 INT/SS pin

When programmed as an I2C device, the INT/SS pin operates as an interrupt. The INT/SS pin rises when new output data is ready and falls when the next I2C communication occurs.

1.4 Transfer sequences

Transmission START Condition (S): The START condition is a unique situation on the bus created by the master, indicating to the slaves the beginning of a transmission sequence (the bus is considered busy after a START).

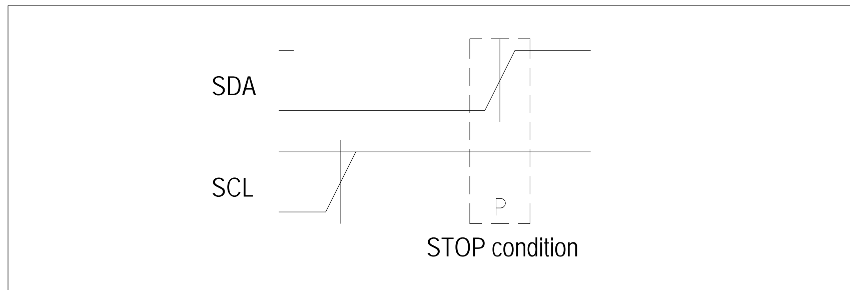
I^2C Transmission start Condition



A HIGH to LOW transition on the SDA line while SCL is HIGH

Transmission STOP Condition (P): The STOP condition is a unique situation on the bus created by the master, indicating to the slaves the end of a transmission sequence (the bus is considered free after a STOP).

I^2C Transmission stop Condition

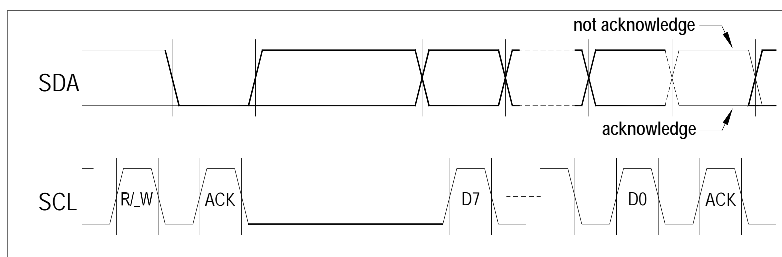


A LOW to HIGH transition on the SDA line while SCL is HIGH

Acknowledge (ACK) / Not Acknowledge (NACK): Each byte (8 bits) transmitted over the I2C bus is followed by an acknowledge condition from the receiver. This means that after the master pulls SCL low to complete the transmission of the 8th bit, SDA will be pulled low by the receiver during the 9th bit time. If after transmission of the 8th bit the receiver does not pull the SDA line low, this is considered to be a NACK condition.

If an ACK is missing during a slave to master transmission, the slave aborts the transmission and goes into idle mode.

I^2C Acknowledge / Not acknowledge



Each byte is followed by an acknowledge or a not acknowledge, generated by the receiver

1.5 Data Transfer Format

Data is transferred in byte packets in the I2C protocol, which means in 8-bit frames. Each byte is followed by an acknowledge bit. Data is transferred with the most significant bit (MSB) first.

A data transfer sequence is initiated by the master generating the Start condition (S) and sending a header byte. The I2C header consists of the 7-bit I2C device address and the data direction bit (R/_W).

The value of the R/_W bit in the header determines the data direction for the rest of the data transfer sequence. If R/_W = 0 (WRITE) the direction remains master-to-slave, while if R/_W = 1 (READ) the direction changes to slave-to-master after the header byte.

1.6 Command Set and data Transfer Sequences

The I2C master command starts with the 7bit slave address with the 8th bit =1 (READ).The sensor as the slave sends an acknowledge (ACK) indicating success. The sensor has four I2C read commands: Read_MR, Read_DF2, Read_DF3, and Read_DF4.Figure 1.6 shows the structure of the measurement packet for three of the four I2C read commands, which are explained in sections 1.6.1 and 1.6.2

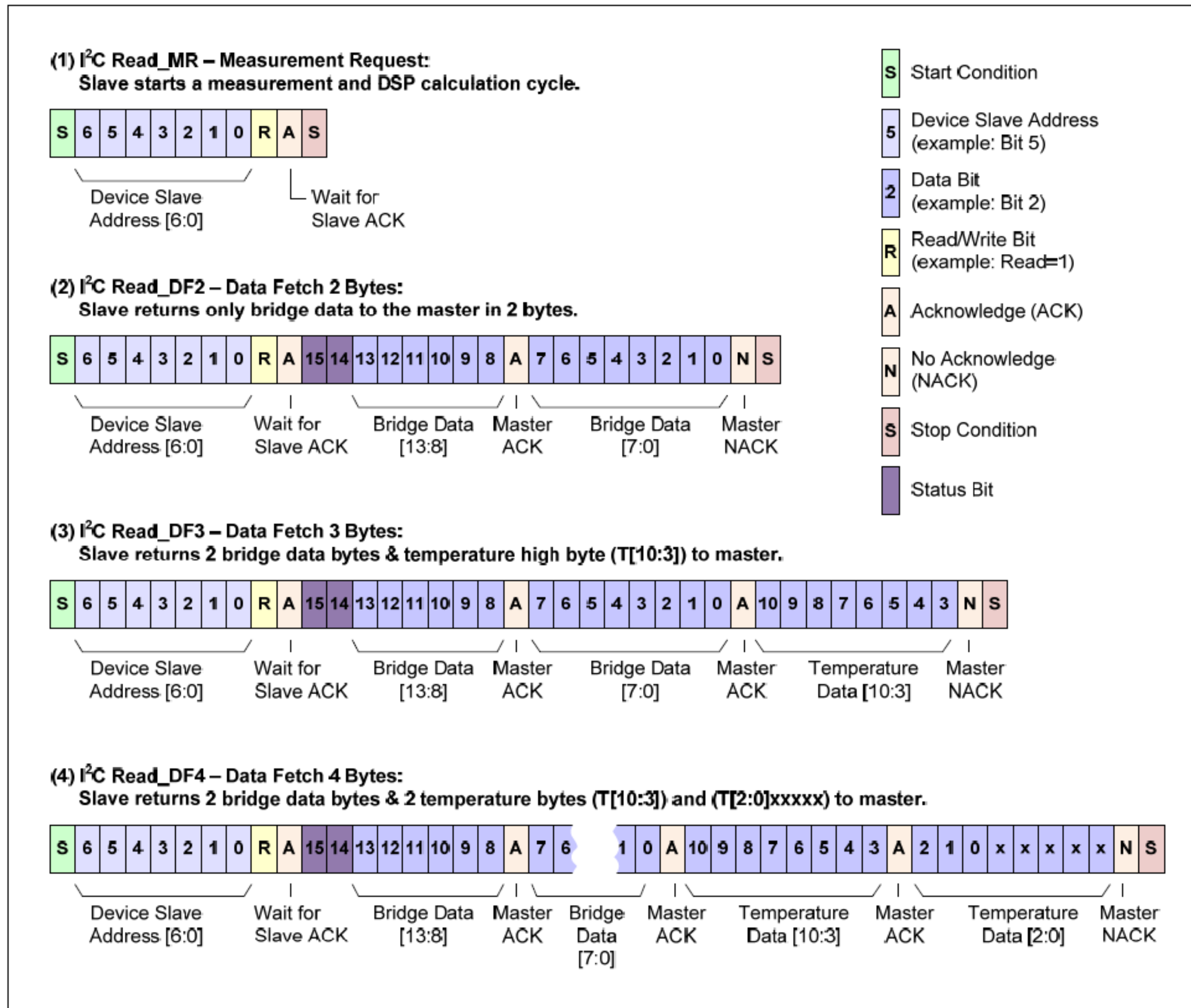


Figure 1.6 – I2C Measurement Packet Reads

I2CRead_DF (Data Fetch)

For Data Fetch commands, the number of data bytes returned by the RBiciLite™ is determined by when the master sends the NACK and stop condition. For the Read_DF3 data fetch command (Data Fetch 3 Bytes; see example 3 in Figure 1.6), the sensor returns three bytes in response to the master sending the slave address and the READ bit (1): two bytes of bridge data with the two status bits as the MSBs and then 1 byte of temperature data (8-bit accuracy). After receiving the required number of data bytes, the master sends the NACK and stop condition to terminate the read operation. For the Read_DF4 command, the master delays sending the NACK and continues reading an additional finalbyte to acquire the full corrected 11-bit temperature measurement. In this case, the last 5 bits of the final byte of the packet are undetermined and should be masked off in the application.The Read_DF2 command is used if corrected temperature is not required. The master terminates the READ operation after the two bytes of bridge data (see example 2 in Figure 1.6).

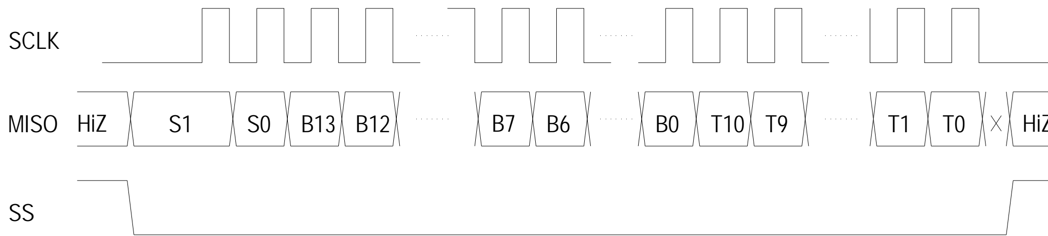
The two status bits (Bit 15 and Bit 14) give an indication of stale or valid data depending on their value. A returned value of 00 indicate “normal operation and a good data packet” while a returned value of 10 indicates “stale data that has been already fetched”. Users that use “status bit” polling should select a frequency slower than 20% more than the response time.

2.SPI interface Specification

The SPI interface of sensor can be programmed for falling-edge MISO change.

2.1 SPI Read_DF (Data Fetch)

For simplifying explanations and illustrations, only falling edge SPI polarity will be discussed in the following sections. The SPI interface will have data change after the falling edge of SCLK. The master should sample MISO on the rise of SCLK. The entire output packet is 4 bytes (32 bits). The high bridge data byte comes first, followed by the low bridge data byte. Then 11 bits of corrected temperature (T[10:0]) are sent: first the T[10:3]byte and then the {T[2:0],xxxxx} byte. The last 5 bits of the final byte are undetermined and should be masked off in the application. If the user only requires the corrected bridge value, the read can be terminated after the 2nd byte. If the corrected temperature is also required but only at an 8-bit resolution, the read can be terminated after the 3rd byte is read.



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)
T(2:0),xxxxx =. Remaining bits of corrected temperature data for full 11-bit resolution
HiZ = High impedance

Figure 2.1 – SPI Output Packet with Falling Edge SPI_Polarity

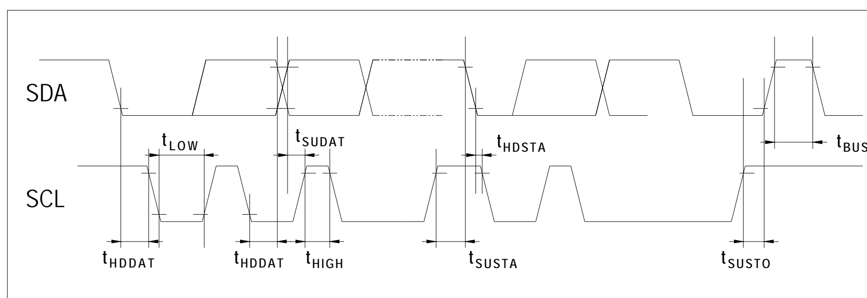
TIMING DIAGRAM

I2C INTERFACE PRRAMETERS

PARAMETERS	SYMBOL	MIN	TYP	MAX	UNITS
SCLK CLOCK FREQUENCY	FSCL	100		400	KHz
START CONDITION HOLD TIME RELATIVE TO SCL EDGE	tHDSTA	0.1			uS
MINIMUM SCL CLOCK LOW WIDTH @1	tLOW	0.6			uS
MINIMUM SCL CLOCK HIGH WIDTH @1	tHIGH	0.6			uS
START CONDITION SETUP TIME RELATIVE TO SCL EDGE	tSUSTA	0.1			uS
DATA HOLD TIME ON SDA RELATIVE TO SCL EDGE	tHDDAT	0			uS
DATA SETUP TIME ON SDA RELATIVE TO SCL EDGE	tSUDAT	0.1			uS
STOP CONDITION SETUP TIME ON SCL	tSUSTO	0.1			uS
BUS FREE TIME BETWEEN STOP AND START CONDITION	tBUS	2			uS

@1 COMBINED LOW AND HIGH WIDTHS MUST EQUAL OR EXCEED MINIMUM SCL PERIOD.

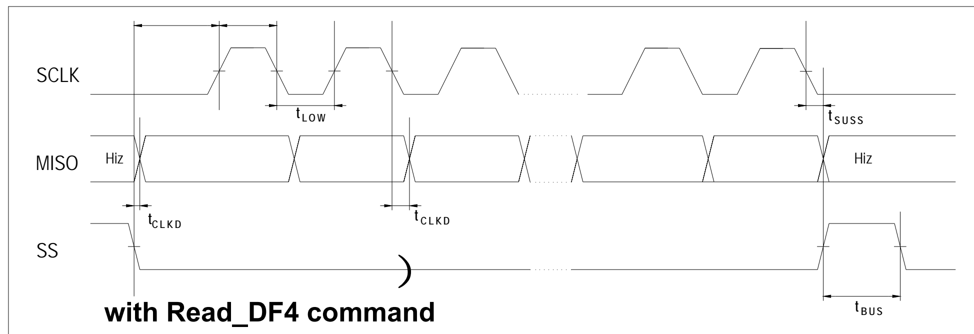
I2C TIMING DIAGRAM



SPI INTERFACE PRRAMETERS

PARAMETERS	SYMBOL	MIN	TYP	MAX	UNITS
SCLK CLOCK FREQUENCY	FSCL	50		800	KHz
SS DROP TO FIRST CLOCK EDGE	tHDSS	2.5			uS
MINIMUM SCL CLOCK LOW WIDTH @1	tLOW	0.6			uS
MINIMUM SCL CLOCK HIGH WIDTH @1	tHIGH	0.6			uS
CLOCK EDGE TO DATA TRANSITION	tCLKD	0		0.1	uS
RISE OF SS RELATIVE TO LAST CLOCK EDGE	tSUSS	0.1			uS
BUS FREE TIME BETWEEN RISE AND FALL OF SS	tBUS	2			uS

SPI TIMING DIAGRAM



I2C/SPI PROGRAMING SAMPLE

I2C （HM191D-DS-3BI002WDP

C code example for I2C

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(byte);             /*send address or command MSB first*/
GetOneByte();                      /*read one byte from serial stream */
SendStop();                 /*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 1 */
      byte = byte << 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 last byte */
  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) + read))                /*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
```

SPI （HM191D-DS-3BS002WDP）

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;

}

void BitDelay(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;
```