

WPS16M Series Pressure Sensor

Product introduction

WPS16M Series Basic Board Mount Pressure Sensors are piezoresistive silicon SOIC16 packaged high-precision high-performance 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 pressure measurement and can be directly mounted on standard printed circuit boards. The WPS16M 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.



Features

· I2C/SPI or analog output	· Accuracy: $\pm 0.25\%FSS$	· Total Error Band (TEB): Optimum $\pm 1.0\%$
· Operating temperature $-20 \sim +85^{\circ}\text{C}$	· Differential type	· Power supply: 3.3V / 5V
· Barb shaped pressure port	· Sleep mode	· 14 bit resolution

Application

Industrial automation

Leak testing

Medical equipment

HAVC

Product specification

Absolute Maximum Ratings

Parameter	Minimum	Maximum	Unit
Supply voltage	-0.3	6	Vdc
Voltage on any pin	-0.3	$V_{DD} + 0.3$	V
Digital interface clock frequency:			
I2C	100	400	kHz
SPI	50	800	
ESD susceptibility (human body model)	4		kV
Storage temperature	-40	125	$^{\circ}\text{C}$
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	

Operating Parameter

Parameter	Minimum	Typical	Maximum	Unit
Power supply: 3.3 Vdc 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
Operating temperature	-20	-	85	°C
Startup time (from power-up to data-ready)	-	2.8	7.3	ms
Response time	-	0.5	-	ms
Low level voltage	-	-	0.2	Vsupply
High level voltage	0.8	-	-	Vsupply
load resistance	1	4.7	-	kOhm
Accuracy	-0.25	-	0.25	%FSS BFSL
Resolution ratio	-	14	-	bit
Default communication address	0X28			

Notes:

- The Absolute Maximum Ratings is the maximum limit that the device can withstand without damage to the sensor.
- 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.

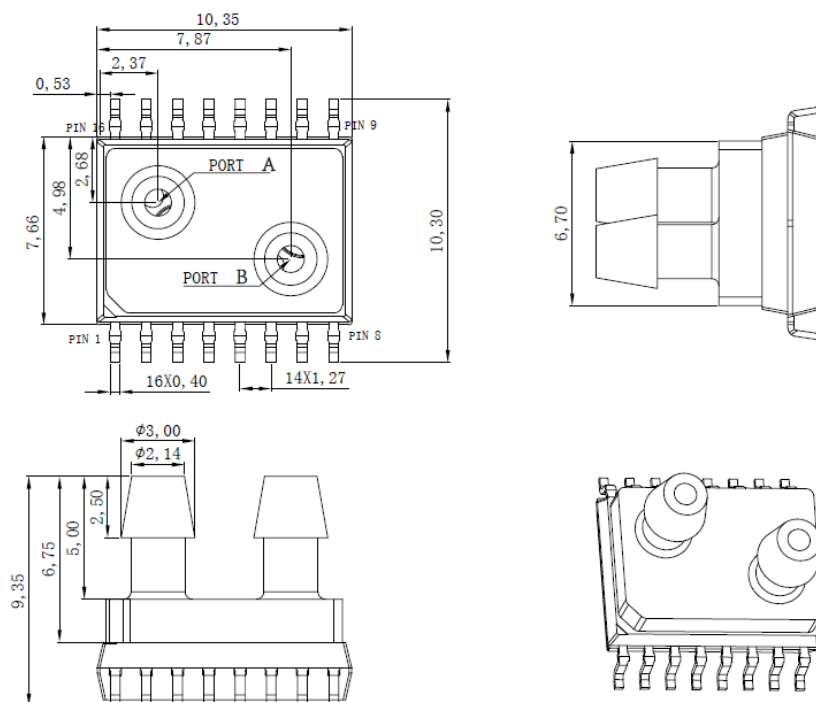
Environmental Specifications

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

Life may vary depending on specific application .

Structure Parameter

Unit: (mm)



Note: PORT A: the reference port

B : the primary input port PORT

Pinouts

	PIN 1-5	PIN 6	PIN 7	PIN 8-9	PIN 10	PIN 11	PIN 12	PIN 13-16
I2C	NC	GND	VDD	NC	SDA	SCL	INT	NC
SPI	NC	GND	VDD	NC	MISO	SCLK	SS	NC
Analog	NC	GND	NC	NC	OUTPUT	VDD	NC	NC

Ordering Information

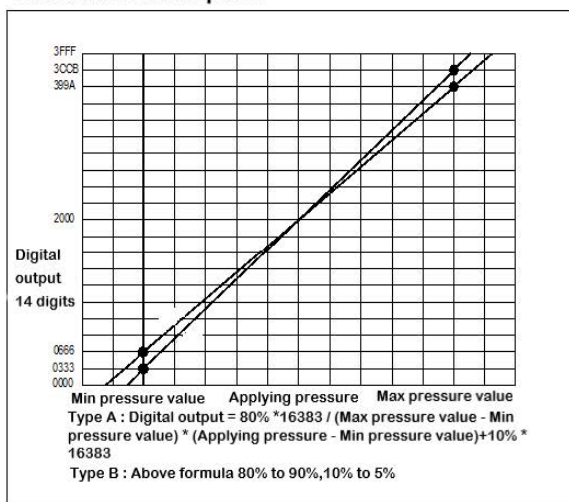
For example: WPS 16M 001 D 10 C DN 3 H
 (1) (2) (3) (4) (5) (6) (7) (8) (9)

Series number	Significance	Description
1	Product series	WPS
2	Package	16 M: 16 PIN SMT
3	Pressure range	500P(500PA) 001K(1KPA) 004K(4KPA) 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	D: Differential
5	TEB Range	10: $\pm 1.0\%$ FSS 15: $\pm 1.5\%$ FSS 20: $\pm 2.0\%$ FSS
6	Output Type	A: Analog C: I ² C D: SPI
7	Pressure Port	DN: Double axial barbed port
8	Power supply	3: 3.3VDC 5: 5.0VDC
9	Compensation Temperature	H: -20 C°~85C° M: 0 C°~85C° L: 0 C°~50C°

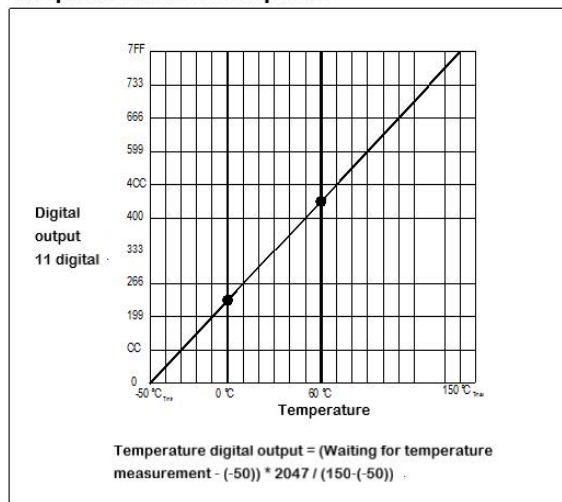
Pressure and Temperature Transfer Function

I2C/SPI Output (14 bits resolutions)

Pressure conversion equation



Temperature Conversion equation



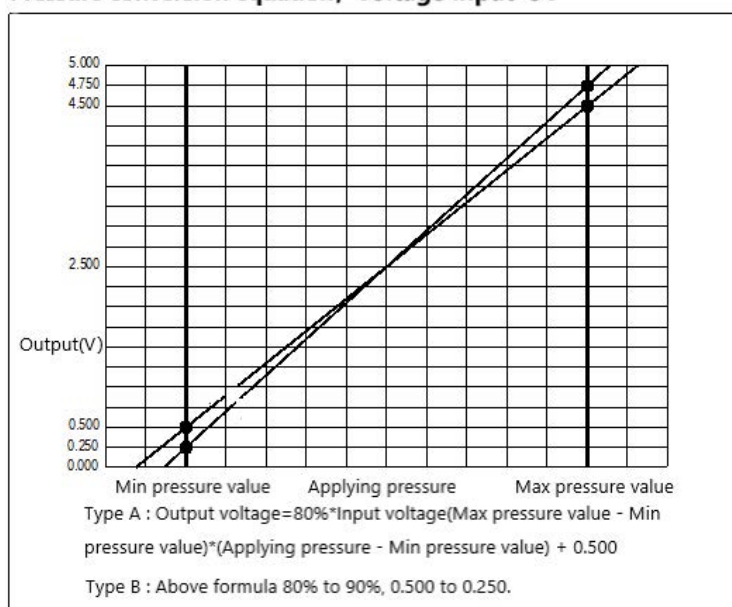
Unless otherwise specified, the products are all pressure transfer function TYPE-A sensors.

Sensor Output at Significant Percentages

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

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

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

Note:

- It is recommended to place the pressure port B 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 .

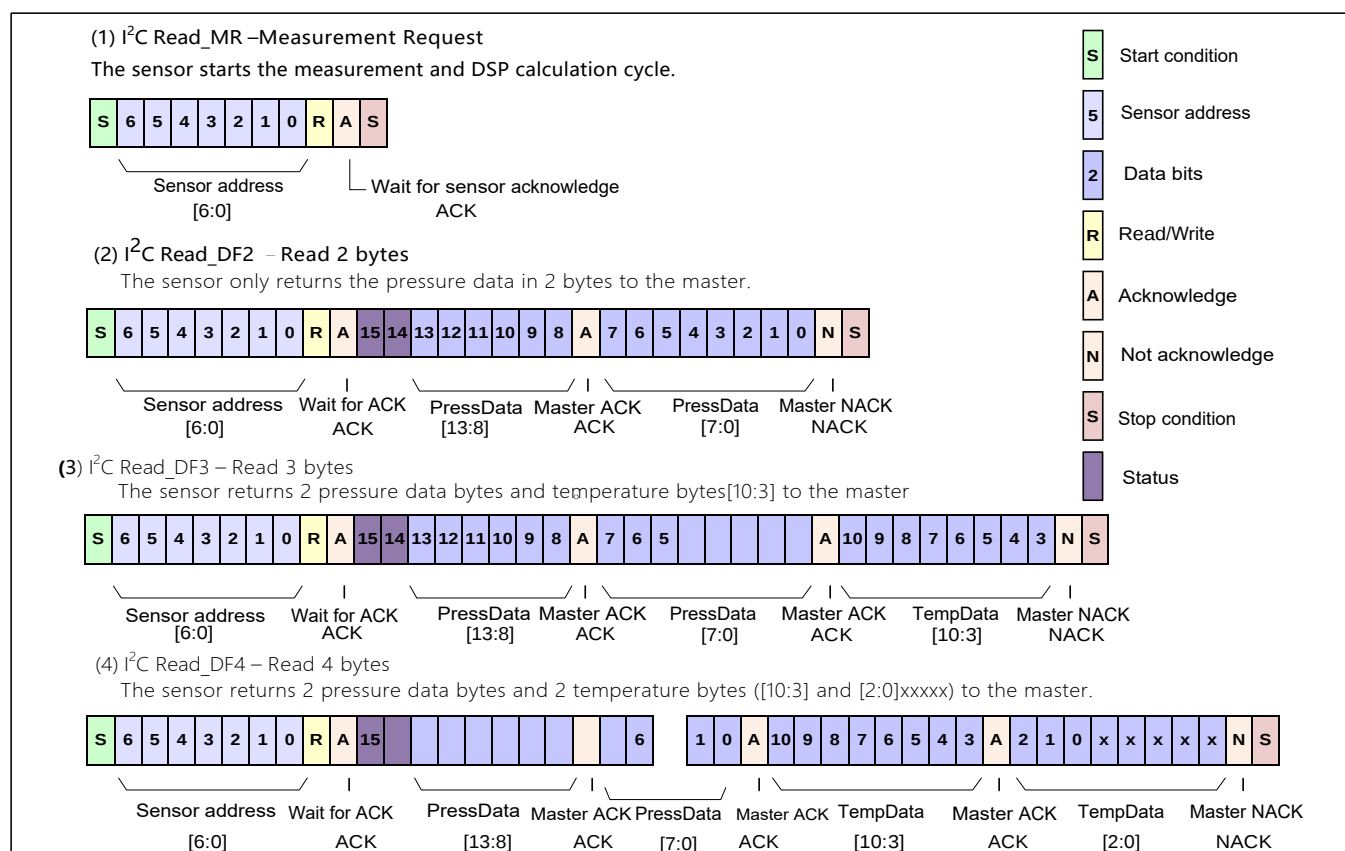
I2C Communication

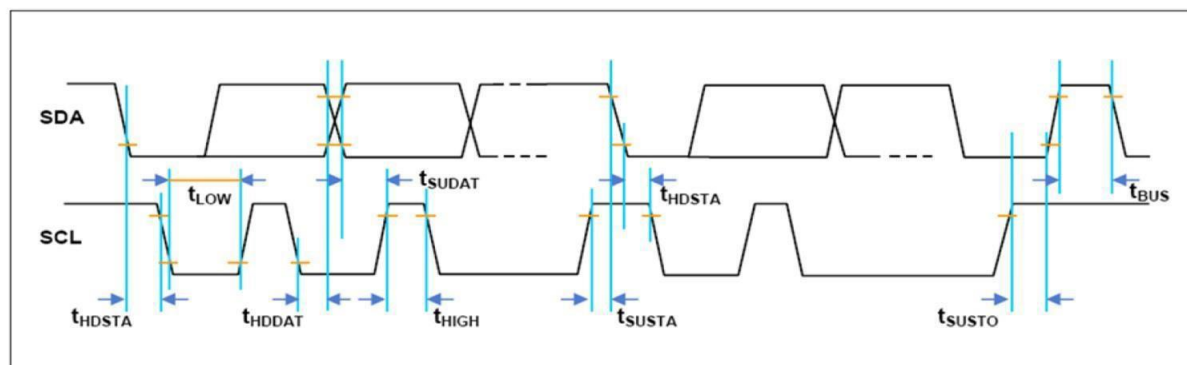
Read Operation

- 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.

I2CREAD:

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	F _{SCL}	100		400	kHz
Start condition hold time relative to SCL edge	t _{HDSTA}	0.1			μs
Minimum SCL clock low width	t _{LOW}	0.6			μs
Minimum SCL clock high width	t _{HIGH}	0.6			μs
Start condition setup time relative to SCL edge	t _{SUSTA}	0.1			μs
Data hold time on SDA relative to SCL edge	t _{HDDAT}	0.0			μs
Data setup time on SDA relative to SCL edge	t _{SUDAT}	0.1			μs
Stop condition setup time on SCL	t _{SUSTO}	0.1			μs
Bus free time between stop condition and start condition	t _{BUS}	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(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
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 */
    unsigned 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)+read)) /*send salve address*/
```

```
{
```

```
    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=1638.3;          /* P1= 10% * 16383 – A type*/

    float P2=14744.7;        /* P2= 90% * 16383 – A type*/

    float Pmax=2.0;

    floatPmin=-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]>>5);

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

SPI Read_DF(Data Reading)

For simplicity of explanation and illustration, the following sections describe only the falling edge SPI polarity.

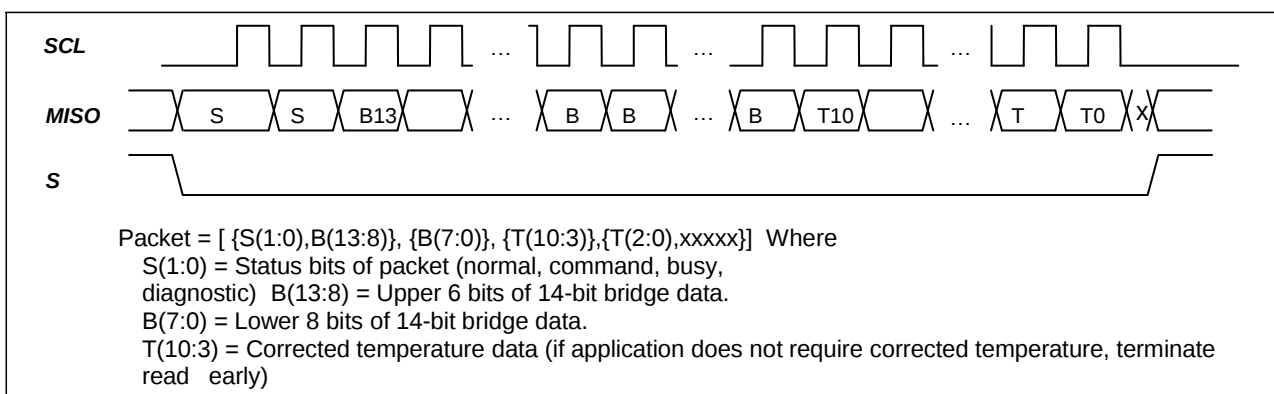
The SPI interface will have data changes after the falling edge of SCLK. Here we take MISO as an example to introduce the rise of SCLK.

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



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 */

/* 0 0 0 0 0 0 1 0 */

/* 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= 1638.3;           /* P1= 10% * 16383 – A type*/
float P2= 14744.7;         /* P2= 90% *16383 – A 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]>>5);

        Pressure= (((float) Dpressure)-P1) * (Pmax-Pmin) / (P2-P1)+Pmin;
        Temperature= ((float) Dtemperature) * 200 / 2047-50;
```