

MSP100

Pressure Transducer

SPECIFICATIONS

- **Analog and 14-Bit Digital Output**
- **Small Size**
- **Low Cost**
- **316L Stainless Steel or 17-4PH**

The MSP100 pressure transducer provides stainless steel media compatibility in a low cost, small profile solution. This sensor has no silicone gel or polymeric media isolation methods to fail in contact with water or other harsh chemicals. Pressure connections are provided via an O-ring seal. The device is available in both analog and 14-bit digital output with a port material of either 316L SS or 17-4PH. Additional custom port options available to meet your application needs. The small size vs. performance and media compatibility are provided through solid-state technology.

FEATURES

- Single Piece Construction; No Welds, No Oil
- 100% Stainless Steel Isolation for Harsh Chemical Measurement
- Low Cost
- 14-Bit Digital Output or Analog

APPLICATIONS

- Beverage Dispensing Systems
- Water Pressure or Flow Monitor
- Medical Equipment
- Industrial Equipment/Hydraulics
- Tank Level Measurement
- Manifold Pressure

STANDARD RANGES

Range	psig
0 to 100	•
0 to 150	•
0 to 250	•
0 to 500	•

PERFORMANCE SPECIFICATIONS (ANALOG, OUTPUT SIGNAL "2")

Ambient Temperature: 25°C (unless otherwise specified)

PARAMETERS	MIN	TYP	MAX	UNITS	NOTES
Supply Voltage	4.75	5.00	5.25	V_{DC}	
Zero Offset	-2		2	mV	Ratiometric
Span	98	100	102	mV	Ratiometric
Current Consumption			2	mA	
Proof Pressure	1.5X			Rated	
Burst Pressure	3X			Rated	
Endurance	1E+6			0~FS Cycles	
Accuracy	-0.5	±0.2	0.5	%Span	RSS of BFSL: Linearity, Hysteresis, Repeatability
Long Term Stability		0.25		%Span	
Minimum Resistance between Transducer and Body	50			MΩ	@250V_{DC}
Thermal Zero Shift	-2.0		2.0	%Span	Reference to 25°C over Compensated Temperature
Thermal Span Shift	-2.0		2.0	%Span	Reference to 25°C over Compensated Temperature
Compensation Temperature	0		45	°C	
Operating Temperature	0		55	°C	
Storage Temperature	-20		85	°C	
Response Time (10% to 90%)		0.1		ms	
Vibration	±20g MIL-STD-810C, Procedure 514.2, Figure 514.2-2, Curve L				
Shock	50g, 11 msec half sine shock per mil standard 202F. Method 213B, Condition A				

PERFORMANCE SPECIFICATIONS (DIGITAL, OUTPUT SIGNAL “J” OR “S”)

Ambient Temperature: 25°C (unless otherwise specified)

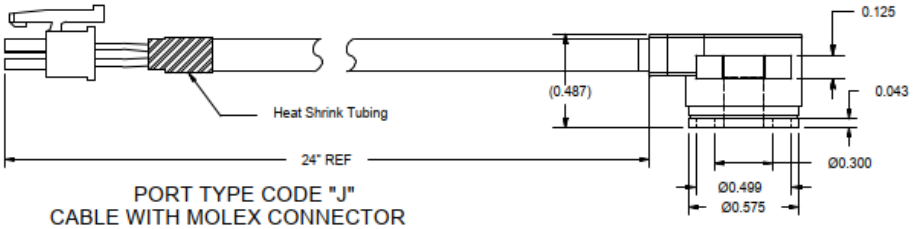
PARAMETERS	MIN	TYP	MAX	UNITS	NOTES
Supply Voltage	2.7	3.0	5.0	V_{DC}	
Output at Zero Pressure	720	1000	1280	Count	
Output at FS Pressure	14,720	15,000	15,280	Count	
Current Consumption			3	mA	
Current Consumption (Sleep mode)			5	uA	
Proof Pressure	1.5X			Rated	
Burst Pressure	3X			Rated	
Endurance	1E+6			0~FS Cycles	
Accuracy	-0.5		0.5	%Span	RSS of BFSL: Linearity, Hysteresis, Repeatability
A/D Resolution		14		Bit	
Operating Temperature	0		55	°C	
Temperature Accuracy	-3		3	°C	1*
Thermal Zero Shift	-2.0		2.0	%F.S.	Reference to 25°C over Compensated Temperature
Thermal Span Shift	-2.0		2.0	%F.S.	Reference to 25°C over Compensated Temperature
Compensated Temperature	0		45	°C	
Operating Temperature	0		55	°C	
Storage Temperature	-20		85	°C	
Response Time (10% to 90%)			3	ms @ 4MHz	Without Sleep Mode
Response Time (10% to 90%)			8.4	ms @ 4MHz	With Sleep Mode
Vibration	±20g MIL-STD-810C, Procedure 514.2, Figure 514.2-2, Curve L				
Shock	50g, 11 msec half sine shock per mil standard 202F. Method 213B, Condition A				

Notes:

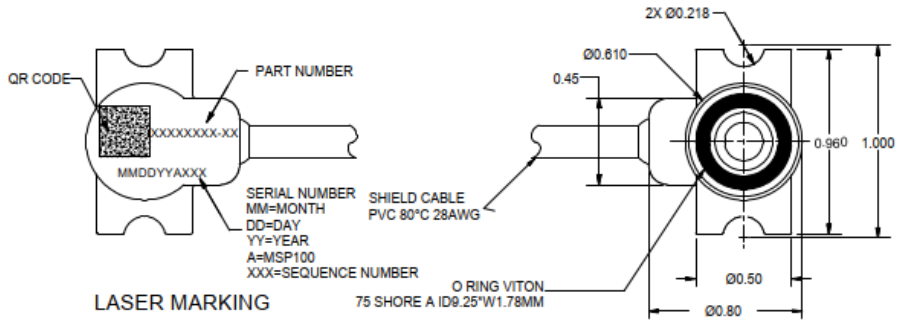
1* Reflect pressure port diaphragm temperature over the compensated temperature range

2* Response time is from power on to reading measurement data.

DIMENSIONS

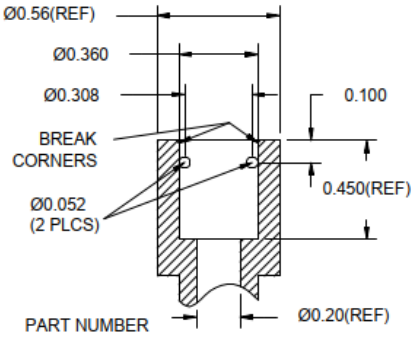


PORT TYPE CODE "J"
CABLE WITH MOLEX CONNECTOR

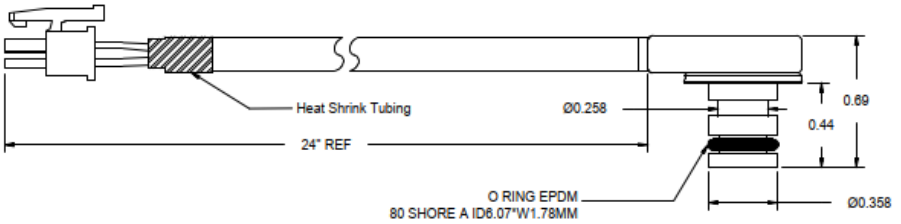


LASER MARKING

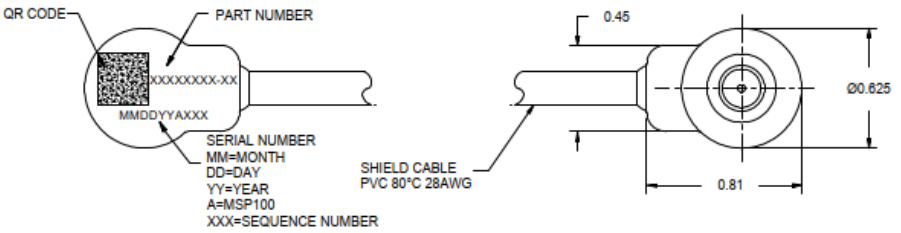
TYPICAL RECEIVER PORT
ONLY FOR U TYPE PORT



DESIGNED TO SEAL WITH -010
FOOD GRADE VITON O-RING



PORT TYPE CODE "U"
CABLE WITH MOLEX CONNECTOR



LASER MARKING

WIRING

Analog mV Output Wiring

Connection	PIN 1	PIN 2	PIN 3	PIN 4
Molex 4pin Connector PCB Mount	+SUPPLY	+OUTPUT	-OUTPUT	-SUPPLY

4 PINS MOLEX CONNECTOR
HOUSING:MOLEX 430-25-040
PIN:MOLEX 430-30-004



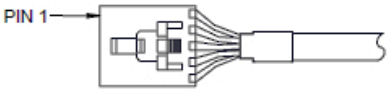
Digital I²C Output Wiring

Connection	PIN 1	PIN 2	PIN 3	PIN 4
Molex 4pin Connector PCB Mount	VDD	GND	SDA	SCL

Digital SPI Output Wiring

Connection	PIN 1	PIN 2	PIN 3	PIN 4	PIN5
Molex 5pin Connector PCB Mount	VDD	GND	MISO	SCLK	SS

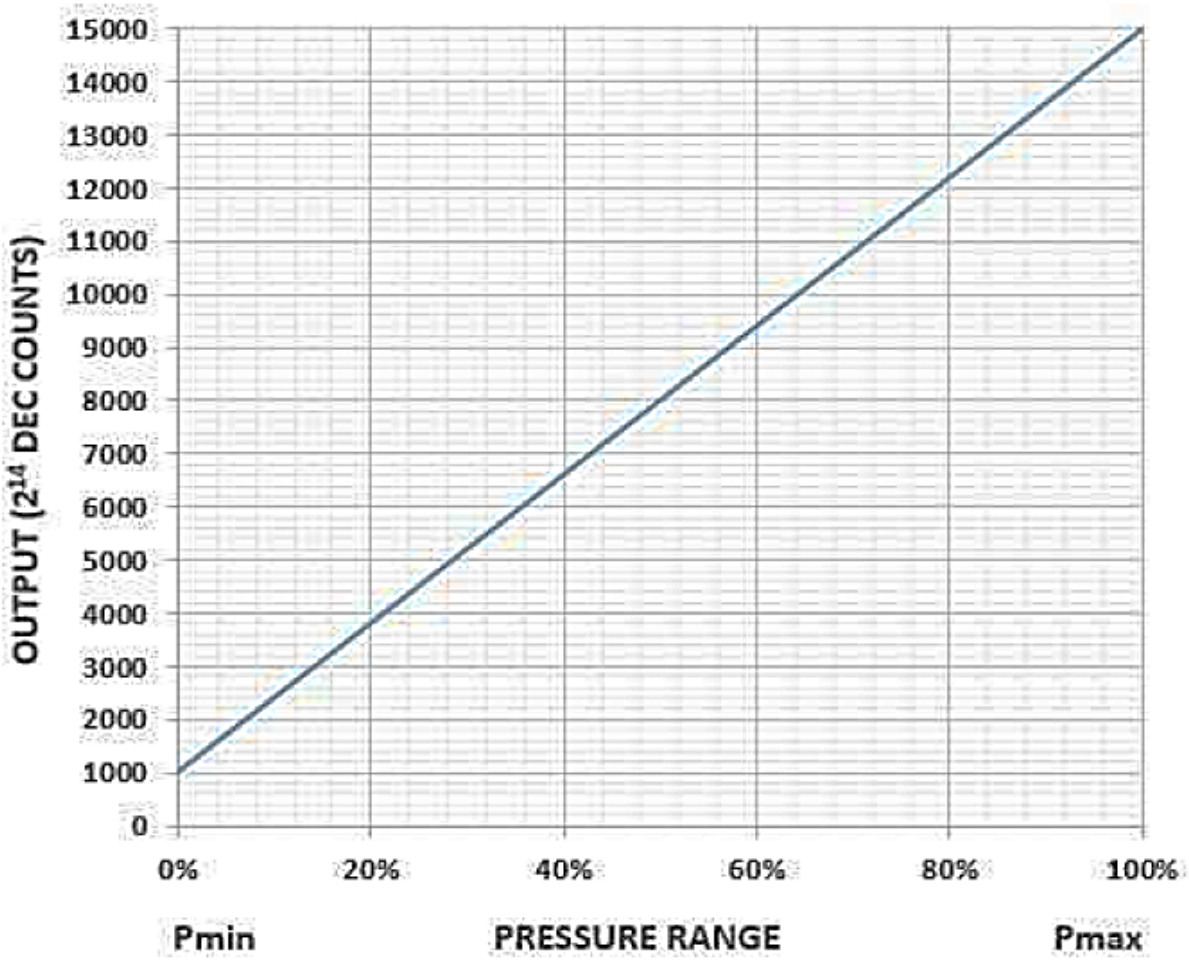
5 PINS MOLEX CONNECTOR
HOUSING:MOLEX 50-57-9405
PIN:MOLEX 16-02-0082



SENSOR OUTPUT

SENSOR OUTPUT AT SIGNIFICANT PERCENTAGES

% OUTPUT	DIGITAL COUNTS (DECIMAL)	DIGITAL COUNTS (HEX)
0%	1000	0 × 3E8
5%	1700	0 × 6A4
10%	2400	0 × 960
50%	8000	0 × 1F40
90%	13600	0 × 3520
95%	14300	0 × 37DC
100%	15000	0 × 3A98

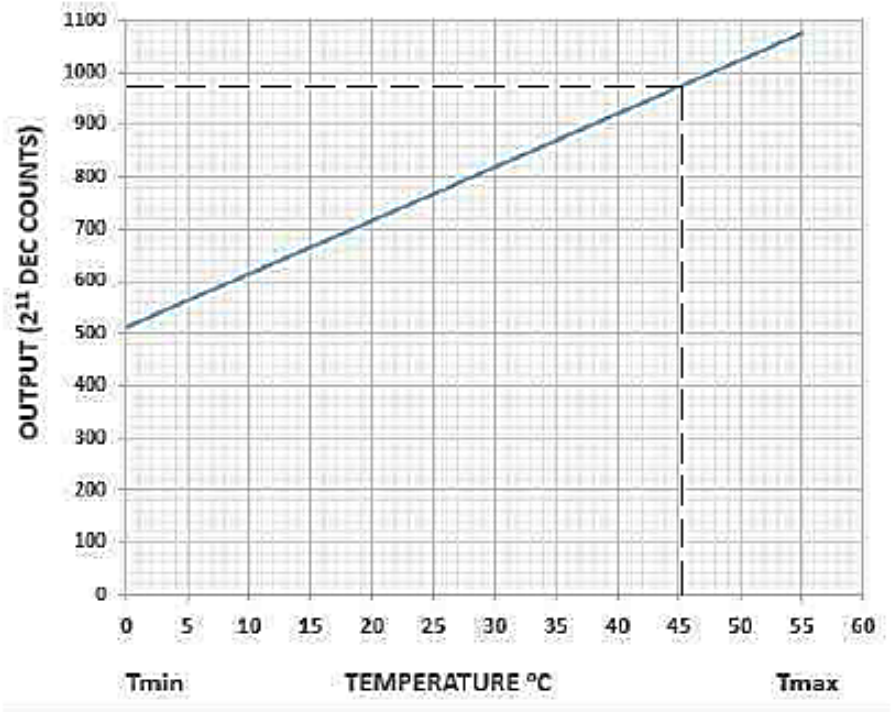


$$\text{OUTPUT (DECIMAL COUNTS)} = \frac{15000\text{-}1000}{\text{Pmax - Pmin}} \times (\text{Papplied - Pmin}) + 1000$$

TEMPERATURE OUTPUT

TEMPERATURE OUTPUT

OUTPUT °C	DIGITAL COUNTS (DECIMAL)	DIGITAL COUNTS (HEX)
0	512	0 × 200
10	614	0 × 266
25	767	0 × 2FF
40	921	0 × 399
55	1075	0 × 433



$$\text{OUTPUT (DECIMAL COUNTS)} = \frac{(\text{OUTPUT°C}+50°\text{C})\times 2048}{150℃-(-50℃)}$$

OUTPUT SIGNAL

Code	Output Signal	Supply Voltage (V)
2	0 – 100mV	5 ± 0.25
J	I^2C	2.7 – 5.0
S	SPI	2.7 – 5.0

ORDERING INFORMATION

MS1 S 1 – 1 0 0 0 0 J- 100P G

Output		
Code	**Type**	**Supply Voltage**
2	0 - 100mV	5±0.25V
J*	I^2C	2.7 – 5.0V
S*	SPI	2.7 – 5.0V

*Digital Output

Pressure Type	
G	Gauge

Pressure Ranges
psi
100P
150P
250P
500P

Cable/Connector	
Code	**Connection Type**
1	Cable, 2 feet with Molex Connector

Port Type Selection	
Code	**Port Type**
J	O-ring Face Seal
U	O-ring Radial Seal

Port Material	
0	316L Stainless Steel
1	17-4PH Stainless Steel

I^2C Address (Digital Only)	
Code	Address
0	0x28H
1	0x36H
2	0x46H
3	0x48H
4	0x51H

SPI Default Code “0”

Cleaning	
0	No Selection
1	Oxygen Clean B40.1 Level IV

Sleep (Digital Only)	
0	Non-Sleep Mode
1	Sleep mode

INTERFACING TO TE DIGITAL PRESSURE MODULES

The TE series of digital pressure sensors uses the latest CMOS sensor conditioning circuitry (SSC) to create a low cost, high performance digital output pressure (14-bit) and temperature (11-bit) sensor designed to meet the strictest requirements from OEM customers.

The MS45x5DO, 85BSD, 85FBSD, 86BSD,154BSD, MSP100(DO) and MSP300(DO) , M3200(DO), FX29(DO) and FS30(DO)are the latest offering from TE to offer digital communication to pressure sensor OEMs.

I²C AND SPI INTERFACE SPECIFICATIONS

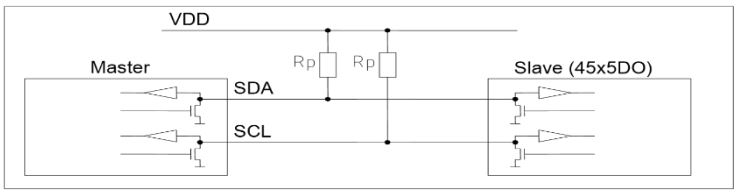
1. I²C Interface Specification

The I²C interface is a simple 8-bit protocol using a serial data line (SDA) and a serial clock line (SCL) where each device connected to the bus is software addressable by a unique address. For detailed specifications of the I²C protocol, see The I²C Bus Specification, Version 2.1, January 2000.

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.

I²C Transmission Start Condition



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

1.2 I²C Address

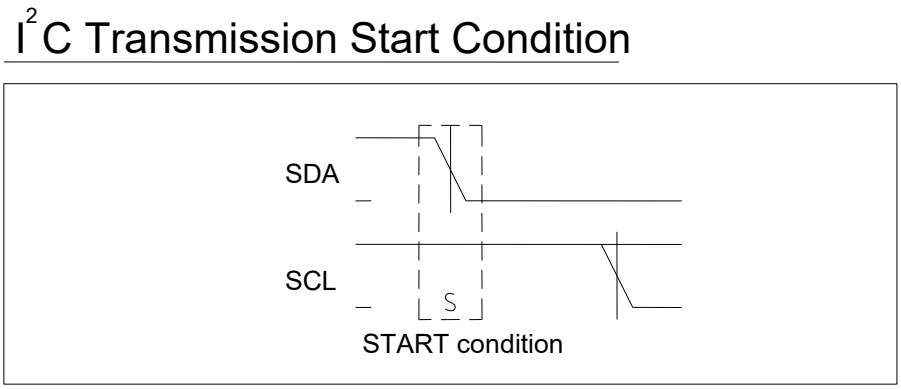
The I²C address consists of a 7-digit binary value. The factory setting for the I²C slave address is 0x28, 0x36 or 0x46 depending on the interface type selected from the ordering information. The address is always followed by a write bit (0) or read bit (1). The default hexadecimal I²C header for read 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.

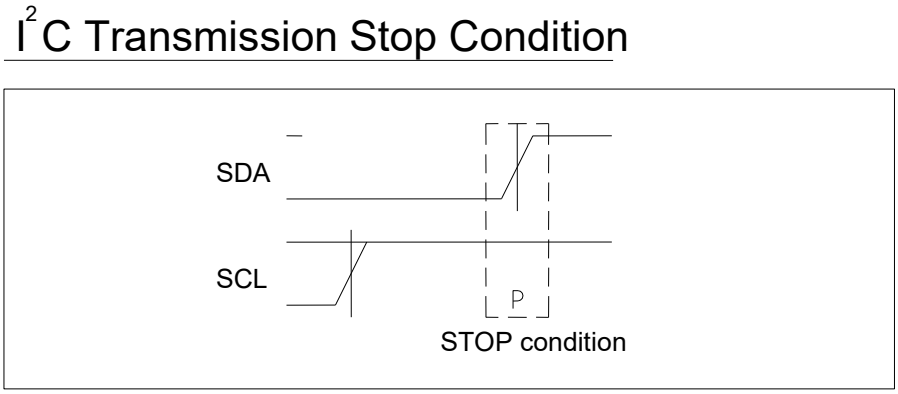
1.4 Transfer Sequences

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).



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).

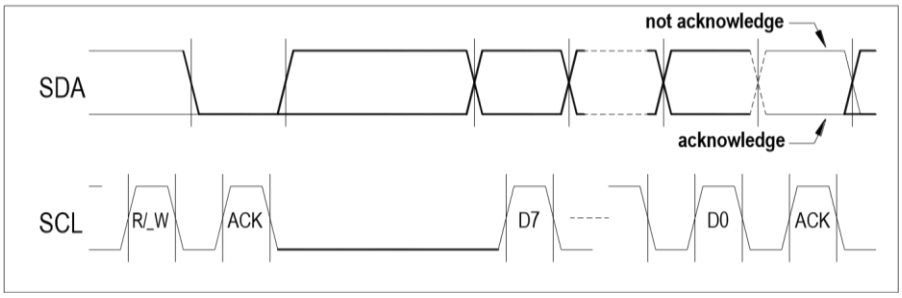


A LOW to HIGH transition on the SDA line while SCL is HIGH

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.

I² C 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 7-bit slave address with the 8th bit = 1 (READ). The sensor acts as the slave and 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 of the four I²C read commands, which are explained in sections 1.6.1.

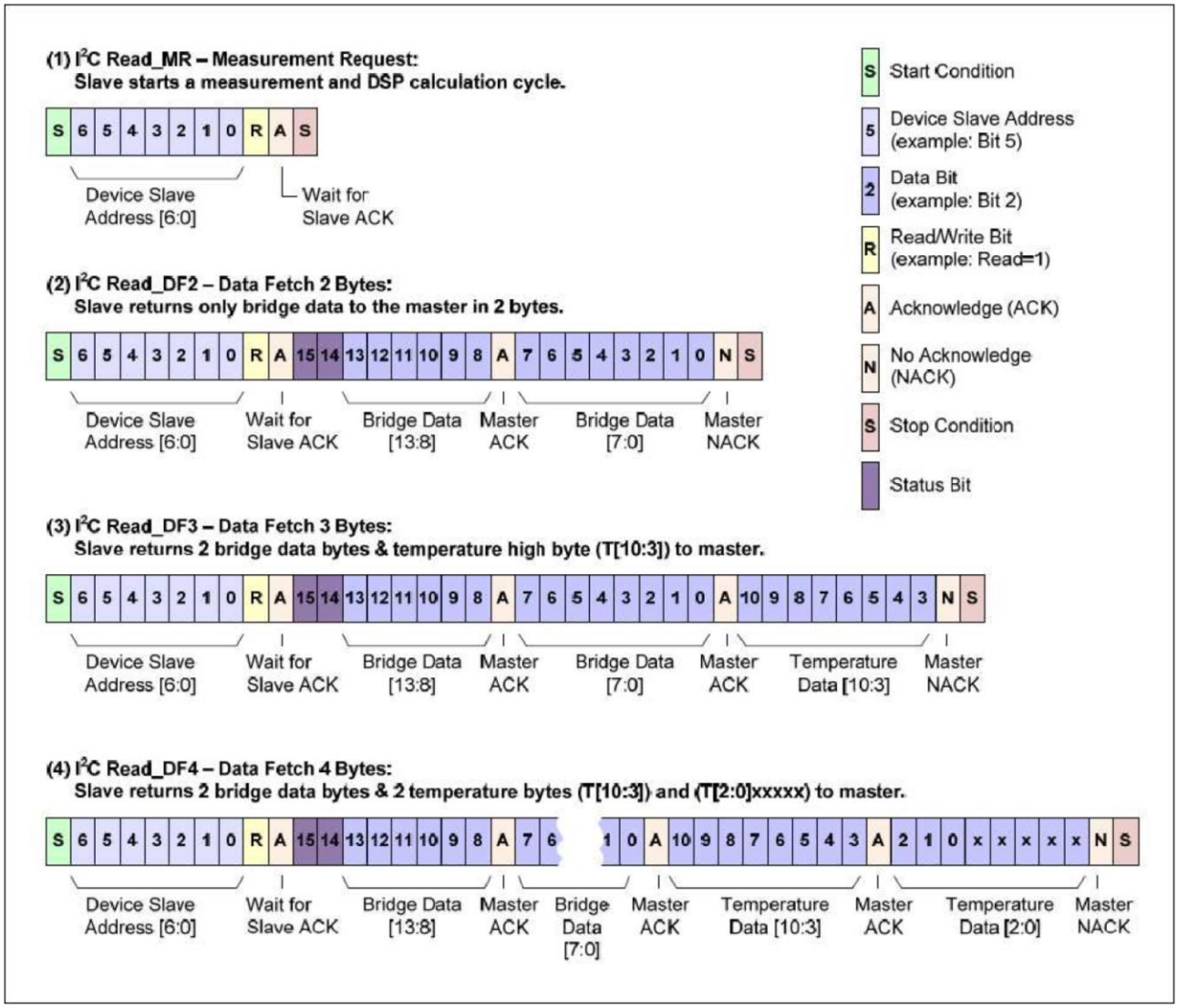


Figure 1.6 – I²C Measurement Packet ReadsI²C Read_DF (Data Fetch)

1.6.1

For Data Fetch commands, the number of data bytes returned by the sensor, is determined 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 final byte 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”. See section 1.7 for additional details. Users that use “status bit” polling should select a frequency slower than 20% more than the update time.

1.7 Status Bits and Diagnostic Features

The table below summarizes the status bits conditions indicated by the 2 MSBs (Bit (15:14) of I^2C data packet, S(1:0) of SPI data packet of the bridge high byte data.

Table 1: Status Bits Encoding

Status Bits (2 MSB of Output Data Packet)	Definition
00	Normal Operation. Good Data Packet
01	Reserved
10	Stale Data. Data has been fetched since last measurement cycle.
11	Fault Detected

The SSC is has on board diagnostic features to ensure robust system operation in the most “mission-critical” applications. A status bit value of “11” indicates a fault condition in the SSC or sensing element. All diagnostics are detected in the next measurement cycle and reported in the subsequent data fetch. Once a diagnostic is reported, the diagnostic status bits will not change unless both the cause of the diagnostic is fixed and a power-on-reset is performed.

1.8 I^2C Protocol Differences

There are three differences in the described above protocol compared with original I^2C protocol:

- Sending a start-stop condition without any transitions on the SCL line (no clock pulses in between) creates a communication error for the next communication, even if the next start condition is correct and the clock pulse is applied. An additional start condition must be sent, which results in restoration of proper communication.
- The restart condition – a falling SDA edge during data transmission when the SCL clock line is still high – creates the same situation. The next communication fails, and an additional start condition must be sent for correct communication.
- A falling SDA edge is not allowed between the start condition and the first rising SCL edge. If using an I^2C address with the first bit 0, SDA must be held down from the start condition through the first bit.

2. SPI Interface Specification

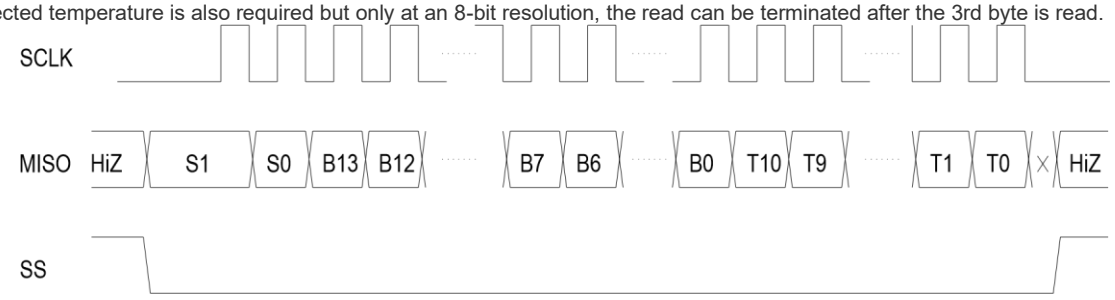
SPI is a general-purpose synchronous serial interface. During an SPI transfer, transmit and receive data is simultaneously shifted out and in serially. A serial clock line synchronizes the shifting and sampling of the information on two serial data lines.

SPI devices communicate using a master-slave relationship. Due to its lack of built-in device addressing, SPI requires more effort and more hardware resources than I^2C when more than one slave is involved. But SPI tends to be simpler and more efficient than I^2C in point-to-point (single master, single slave) applications for the very same reason; the lack of device addressing means less overhead.

The SPI interface is programmed for falling-edge MISO change.

2.1 SPI Read_DF (Data Fetch)

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

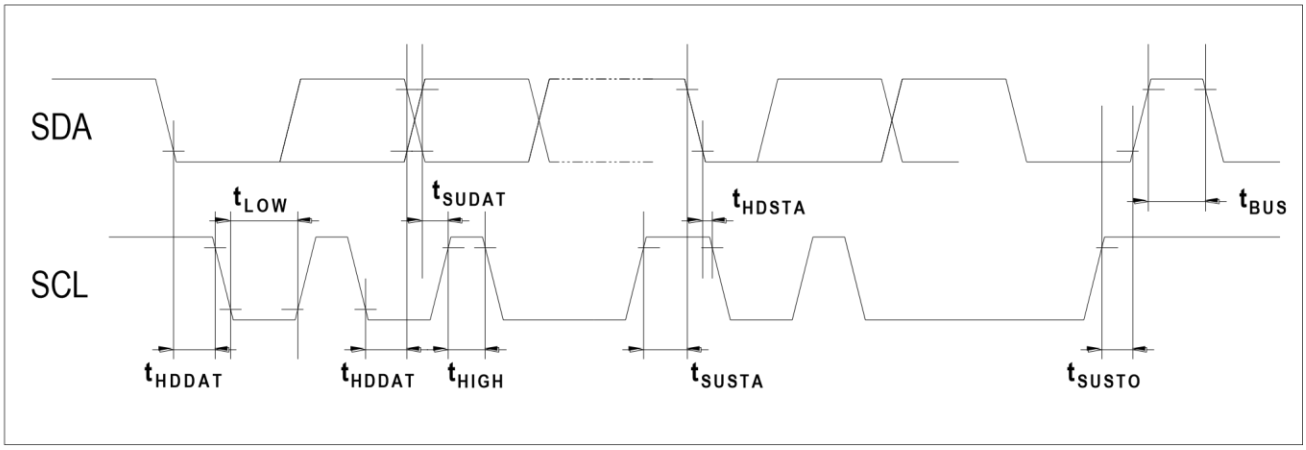
TIMING DIAGRAMS

I^2C INTERFACE PARAMETERS

PARAMETERS	SYMBOL	MIN	TYP	MAX	UNITS
SCLK CLOCK FREQUENCY	f_{SCL}	100		400	KHz
START CONDITION HOLD TIME RELATIVE TO SCL EDGE	t_{HDSTA}	0.1			uS
MINIMUM SCL CLOCK LOW WIDTH [1]	t_{LOW}	0.6			uS
MINIMUM SCL CLOCK HIGH WIDTH [1]	t_{HIGH}	0.6			uS
START CONDITION SETUP TIME RELATIVE TO SCL EDGE	t_{SUSTA}	0.1			uS
DATA HOLD TIME ON SDA RELATIVE TO SCL EDGE	t_{HDDAT}	0			uS
DATA SETUP TIME ON SDA RELATIVE TO SCL EDGE	t_{SUDAT}	0.1			uS
STOP CONDITION SETUP TIME ON SCL	t_{SUSTO}	0.1			uS
BUS FREE TIME BETWEEN STOP AND START CONDITION	t_{BUS}	2			uS

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

I2C Timing Diagram

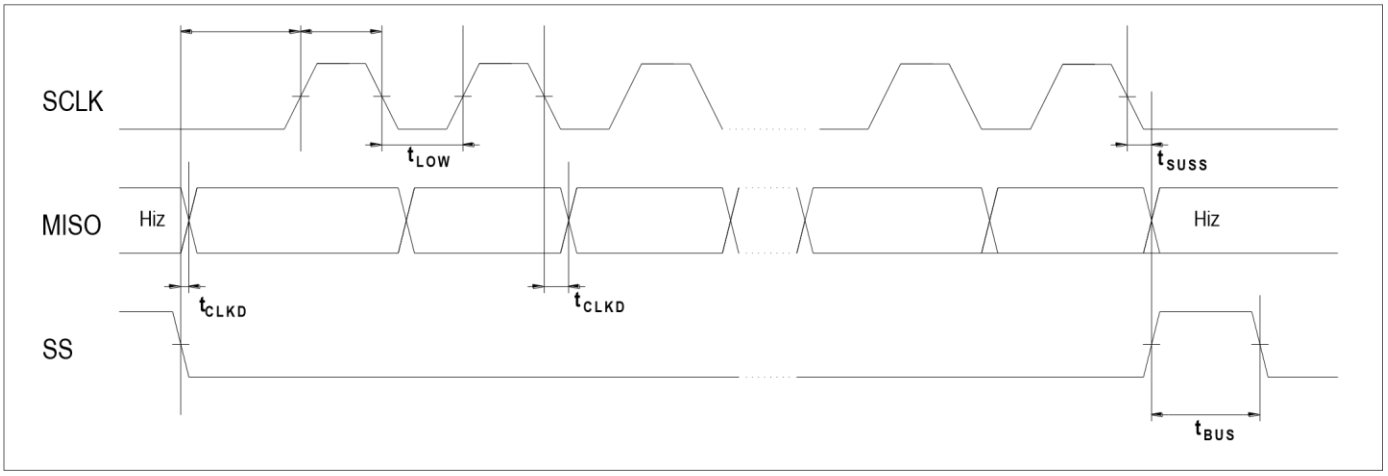


SPI INTERFACE PARAMETERS

PARAMETERS	SYMBOL	MIN	TYP	MAX	UNITS
SCLK CLOCK FREQUENCY	f_{SCL}	50		800	KHz
SS DROP TO FIRST CLOCK EDGE	t_{HDSS}	2.5			uS
MINIMUM SCL CLOCK LOW WIDTH [1]	t_{LOW}	0.6			uS
MINIMUM SCL CLOCK HIGH WIDTH [1]	t_{HIGH}	0.6			uS
CLOCK EDGE TO DATA TRANSITION	t_{CLKD}	0		0.1	uS
RISE OF SS RELATIVE TO LAST CLOCK EDGE	t_{SUSS}	0.1			uS
BUS FREE TIME BETWEEN RISE AND FALL OF SS	t_{BUS}	?			uS

[1] COMBINED LOW AND HIGH WIDTHS MUST EQUAL OR EXCEED MINIMUM SCLK PERIOD.

SPI TIMING DIAGRAM



C Code Example For FX29

```
//Note: The C code is use for communication with FX29K0-040B-0100-L using STM32L031.
//        This routine is applicable to other models mentioned in this document.
#include "main.h"
#include "stm32l0xx_hal.h"
#include "stdlib.h"
#include "delay.h"
#include "config.h"

u8 temp[7];
float Tscope,Pscope,Tdisplay,Pdisplay;
float Lmax=100,Lmin=0 ; //Span 100L,  Zero 0L, Span should be defined by the sensor pressure range of customer used. 100 means pressure range of 100L
u32 Pvalue,Tvalue,Tspan,Pspan;
u16 P1=1000,P2=15000;

void SDA_IN2(void);
void SDA_OUT2(void);
void IIC_Start2(void);
void IIC_Stop2(void);
unsigned char IIC_Wait_Ack2(void);
void IIC_Ack2(void);
void IIC_NAck2(void);
void IIC_Send_Byte(unsigned char txd);
unsigned char IIC_Read_Byte(unsigned char ack);
float Get_I2CValue(void);

void SDA_IN2()
{
        GPIO_InitTypeDef  GPIO_InitStructure;
        GPIO_InitStructure.Pin = SDA2_Pin;
  GPIO_InitStructure.Mode = GPIO_MODE_INPUT;
  GPIO_InitStructure.Pull = GPIO_NOPULL;
  //GPIO_InitStructure.Alternate = GPIO_PuPd_UP;
  GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(SDA2_GPIO_Port, &GPIO_InitStructure);
}
void SDA_OUT2()
```

```
{
        GPIO_InitTypeDef  GPIO_InitStructure;
        GPIO_InitStructure.Pin = SDA2_Pin;
  GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStructure.Pull = GPIO_NOPULL;
        GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(SDA2_GPIO_Port, &GPIO_InitStructure);
}
void IIC_Start2()
{
        SDA_OUT2();     //sda???
        Sensor_SDA_ON ;
        Sensor_SCL_ON;
        delay_us(4);
        Sensor_SDA_OFF;//START:when CLK is high,DATA change form high to low
        delay_us(4);
        Sensor_SCL_OFF;//??I2C??,?????????
}
void IIC_Stop2()
{
        SDA_OUT2();//sda???
        Sensor_SCL_OFF;
        Sensor_SDA_OFF;//STOP:when CLK is high DATA change form low to high
        delay_us(4);
        Sensor_SCL_ON;
        Sensor_SDA_ON ;//??I2C??????
        delay_us(4);
}
unsigned char IIC_Wait_Ack2()
{
        unsigned char ucErrTime=0;
        SDA_IN2();      //SDA?????
        Sensor_SDA_ON ;delay_us(1);
        Sensor_SCL_ON;delay_us(1);
        while(READ_Sensor_SDA)
        {
                ucErrTime++;
                if(ucErrTime>250)
                {
                        IIC_Stop2();
```

```
                    return 1;
                }
            }
            Sensor_SCL_OFF;//????0
            return 0;
}
void IIC_Ack2()
{
            Sensor_SCL_OFF;
            SDA_OUT2();
            Sensor_SDA_OFF;
            delay_us(2);
            Sensor_SCL_ON;
            delay_us(2);
            Sensor_SCL_OFF;
}
void IIC_NAck2()
{
            Sensor_SCL_OFF;
            SDA_OUT2();
            Sensor_SDA_ON;
            delay_us(2);
            Sensor_SCL_ON;
            delay_us(2);
            Sensor_SCL_OFF;
}
void IIC_Send_Byte(unsigned char txd)
{
    unsigned char t;
                SDA_OUT2();
    Sensor_SCL_OFF;//??????????
    for(t=0;t<8;t++)
    {
                                if(txd&0x80)
                                {Sensor_SDA_ON;}
                                else
                                {Sensor_SDA_OFF;}
        txd<<=1;
                delay_us(2);   //?TEA5767??????????
                Sensor_SCL_ON;
```

```
                delay_us(2);
                Sensor_SCL_OFF;
                delay_us(2);
    }
}
unsigned char IIC_Read_Byte(unsigned char ack)
{
        unsigned char i,receive=0;
        SDA_IN2();//SDA?????
  for(i=0;i<8;i++ )
        {
                Sensor_SCL_OFF;
                delay_us(2);
                Sensor_SCL_ON;
                receive<<=1;
    if(READ_Sensor_SDA)receive++;
                delay_us(1);
    }
    if (!ack)
       IIC_NAck2();//??nACK
    else
       IIC_Ack2(); //??ACK
    return receive;
}
u8 I2C_ERR=0;
float Get_I2CValue()
{
                //Wake_up,   if non-sleep mode this part is no needed.
                IIC_Start2();           //MR command
    IIC_Send_Byte(0x51);
                IIC_Wait_Ack2();
                IIC_Stop2();
                HAL_Delay(2);                                                   //2ms
delay
                ////////////////////////////////////////////////////////////////////////////////

                IIC_Start2();           //DF4
    IIC_Send_Byte(0x51);
    IIC_Wait_Ack2();
                temp[0]=IIC_Read_Byte(1);
```

```
                temp[1]=IIC_Read_Byte(1);
                temp[2]=IIC_Read_Byte(1);
                temp[3]=IIC_Read_Byte(0);
    IIC_Stop2();
                if((temp[0]&0xc0)==0x00)
                {
                        Pvalue=(temp[0]<<8)  |        temp[1];
                        Tvalue=(temp[2]<<3)  |        (temp[3]>>5);
                        I2C_ERR=0;
                }
                else
                        I2C_ERR=1;

                Tscope=200;//-50~150
                Tspan=2048;//11bit

        if(I2C_ERR==0)
        {
                Pspan=P2-P1;
                Tdisplay=Tvalue*Tscope/Tspan-50;
                Pdisplay=Pvalue*(Lmax-Lmin)/Pspan+Lmin;//100L
        }
        return Pdisplay;
}
```