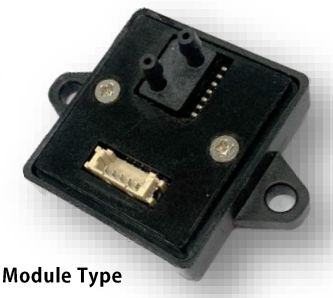
SM series

Differential Pressure Sensor-Package

SOIC16 Package Type



Module Type

- I²C / Analog Interface
- Pressure range : 0 ~250Pa / ±125Pa
- Total Error band : +/-1.0% F.S.
- 24 Bit / 16 Bit Σ-Δ ADC
- Operating voltage 3.3V / 5.0V
- Operating Temperature : -20 ~ 70℃
- SOIC-16 package

The SM Series is the pressure sensor which measures differential gauge pressures. The pressure sensor element and the ASIC are mounted inside a system-in-package and wire-bonded to appropriate contacts. The SM Series provides the digital output data with the format of I²C interface. It can achieve ESD robustness, fast response time, high accuracy and linearity as well as long-term stability. All measurement data is fully calibrated and temperature compensated.
In addition, it allows for easy system integration.

❒ Electrical Characteristics

Parameter	Symbol	Min.	Typ.	Max.	Unit	Comments
Operating Current	I_{avdd}	-	2.4	-	mA	Operating Mode @25℃
		-	-	1	uA	Standby Mode @25℃
ADC Resolution	RES_{RAW}	-	24	-	bits	
DAC Resolution		-	12		bits	
Output load resistance	R_{load}	1	-	-	KOhm	Analog output
Output load capacitance	C_{load}	-	-	15	nF	Analog output
Full Accuracy	ACC	-1	-	+1	%FS	
Power up time	T_{UP}	-	100	-	ms	0~50℃

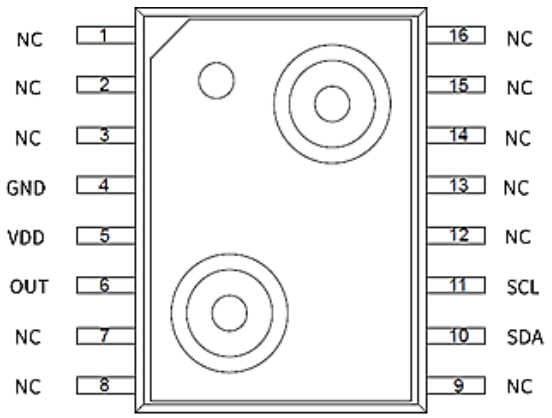
❒ Operating Range

Parameter	Symbol	Min.	Typ.	Max.	Unit	Comments
Supply Voltage Proof Pressure	VDD	3	3.3	3.6	V	VDD=3.3V
		4.5	5	5.5	V	VDD=5.0V
Operating Pressure Range	P_{range}	-125	-	250	Pa	P_{max} - P_{min}
High level voltage at digit I/O	VIH	2	-	-	V	
Low level voltage at digit I/O	VIL	-	-	0.8	V	
I²C clock frequency	F_{sclk}	-	-	400	kHz	
Operating temperature	T_{opr}	-20	-	70	℃	

❒ Absolute Max Ratings

Parameter	Symbol	Min.	Typ.	Max.	Unit	Comments
Supply Voltage	VDD_{max}	-0.3	-	6.5	V	
Analog output current limit		-	-	25	mA	
Digital pin voltage		VDD -0.3	-	VDD+0.3	V	@25 ℃
Proof pressure	P_{proof}	30	-	-	kPa	Pressure range<10Kpa
Burst pressure	P_{burst}	50	-	-	kPa	Pressure range<10Kpa
ESD	HBM	-	2	-	kV	
Storage temperature	Tstg	-40	-	100	℃	

❒ PIN Description

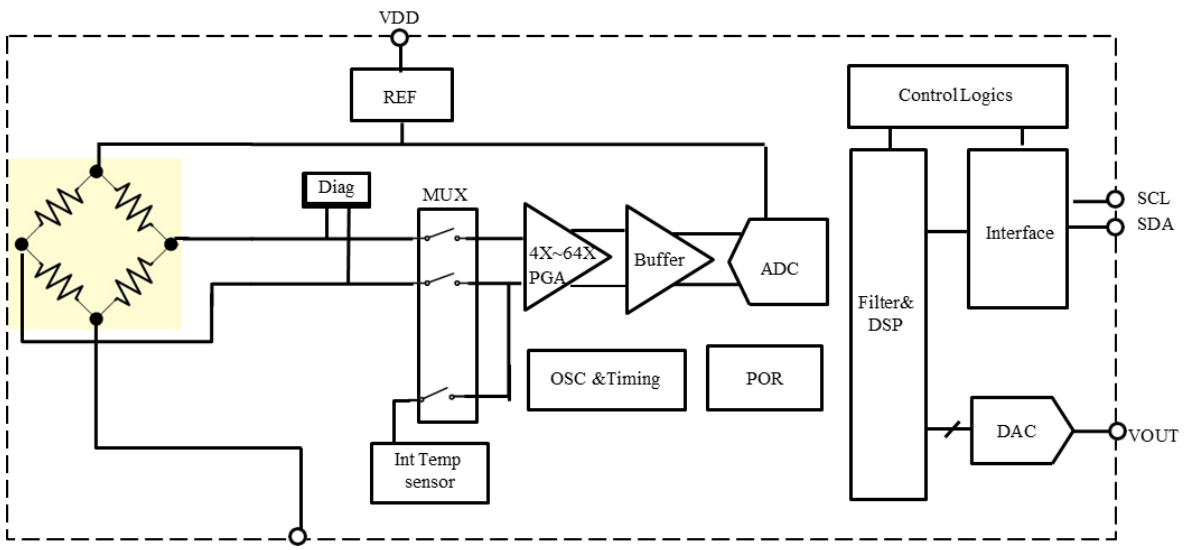
SOIC16 Package Description	P/N No.	Name	Function description
	1-3	NC	No Connected
	4	GND	Ground
	5	VDD	Power Supply
	6	OUT	Analog output
	7-9	NC	No Connected
	10	SDA	I²C data Signal
	11	SCL	I²C clock Signal
	12-16	NC	No Connected

❒ I²C Interface

Parameter	Symbol	Min.	Typ.	Max.	Unit	Comments
Clock frequency	f_{scl}	-	-	400	kHz	
SCL low pulse	t_{LOW}	1.3	-	-	us	
SCL high pulse	t_{HIGH}	0.6	-	-	us	
SCL setup time	t_{SUDAT}	0.1	-	-	us	
SDA hold time	t_{HDDAT}	0.0	-	-	us	
Setup time for a repeated Start condition	t_{SUSTA}	0.6	-	-	us	
Hold time for a start condition	t_{HDSTA}	0.6	-	-	us	
Setup time for a stop condition	t_{SUSTO}	0.6	-	-	us	
Time before a new transmission can start	t_{BUF}	1.3	-	-	us	

❒ Function Description

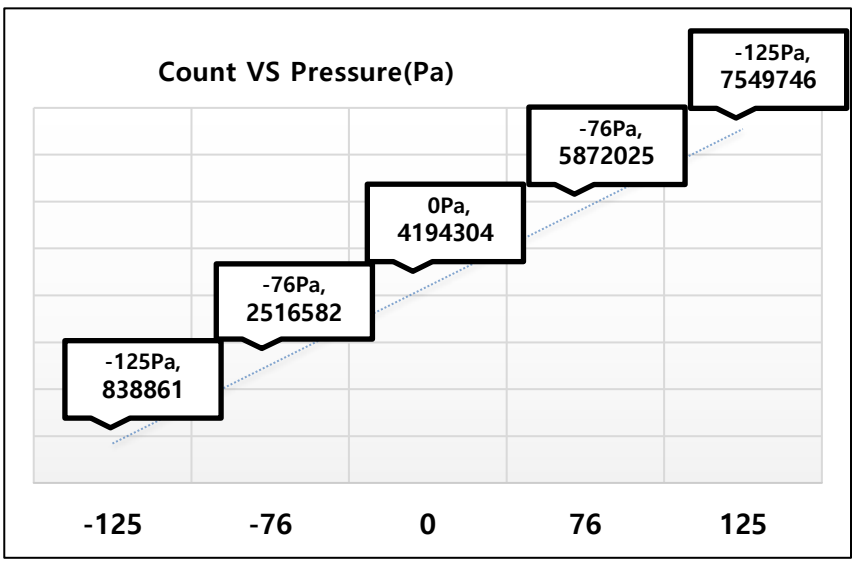
Diagram



❒ Digital output Transfer Function

P = $A * code$/8388608 + B
Code is the register 0x06~0x08 value;
P is the pressure value, differential pressure ,unit is Pa/kPa;

Product NO.	Pressure range		Output code range		Gain and offset	
	P_L	P_H	O_L	O_H	A	B
EX) SM250CA	-125Pa	+125Pa	838861	7549746	1.250	-0.625



Ex)
Pressure range Low : -125Pa
Pressure range High : +125Pa

Output range
⇒ Low : 838861
⇒ High : 7549746

Gain and offset
⇒ A : 1.250
⇒ B : -0.625

❒ Resister Map

Addr	Bit Addr	Description	Default	Description
0x30	7 – 4	Reserve	4'b0000	Write with 0x0A to start a conversion, automatically come back to 0x02 after conversion ends.
	3	Sco	1'b0	
	2 – 0	Measurement_ ctrl<2:0>	3'b000	
0x06	7 – 0	PDATA<23:16>	0x00	Output Pressure Data. Code = Data0x06*2^16+ Data0x07*2^8+ Data0x08 ;
0x07	7 – 0	PDATA<15:8>	0x00	
0x08	7 – 0	PDATA<7:0>	0x00	

For example:
If the value of the registers 0x06、0x07、0x08 are 0x3F, 0xFF, 0xFF, according to SM series transfer function, Code = 4194303, P(Pa) = 4194303/8388607*A+B, and finally get the value of pressure about 0kPa.

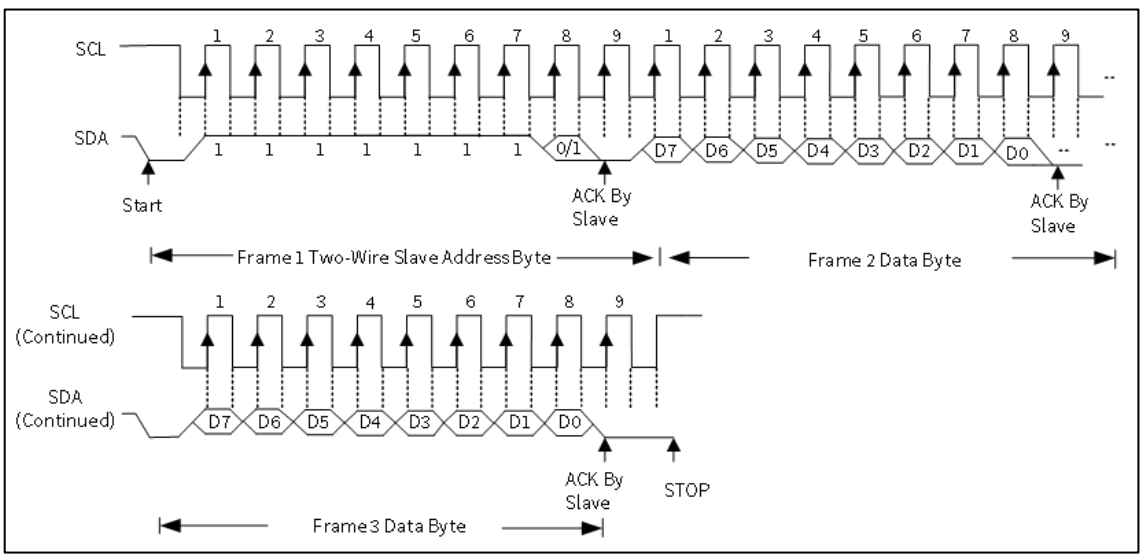
❒ I²C Interface

I²C bus uses SCL and SDA as signal lines. Both lines are connected to VDD externally via pull-up resistors so that they are pulled high when the bus is free. The I²C device address of SM series is shown below.

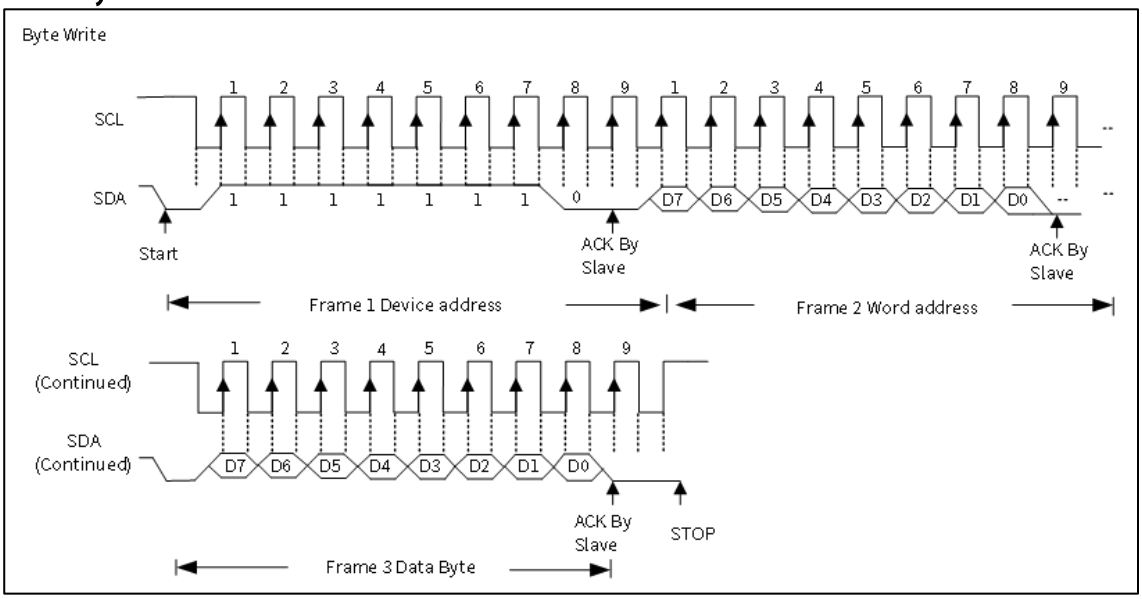
A7	*A6*	*A5*	*A4*	*A3*	*A2*	*A1*	*W/R*
1	1	1	1	1	1	1	0/1

The I²C interface protocol has special bus signal conditions. Start (S), stop (P) and binary data conditions are shown below. At start condition, SCL is high and SDA has a falling edge. Then the slave address is sent. After the 7 address bits, the direction control bit R/W selects the read or write operation. When a slave device recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle. At stop condition, SCL is also high, but SDA has a rising edge. Data must be held stable at SDA when SCL is high. Data can change value at SDA only when SCL is low.

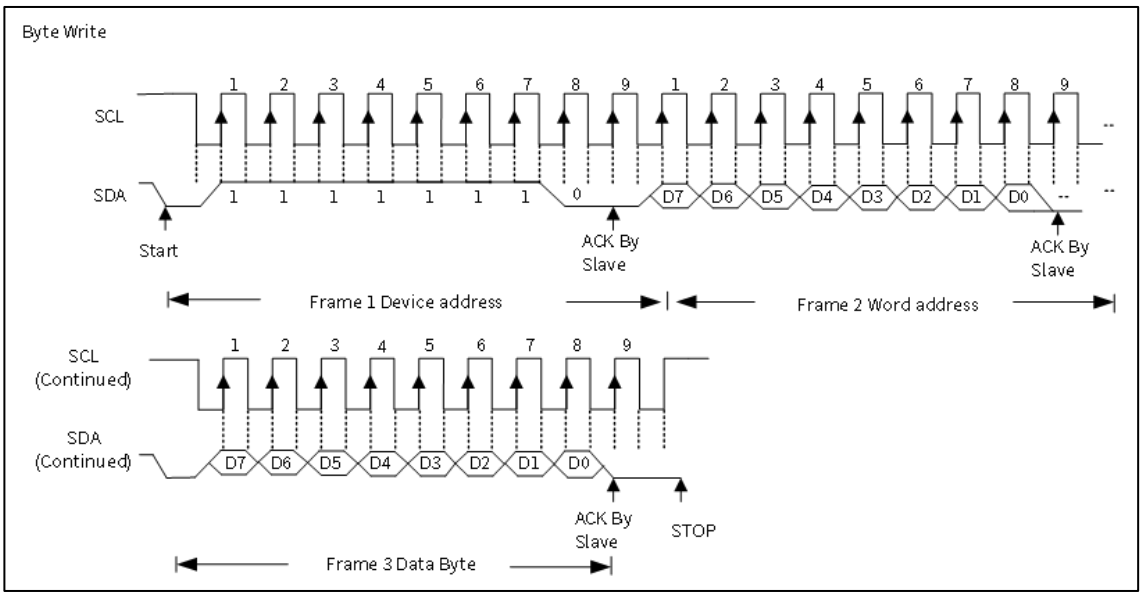
❒ I²C Protocol



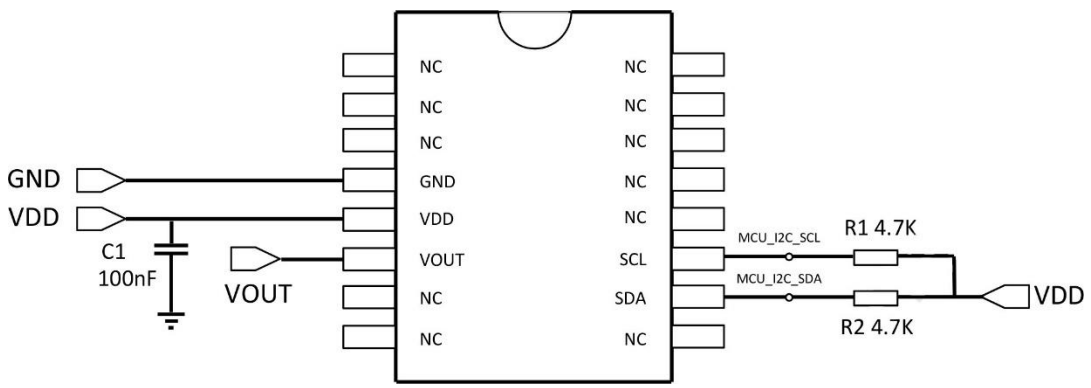
❒ I²C Write Byte



❒ I²C Read Byte

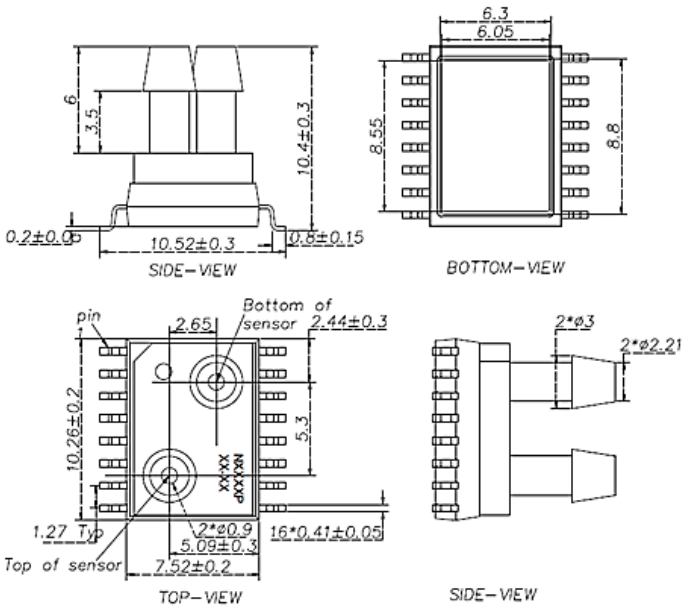


❒ Typical Application



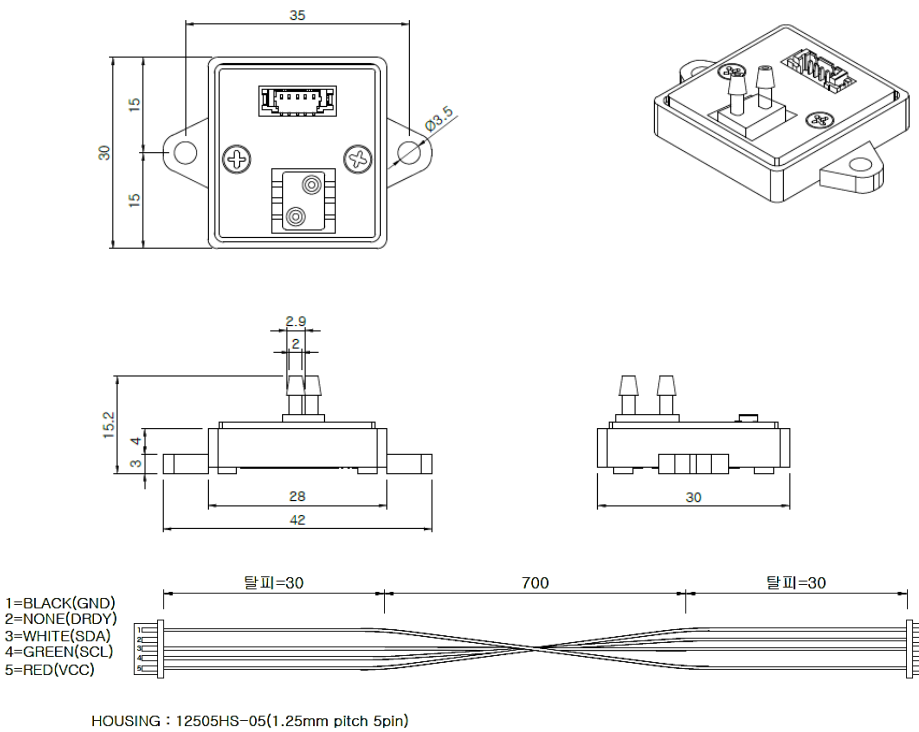
❒ Dimension

SOIC16 Package [unit/mm]



Top of sensor is tube connected to top side of sensor die. Topside pressure is positive pressure.
An increase in topside pressure will result in a increase in sensor output.
Bottom of sensor is tube connected to bottom side of sensor die.

Module Type [unit/mm]



HOUSING : 12505HS-05(1.25mm pitch 5pin)
수축튜브 : 각 20mm

❒ How to order

How to order

	SM	250	C	A	M
1) Pressure Range		I	I	I	I
0 to 250 Pa		250	I	I	I
-125 to 125 Pa		125	I	I	I
Option		XXX	I	I	I
2) Output		----------	I	I	I
I²C Interface		----------	C	I	I
Analog Interface		----------	A	I	I
3) Power		----------	----------	I	I
5VDC		----------	----------	A	I
3.3VDC		----------	----------	B	I
4) With Module		----------	----------	----------	I
Module Type		----------	----------	----------	M

❒ Document Revision History

Revision	DATE	Description
0.0	2022.12.11	First Release.
0.1	2023.05.15	Spec changed to page 1 .
0.2	2023.05.24	Add to Module Type

DISCLAIMER

We hereby expressly disclaims any liability of us to any customer, licensee or any other third party, and any such customer, licensee and any other third party hereby waives any liability of us for any damages in connection with or arising out of the furnishing, performance or use of this technical data, whether based on contract, warranty, tort (including negligence), strict liability, or otherwise.

This datasheet is applies to a product under development. Its characteristics and specifications are subject to change without notice and we assumes no obligation regarding future manufacture unless otherwise agreed to in writing.

However, under no circumstances shall us be liable to any customer, licensee, or any other third party for any special, indirect, incidental, or consequential damages of any kind or nature whatsoever arising out of or in any way related to the furnishing, performance, or use of this technical data. The datasheet is furnished hereby is believed to be true and accurate.

❒ I²C Reading Code Example

```
//* MAIN PRORGAM *//

#define ACK     1
#define NACK    0
uchar REG06=0,REG07=0,REG08=0;
uchar number=1;
uchar Reg30[1];
int PCode=0,Pdata=0;
float Pressure=0.0;
void IIC_Start(void)                //Start the I²C, SDA High-to-low when SCL is high
{
    IIC_SCL(1);                     //SCL output high level
    SDA_OUT(1);                     //SDA output high level
    Delay_us(2);                    //Delay 2us
    SDA_OUT(0);                     //SDA output low level
    Delay_us(2);
}

void IIC_Stop(void)                 //Stop the I²C, SDA Low-to-high when SCL is high
{
    IIC_SCL(0);
    Delay_us(2);
    IIC_SCL(1);
    SDA_OUT(0);
    Delay_us(2);
    SDA_OUT(1);
    Delay_us(2);
}

void IIC_ACK(void)                  //Send ACK (LOW)
{
    SDA_OUT(0);
    IIC_SCL(1);
    Delay_us(2);
    IIC_SCL(0);
}

void IIC_NACK(void)                 //Send No ACK (High)
{
    SDA_OUT(1);
    IIC_SCL(1);
    Delay_us(2);
    IIC_SCL(0);
}

uchar IIC_Wait_ACK(void)
{
    int ErrTime=0;
```

❒ I²C Reading Code Example

//* MAIN PRORGAM *//

```
    SDA_IN();                       //SDA set as input
    IIC_SCL(1);
    Delay_us(2);
    while(Read_SDA)
    {
        ErrTime++;
        if(ErrTime>200)
        {
            IIC_Stop();
            return 1;
        }
    }
    IIC_SCL(0);
    SDA_OUT(0);
    Delay_us(2);
    return 0;
void IIC_Send(uchar IIC_Data)       //Send a byte to I²C
{
    uchar i;
    IIC_SCL(0);
    Delay_us(2);
    for(i=0;i<8;i++)
    {
        if((IIC_Data&0x80)>>7)
            SDA_OUT(1);
        else
            SDA_OUT(0);
        IIC_Data<<=1;
        IIC_SCL(1);
        Delay_us(2);
        IIC_SCL(0);
        Delay_us(2);
    }
}
uchar IIC_Receive(uchar ACK)        //Receive a byte from I²C
{
    uchar i,Receive_Data=0;
    SDA_IN();
    for(i=0;i<8;i++)
    {
        IIC_SCL(0);
        Delay_us(2);
        IIC_SCL(1);
        Receive_Data<<=1;
        if(Read_SDA==1)
            Receive_Data++;
```

❒ I²C Reading Code Example

```
//* MAIN PRORGAM *//
                Delay_us(2);
        }
        IIC_SCL(0);
        Delay_us(2);
        if(ACK==0x01)
                IIC_ACK();
        else
                IIC_NACK();
        return Receive_Data;
}
void NSPDS5F001DT02_Write_Byte(uchar WriteAddr,uchar WriteData)
{
        IIC_Start();
        IIC_Send(0xFE|0x00);
        IIC_Wait_ACK();
        IIC_Send(WriteAddr);
        IIC_Wait_ACK();
        IIC_Send(WriteData);
        IIC_Wait_ACK();
        IIC_Stop();
}
void NSPDS5F001DT02_Read_Byte(uchar ReadAddr, uchar *pBuffer)
{
        IIC_Start();
        IIC_Send(0xFE|0x00);
        IIC_Wait_ACK();
        IIC_Send(ReadAddr);
        IIC_Wait_ACK();
        IIC_Start();
        IIC_Send(0xFE|0x01);
        IIC_Wait_ACK();
        pBuffer[0]=IIC_Receive(0);
        IIC_Stop();
}
void NSPDS5F001DT02_Read_3Byte(uchar ReadAddr,uchar *pBuffer)
{
        IIC_Start();
        IIC_Send(0xFE|0x00);
        IIC_Wait_ACK();
        IIC_Send(ReadAddr);
        IIC_Wait_ACK();
        IIC_Start();
        IIC_Send(0xFE|0x01);
        IIC_Wait_ACK();
        pBuffer[0]=IIC_Receive(ACK);
        pBuffer[1]=IIC_Receive(ACK);
```

❒ I²C Reading Code Example

//* MAIN PRORGAM *//

```
    pBuffer[2]=IIC_Receive(NACK);
    IIC_Stop();
}

void main()
{
    uchar PData[3]={0,0,0};
    while(1)
    {
        NSPDS5F001DT02_Write_Byte(0x30,0x0A);
        while(1)                                        //Check whether the conversion ends
        {
            if(number<=50)
            {
                number++;
                delay_ms(1);
                NSPDS5F001DT02_Read_Byte(0x30,Reg30);
                if(0x02==Reg30[0])
                {
                    number=1;
                    break;
                }
            }
            if(number>50)
            {
                number=1;
                //User can add his own error handler function
                break;
            }
        }
        NSPDS5F001DT02_Read_3Byte(0x06,PData);
        REG06 = PData [0];                              //Register 0x06
        REG07 = PData [1];                              //Register 0x07
        REG08 = PData [2];                              //Register 0x08
        PCode=(REG06*65536+REG07*256+REG08);            //PCode = Data0x06*2^16+ Data0x07*2^8+ Data0x08
        if (PCode >8388607)
            Pdata= PCode-16777216;                      //Symbol processing
        else
            Pdata= PCode;
        Pressure = float (1.25*Pdata/8388607-0.625) ;   // PCode=(A*P+B*) 8388607   P=A*PCode/8388607+B
                                                        // A=1.25, B= -0.625
                                                        //PNormalized=PCode/8388607
    }
}
```