Application Note

NOVOSENSE

I²C Data Reading and Custom Address of NSPGS2/NSPGD1/NSPDSx

AN-12-0008

Author: Kun He

AN-12-0008

NOVOSENSE

I²C Data Reading and Custom Address of NSPGS2/NSPGD1/NSPDSx

ABSTRACT

This application note introduces the I²C interface of NSPGS2/NSPGD1/NSPDSx series products, and provides detailed configuration methods for reading pressure, temperature data and communication addresses, to facilitate customer applications.

INDEX

I²C Data Reading and Custom Address of NSPGS2/NSPGD1/NSPDSx

1.Introduction

Inter-Integrated Circuit (IIC) communication, also known as I²C, is a serial communication protocol that is widely used for data transmission between various electronic devices. I²C uses two lines for data communication between the main controller and the slave. One is SCL (serial clock line), and the other is SDA (serial data line). These two data lines need to be connected to a pull-up resistor respectively. When the bus is idle, SCL and SDA are at high levels. In the I²C communication protocol, devices are divided into master devices (Master) and slave devices (Slave). The master device is responsible for controlling the entire communication process, while the slave device accepts the control of the master device and provides data as needed. One or more slave devices can be connected to the I²C communication bus. However, the default factory address for NSPGS2/NSPGD1/NSPDSx series sensors is the same, which poses certain difficulties for I²C bus to mount multiple sensors for data reading. In this application note, not only are methods for reading pressure and temperature data provided, but methods for configuring devices with different addresses are also provided.

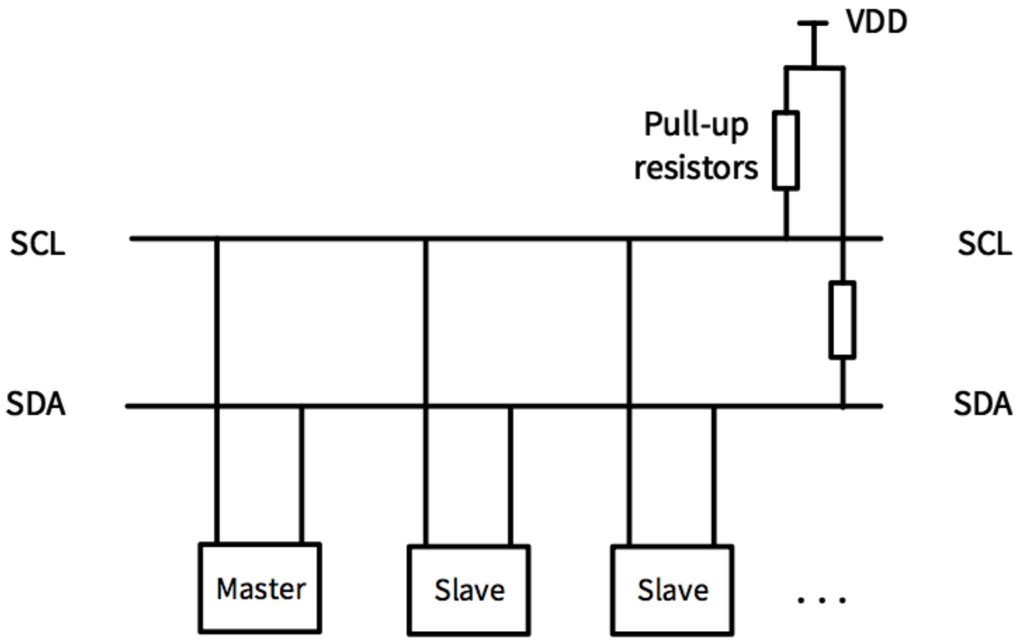


Figure 1.1 I²C Bus Diagram

1.1.I²C timing diagram

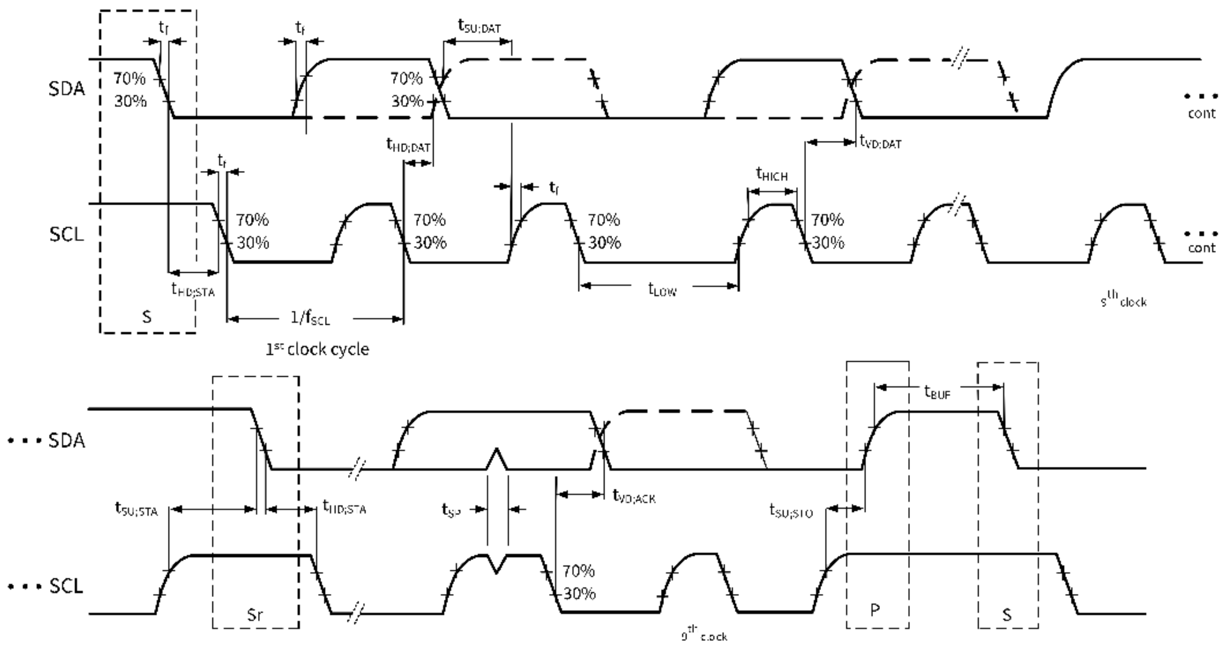


Figure 1.2 I²C Timing Diagram

1.2.I²C electrical characteristics

Parameters	Symbol	Min	Typ	Max	Unit	Comments
Clock frequency	fB_{sclB}			400	kHz	
SCL low pulse	tB_{LOWB}	1.3			us	
SCL high pulse	tB_{HIGHB}	0.6			us	
SDA setup time	tB_{SUDATB}	0.1			us	
SDA hold time	tB_{HDDATB}	0.0			us	
Setup time for a repeated start condition	tB_{SUSTAB}	0.6			us	
Hold time for a start condition	tB_{HDSTAB}	0.6			us	
Setup time for a stop condition	tB_{SUSTOB}	0.6			us	
Time before a new transmission can start	tB_{BUFB}	1.3			us	

Copyright © NOVOSENSE Microelectronics Co.,Ltd 2023. All rights reserved.

2.Data read and processing

2.1.Register 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. Signed, 2's complement. P_Code = Data0x06*2^{16}+ Data0x07*2^{8}+ Data0x08 ;
0x07	7 – 0	PDATA<15:8>	0x00	
0x08	7 – 0	PDATA<7:0>	0x00	
0x09	7 – 0	TDATA<15:8>	0x00	Output Temperature Data. Signed, 2's complement. T_Code = Data0x09*2^8+ Data0x0A ;
0x0A	7 – 0	TDATA<7:0>	0x00	
0xA3	7	Ex_addr_en	1'b0	Ex_addr_en = 0, I²C address is decided by chip_address<6:0>. Configurable address range 1~126
	6-0	Chip_address<6:0>	7'b0000000	

Copyright © NOVOSENSE Microelectronics Co.,Ltd 2023. All rights reserved.

2.2.Transfer function

The transfer functions of NSPGS2, NSPGD1, and NSPDSx series are shown in the table below:

Table 2.1 Transfer Function

Transfer Function Type	**NSPGS2, NSPGD1**	**NSPDSx**
Pressure	P = (P_Code/8388607-B)/A	P = A * P_Code/8388607+B
Temperature	T = T_Code/256+7	

P_Code is the value of register 0x06~0x08;

T_Code is the value of register 0x09~0x0A;

P is the pressure value, gauge/differential pressure, unit is kPa/Pa/mmH2O;

T is the temperature value, unit is °C.

Taking the part No. NSPGS2F035DT09 as an example. The transfer function coefficient is as follows:

Table 2.2 I²C Output Transfer Function Coefficient

Product NO.	**Pressure range**		**Output range**		**Gain and offset**	
	P_L	**P_H**	**O_L**	**O_H**	**A**	**B**
NSPGS2F035DT09	0	-35	838861	7549746	-0.02286	0.1

For example:

If the values of the registers 0x06, 0x07, 0x08 are 0x3F, 0xFF and 0xFF, according to NSPGS2F035DT09 transfer function, Code = 4194303, P(kPa) = (4194303/8388607-B)/A, and the final value of pressure is about -17.5kPa.

If the values of the registers 0x09 and 0x0A are 0x17 and 0x00, according to the transfer function, T_Code = 5888, T(°C) = 5888/256+7, and the final value of pressure is about 30°C.

Copyright © NOVOSENSE Microelectronics Co.,Ltd 2023. All rights reserved.

2.3.Data read and processing

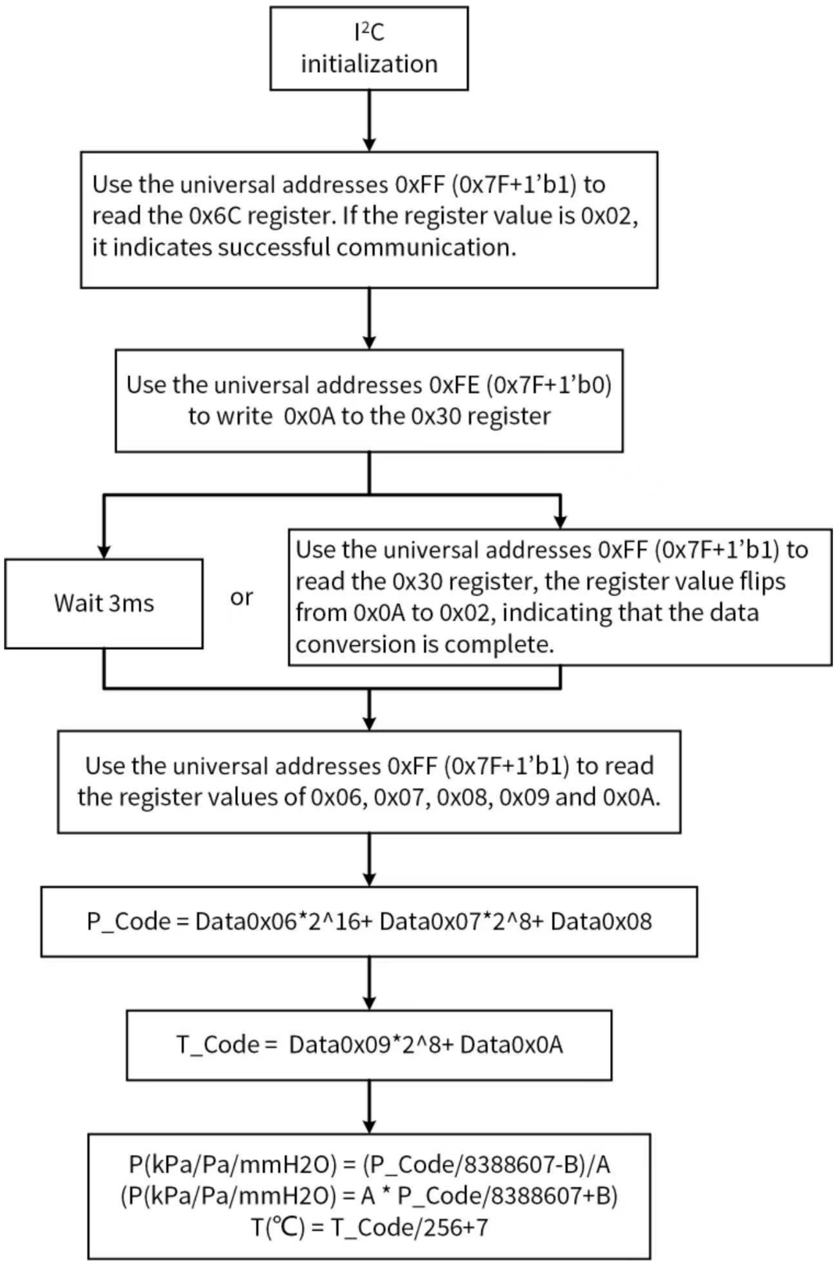

I²C initialization
Use the universal addresses 0xFF (0x7F+1'b1) to read the 0x6C register. If the register value is 0x02, it indicates successful communication.
Use the universal addresses 0xFE (0x7F+1'b0) to write 0x0A to the 0x30 register
Wait 3ms
or
Use the universal addresses 0xFF (0x7F+1'b1) to read the 0x30 register, the register value flips from 0x0A to 0x02, indicating that the data conversion is complete.
Use the universal addresses 0xFF (0x7F+1'b1) to read the register values of 0x06, 0x07, 0x08, 0x09 and 0x0A.
P_Code = Data0x06*2^16+ Data0x07*2^8+ Data0x08
T_Code = Data0x09*2^8+ Data0x0A
P(kPa/Pa/mmH2O) = (P_Code/8388607-B)/A
(P(kPa/Pa/mmH2O) = A * P_Code/8388607+B)
T(℃) = T_Code/256+7

Figure 2.1 Data reading and processing flowchart

Pressure and temperature data are signed data. The highest bit of the data is the sign bit. When the sign bit value is '1', it represents 'negative'; when the sign bit value is '0', it represents 'positive'.

2.3.1.Pressure data processing

After power on, directly read the calibrated register data of 0x06, 0x07, and 0x08 to form a 24-bit pressure AD value. Perform the following operation to calculate the pressure value based on the read AD value:

P_Code =Data0x06*2^16+Data0x07 *2^8+ Data0x08

Pressure value positive and negative processing:

If P_ Code>2^23, it is a negative value, and the pressure P is obtained from the following equation:

P=(P_Code-16777216)/8388607-B)/A --NSPGS2/NSPGD1

P=A*(P_Code -16777216)/8388607+ B --NSPDSx

If P_ Code<2^23, then it is a positive value, and the pressure P is obtained from the following equation:

P=(P_Code/8388607-B)/A --NSPGS2/NSPGD1

P=A*P_Code/8388607+B -- NSPDSx

2.3.2.Temperature data processing

Read the calibrated register data of 0x09 and 0x0A to form a 16-bit temperature AD value. Perform the following operations to calculate the temperature value based on the read AD value:

T_Code=Data0x09*2^8+ Data0x0A

Temperature value positive and negative processing:

If T_ Code>2^16, it is a negative value, and the temperature T is obtained from the following equation:

T=(T_Code-65536)/256+7

If T_ Code<2^16, it is a negative value, and the temperature T is obtained from the following equation:

T=T_Code/256+7

2.4.I²C device universal address

The universal I²C device addresses of NSPGS2 are shown below.

Table 2.3 Universal I²C address

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

The format is: 7bit addr+1bit w/r;

A7~A1 are device addresses: 0x7F; A0 is read/write bit: 0 (write), 1 (read).

2.5.Reading timing of I²C pressure data

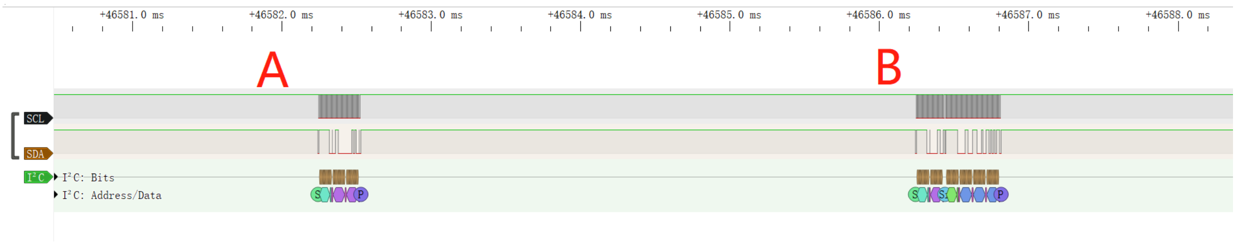


Figure 2.2 Reading Timing of I²C Pressure Data

A： The master writes pressure data reading instruction 0x0A to the sensor 0x30 register.

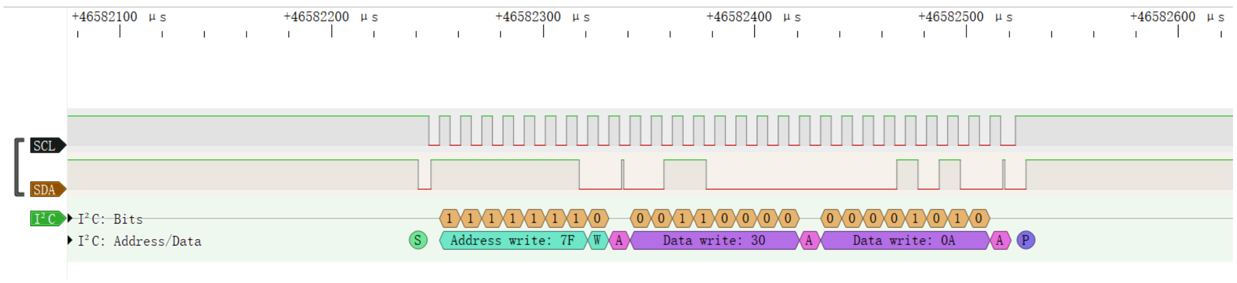


Figure 2.3 Master sending read instructions

Wait 3ms.

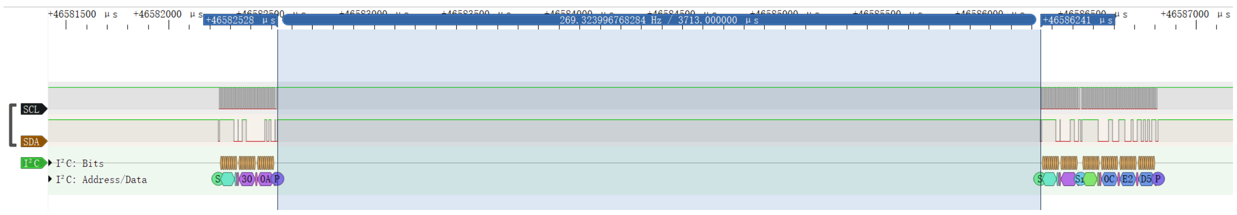


Figure 2.4 Wait 3ms

B： The master continuously reads 3-byte pressure data from the sensor address 0x06.

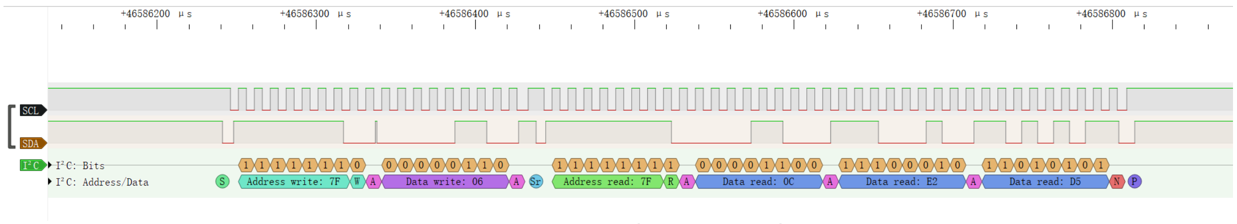
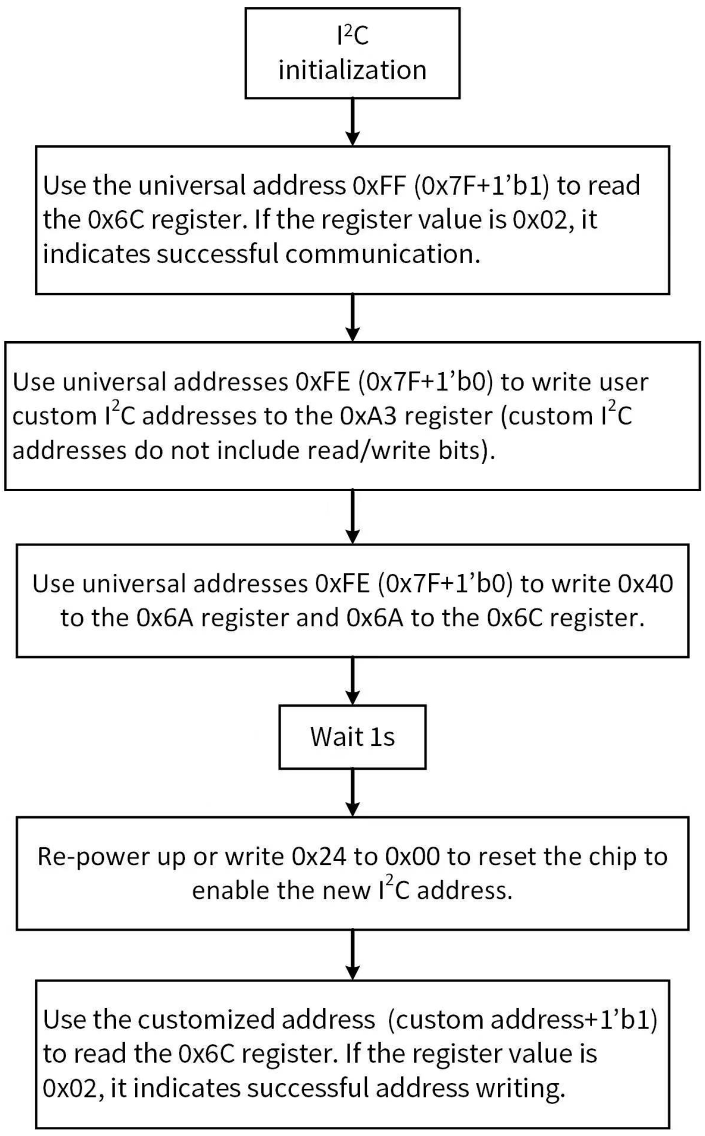


Figure 2.5 Master reading pressure data

3.I²C address customization



I²C initialization

Use the universal address 0xFF (0x7F+1'b1) to read the 0x6C register. If the register value is 0x02, it indicates successful communication.

Use universal addresses 0xFE (0x7F+1'b0) to write user custom I²C addresses to the 0xA3 register (custom I²C addresses do not include read/write bits).

Use universal addresses 0xFE (0x7F+1'b0) to write 0x40 to the 0x6A register and 0x6A to the 0x6C register.

Wait 1s

Re-power up or write 0x24 to 0x00 to reset the chip to enable the new I²C address.

Use the customized address (custom address+1'b1) to read the 0x6C register. If the register value is 0x02, it indicates successful address writing.

Figure 3.1 I²C address customize flowchart

Example of customizing I²C address to 0x45(01000101):

Table 3.1 Custom I²C address

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

Use the universal address 0xFF (0x7F+1’b1) to read the 0x6C register. If the register value is 0x02, it indicates successful communication.

Use universal addresses 0xFE (0x7F+1’b0) to write 0x45 to the 0xA3 register.

Use universal addresses 0xFE (0x7F+1’b0) to write 0x40 to the 0x6A register and 0x6A to the 0x6C register. Wait for 1s for EEPROM programming to complete.

Re-power up or write 0x24 to 0x00 to reset the chip to enable the new I²C address.

Use the customized address 0x8B (0x45+1’b1) to read the 0x6C register. If the register value is 0x02, it indicates successful address writing. Use the customized addresses 0x8A(0x45+1’b0) to write 0x0A to the 0x30 register to start a conversion of pressure and temperature.

Note that the universal address 0xFE/0xFF is always valid. When multiple sensors on the same I2C bus need to communicate, please configure different addresses in advance and use the new address for data reading.

4.Summary

This application note introduces the I²C interface of NSPGS2/NSPGD1/NSPDSx series products. It also provides detailed configuration methods for reading pressure, temperature data and communication addresses, to facilitate customer applications.

Copyright © NOVOSENSE Microelectronics Co.,Ltd 2023. All rights reserved.

5.Notes

1. I²C code

```
#define ACK     1
#define NACK    0
#define IIC_Addr  0x7F      // Universal_Addr:  0x7F;  Custom_Addr:  0x45;
uchar CalData[5]={0};
uchar number=1;
uchar Reg30[1];
uchar REG06=0,REG07=0,REG08=0,REG09=0,REG0A=0;
float A=-0.02286,B=0.1;  //NSPGS2F009DT09 Transfer function coefficient, Different part numbers have different values,
please refer to Datasheet.
int PCode=0, Pdata=0, TCode=0, Tdata=0;
float Pressure=0.0, Temperature=0.0;

void IIC_Start(void)                    //Start the IIC, 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 IIC, 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)           //Check ACK, if return 0, then right, if return 1, then error
{
        int ErrTime=0;
        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 IIC
{
        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++;
                delay_us(2);
        }
        IIC_SCL(0);
        delay_us(2);
        if(Ack==0x01)
                IIC_ACK();
        else
                IIC_NACK();
        return Receive_Data;
}

void NSPXXX_Write_Byte(uchar WriteAddr,uchar WriteData)
{
        IIC_Start();
        IIC_Send(IIC_Addr<<1|0);
        IIC_Wait_ACK();
        IIC_Send(WriteAddr);
        IIC_Wait_ACK();
        IIC_Send(WriteData);
        IIC_Wait_ACK();
        IIC_Stop();
}

void NSPXXX_Read_Byte(uchar ReadAddr, uchar *pBuffer)
{
        IIC_Start();
        IIC_Send(IIC_Addr<<1|0);
        IIC_Wait_ACK();
        IIC_Send(ReadAddr);
        IIC_Wait_ACK();
        IIC_Start();
        IIC_Send(IIC_Addr<<1|1);
        IIC_Wait_ACK();
        pBuffer[0]=IIC_Receive(0);
        IIC_Stop();
}

void NSPXXX_Read_5Byte(uchar ReadAddr,uchar *pBuffer)
{
        IIC_Start();
        IIC_Send(IIC_Addr<<1|0);
        IIC_Wait_ACK();
        IIC_Send(ReadAddr);
        IIC_Wait_ACK();
        IIC_Start();
        IIC_Send(IIC_Addr<<1|1);
        IIC_Wait_ACK();
        pBuffer[0]=IIC_Receive(ACK);
        pBuffer[1]=IIC_Receive(ACK);
        pBuffer[2]=IIC_Receive(ACK);
        pBuffer[3]=IIC_Receive(ACK);
        pBuffer[4]=IIC_Receive(NACK);
        IIC_Stop();
}

void NSPXXX_Address_Custom(uchar Addr)
{
        NSPXXX_Write_Byte(0xA3,Addr);
        NSPXXX_Write_Byte(0x6A,0x40);
        NSPXXX_Write_Byte(0x6C,0x6A);
        delay_ms(1000);
}

void main()
{
        delay_ms(80);
//      NSPXXX_Address_Custom(IIC_Addr);   // I2C Address Customize

        while(1)
        {
                NSPXXX_Write_Byte(0x30,0x0A);
                        while(1)                                //Check whether the conversion ends
                        {
                                if(number<=50)
                                        {
                                                number++;
                                                delay_ms(1);
                                                NSPXXX_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;
                                        }
                        }
//      delay_ms(3);
        NSPXXX_Read_5Byte(0x06,CalData);
        REG06 = CalData[0];                                     //Register 0x06
        REG07 = CalData[1];                                     //Register 0x07
        REG08 = CalData[2];                                     //Register 0x08
        REG09 = CalData[3];                                     //Register 0x09
        REG0A = CalData[4];                                     //Register 0x0A
        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)Pdata/8388607-B)/A;               //NSPGS2、NSPGD1    series    transfer
function, PCode=(AxP+B)*8388607,  P=(PCode/8388607-B)/A
//              Pressure =A*(float)Pdata/8388607+B;             //NSPDSx    series    transfer    function,
PCode=(P-B)*8388607/A,  P=A*Pcode/8388608+B
//PNormalized=PCode/8388607
        TCode=(REG09*256+REG0A);        //TCode = Data0x09*2^8+ Data0x0A
        if (TCode >32768)
                Tdata= TCode-65536;         //Symbol processing
        else
                Tdata= TCode;
                Temperature =(float)Tdata/256+7;      // T=TCode/256+7
        }
}
```

6.Revision history

Revision	Description	Author	Date
1.0	Initial version	Kun He	2023/09/01

Sales Contact: sales@novosns.com; Further Information: www.novosns.com

IMPORTANT NOTICE

The information given in this document (the "Document") shall in no event be regarded as any warranty or authorization of, express or implied, including but not limited to accuracy, completeness, merchantability, fitness for a particular purpose or infringement of any third party's intellectual property rights.

Users of this Document shall be solely responsible for the use of NOVOSENSE's products and applications, and for the safety thereof. Users shall comply with all laws, regulations and requirements related to NOVOSENSE's products and applications, although information or support related to any application may still be provided by NOVOSENSE.

This Document is provided on an "AS IS" basis, and is intended only for skilled developers designing with NOVOSENSE' products. NOVOSENSE reserves the rights to make corrections, modifications, enhancements, improvements or other changes to the products and services provided without notice. NOVOSENSE authorizes users to use this Document exclusively for the development of relevant applications or systems designed to integrate NOVOSENSE's products. No license to any intellectual property rights of NOVOSENSE is granted by implication or otherwise. Using this Document for any other purpose, or any unauthorized reproduction or display of this Document is strictly prohibited. In no event shall NOVOSENSE be liable for any claims, damages, costs, losses or liabilities arising out of or in connection with this Document or the use of this Document.

For further information on applications, products and technologies, please contact NOVOSENSE (www.novosns.com).

Suzhou NOVOSENSE Microelectronics Co., Ltd