NOVOSENSE

# NSPGS2E Series
# Integrated Gauge Pressure Sensor
Datasheet (EN) 1.0

## Product Overview

NSPGS2E series are calibrated gauge pressure sensor which combines state-of-art MEMS sensor technology and CMOS mix-signal processing technology to produce an amplified, fully conditioned, multi-order pressure and temperature compensated sensor in a Small Outline Package (SOP) with tube port. NSPGS2E series pressure sensor is target for Auto and industrial application. Combining the pressure sensor with a signal conditioning ASIC in a single package simplifies the use of advanced silicon micromachined pressure sensors. The pressure sensor can be mounted directly to a standard printed circuit board and an amplified, high-level, calibrated pressure signal can be acquired from the digital interface or analog output. This eliminates the need for additional circuitry, such as a compensation network or micro-controller containing a custom correction algorithm. NSPGS2E series are designed for operating pressure ranges of -100kPa Gauge to 250kPa Gauge, very suitable for automotive applications such as seat pressure measurement and industrial pressure applications.

## Key Features

- High accuracy

  Total error band initially better than ±1.5% (-40°C~115°C)

  Full life accuracy better than ±2.5% (-40°C~115°C)
- Large temperature range -40°C~115°C
- 24bit I$^2$C/SPI
- SOP package with air nozzle, easy to assembly
- RoHS & REACH Compliance
- AEC-Q100 qualified

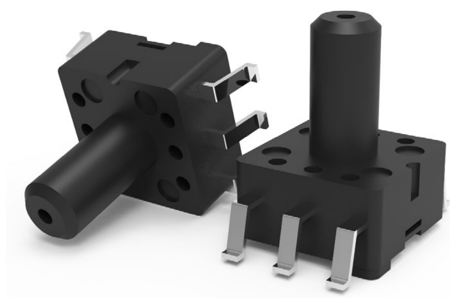
## Applications

- Automotive applications(seat pressure measurement)
- Industrial pressure sensor
- IoT pressure sensor

## Device Information

| Part Number | Package | Body Size |
|---|---|---|
| NSPGS2E | SOP6 | 7mm*7mm |

## Outline



# INDEX

# 1. Pin Configuration and Functions

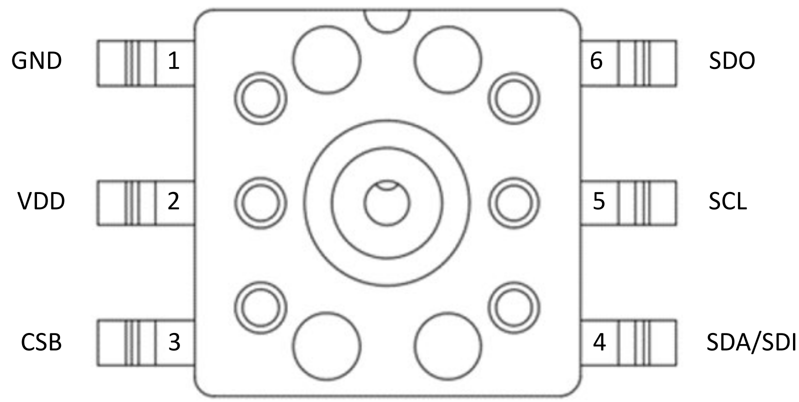



Figure 1.1 NSPGS2E Series Digital Output Pin Definition (Top view)

Table 1.1 Digital Output Pin Description

| Pin NO. | Pin Name | Description |
|---|---|---|
| 1 | GND | Ground |
| 2 | VDD | Power supply |
| 3 | CSB | Chip select |
| 4 | SDA/SDI | Serial data input/output in $I^2C$ mode (SDA) |
|  |  | Serial data input in SPI mode (SDI) |
| 5 | SCL | Serial clock |
| 6 | SDO | Serial data output in SPI mode (SDO) |

## 2. Absolute Maximum Ratings

| Parameters | Symbol | Min | Typ | Max | Unit | Comments |
|---|---|---|---|---|---|---|
| Supply voltage | VDD | -0.3 | | 6.5 | V | |
| Digital pin voltage | | -0.3 | | VDD+0.3 | V | @25°C |
| Proof pressure | $P_{proof}$ | 300 | | | kPa | |
| Burst pressure | $P_{burst}$ | 500 | | 1000 | kPa | |
| Storage temperature | $T_{stg}$ | -40 | | 125 | °C | |

## 3. ESD Ratings

| Ratings | | Value | Unit |
|---|---|---|---|
| Electrostatic discharge | Human body model (HBM), per AEC-Q100-002-RevE | ±2 | kV |
| | Charged device model (CDM), per AEC-Q100-011-RevD | ±500 | V |

## 4. Recommended Operating Conditions

| Parameters | Symbol | Min | Typ | Max | Unit | Comments |
|---|---|---|---|---|---|---|
| Supply voltage | VDD | 3 | 3.3 | 3.6 | V | |
| | | 4.5 | 5 | 5.5 | V | |
| Operating pressure | $P_{amb}$ | -100 | | 250 | kPa | |
| Operating pressure range | $P_{range}$ | 20 | | 350 | kPa | $P_{max}$ - $P_{min}$ |
| Operating temperature | $T_{opr}$ | -40 | | 115 | °C | |

# 5. Specifications

## 5.1. Electrical Characteristic

| Parameters | Symbol | Min | Typ | Max | Unit | Comments |
|---|---|---|---|---|---|---|
| Operating Current | $I_{avdd}$ | | 0.3 | 30 | uA | Standby mode |
| ADC Resolution | $RES_{RAW}$ | | 24 | | Bits | |
| PSRR | PSRR | 90 | 120 | | dB | |
| Accuracy [1,2,3] | ACC | -1.5% | | 1.5% | %FS | Initially accuracy @-40°C~115°C |
| | | -2.5% | | 2.5% | %FS | Full life accuracy @-40°C~115°C |
| Power up Time | $T_{UP}$ | | 100 | | ms | |
| EEPROM Data Retention | $T_{live}$ | 10 | | | years | @125°C |

1. Accuracy includes non-linearity, temperature, pressure hysteresis, temperature hysteresis.
2. Full life accuracy based on the part number NSPGS2E170DT42 1000 hour HTOL, LTOL, THB and PCT testing.
3. For pressure accuracy of different part number, please refer to complete part number list at chapter 9.

## 5.2. I²C Timing Diagram

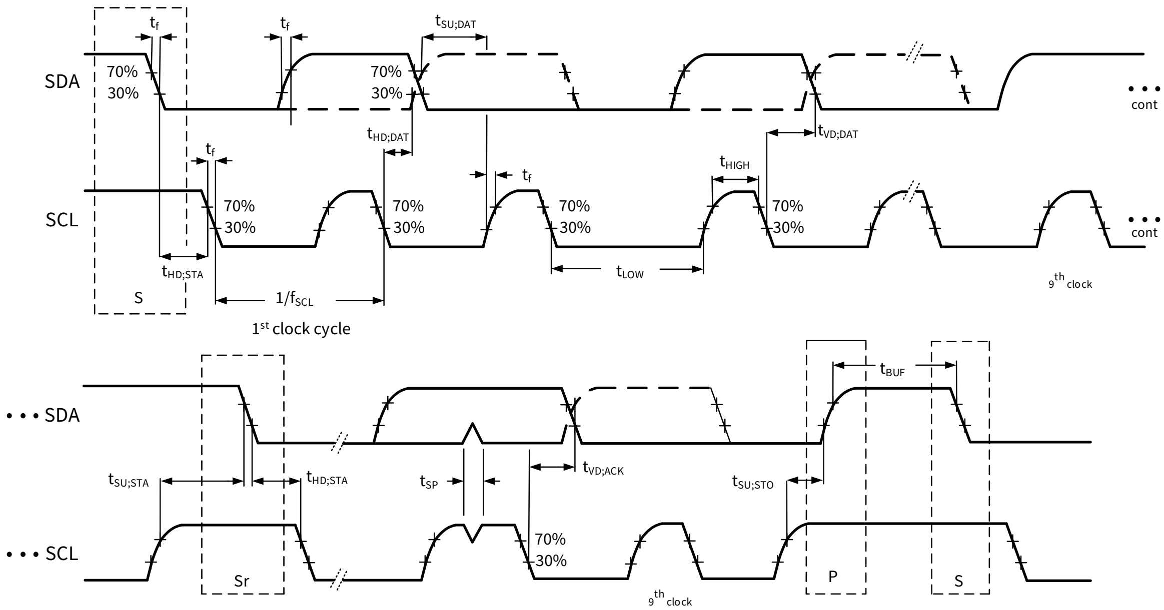



Figure 5.1 I²C Timing Diagram

## 5.3. I²C Electrical Characteristics

| *Parameters* | *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 | |
| SDA 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 | |

## 5.4. SPI Timing Diagram

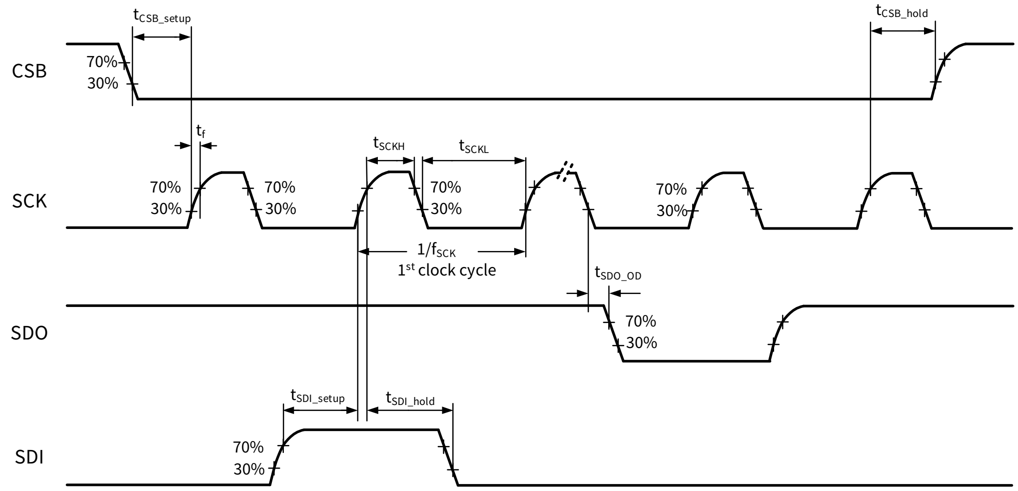



Figure 5.2 SPI Timing Diagram

## 5.5. SPI Electrical Characteristics

| ***Parameters*** | ***Symbol*** | ***Min*** | ***Typ*** | ***Max*** | ***Unit*** | ***Comments*** |
|---|---|---|---|---|---|---|
| Clock frequency | $f_{SCK}$ | | | 10 | MHz | Max load on SDI or SDO = 25pF |
| SLCK low pulse | $t_{SCKL}$ | 20 | | | ns | |
| SLCK high pulse | $t_{SCKH}$ | 20 | | | ns | |
| SDI setup time | $t_{SDI\_setup}$ | 20 | | | ns | |
| SDI hold time | $t_{SDI\_hold}$ | 20 | | | ns | |
| SDO/SDI output delay | $t_{SDO\_OD}$ | | | 30 | ns | Load = 25pF |
| | | | | 40 | ns | Load = 250pF |
| CSB setup time | $t_{CSB\_setup}$ | 20 | | | ns | |
| CSB hold time | $t_{CSB\_hold}$ | 40 | | | ns | |

# 6. Function Description

## 6.1. Overview

NSPGS2E uses a MEMS piezoresistive differential pressure sensor element as a pressure sensitive component that provide an original signal output that is proportional to ambient pressure. The built-in conditioning IC drives the sensitive component and amplifies, temperature compensates, and linearizes the original signal to output a voltage signal that is linear with the applied pressure.

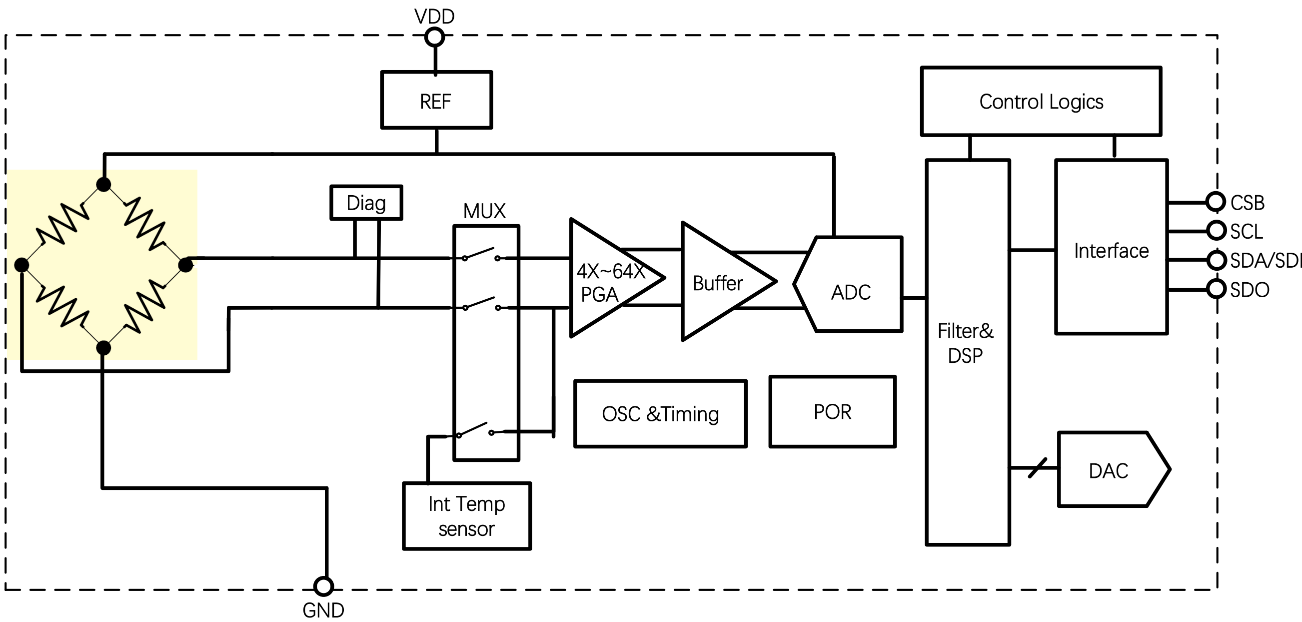



Figure 6.1 Product Function Block Diagram

## 6.2. Digital Output Transfer Function

$$Code = (A \times P + B) \times 8388607$$

Code is the register 0x06~0x08 value.

P is the pressure value, gauge pressure, unit is kPa.

Table 6.1 Digital Output Transfer Function Coefficient

| Product Type | Pressure Range | | Output Range | | Gain and Offset | |
|---|---|---|---|---|---|---|
| | $P_L$ | $P_H$ | $O_L$ | $O_H$ | A | B |
| NSPGS2E170DT41 | -50kPa | 120kPa | 838861 | 7549746 | 0.00471 | 0.33529 |

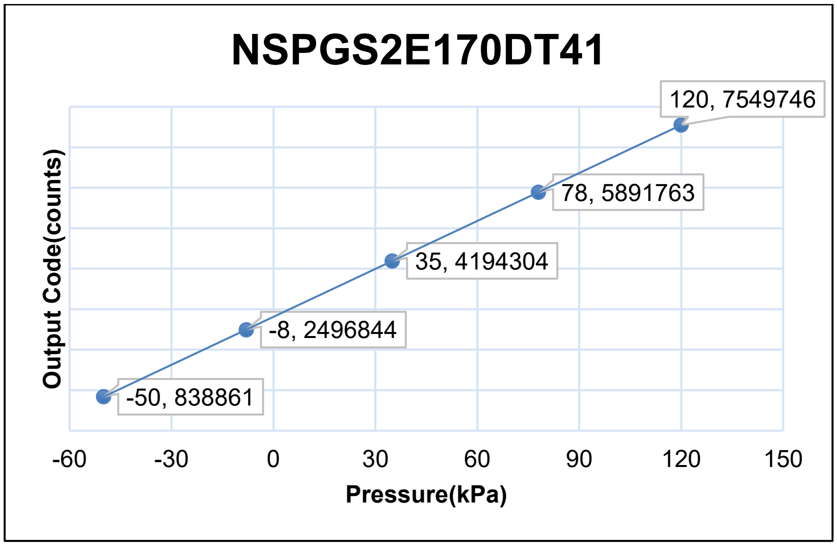



Figure 6.2 Digital Output Transfer Function

Register Map：

| ***Addr*** | ***Bit Addr*** | ***Description*** | ***Default*** | ***Description*** |
|---|---|---|---|---|
| 0x00 | 7,0 | SDO_ACTIVE | 1’b0<br>1’b0 | Set either of these two bits to 1 for SPI-wire:<br>0: SPI 3-wire<br>1: SPI 4-wire (SDO as serial output) |
| | 6 – 1 | Reserve | 6’b000000 | |
| 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.<br>Code = Data0x06*2^16+ Data0x07*2^8+ Data0x08； |
| 0x07 | 7 – 0 | PDATA<15:8> | 0x00 | |
| 0x08 | 7 – 0 | PDATA<7:0> | 0x00 | |
| 0x6C | 7 – 0 | Reserve | 0x02 | Default value: 0x02. |

For example:

If the value of the registers 0x06、0x07、0x08 are 0x2A, 0xEA, 0xEA, according to NSPGS2E170DT41 transfer function, Code = 2812651, P(kPa) = (2812650/8388607-B)/A, and finally get the value of pressure about 0kPa.

## 6.3. 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 NSPGS2E is shown below.

Table 6.2 I²C Address

| 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.

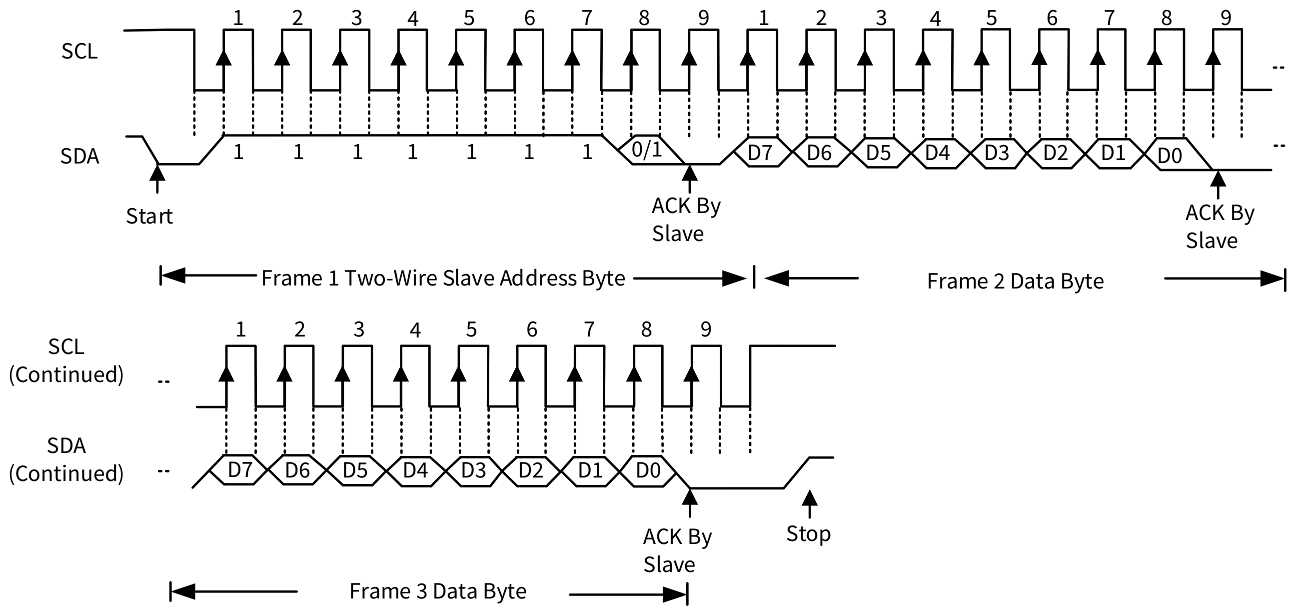



Figure 6.3 I²C Protocol

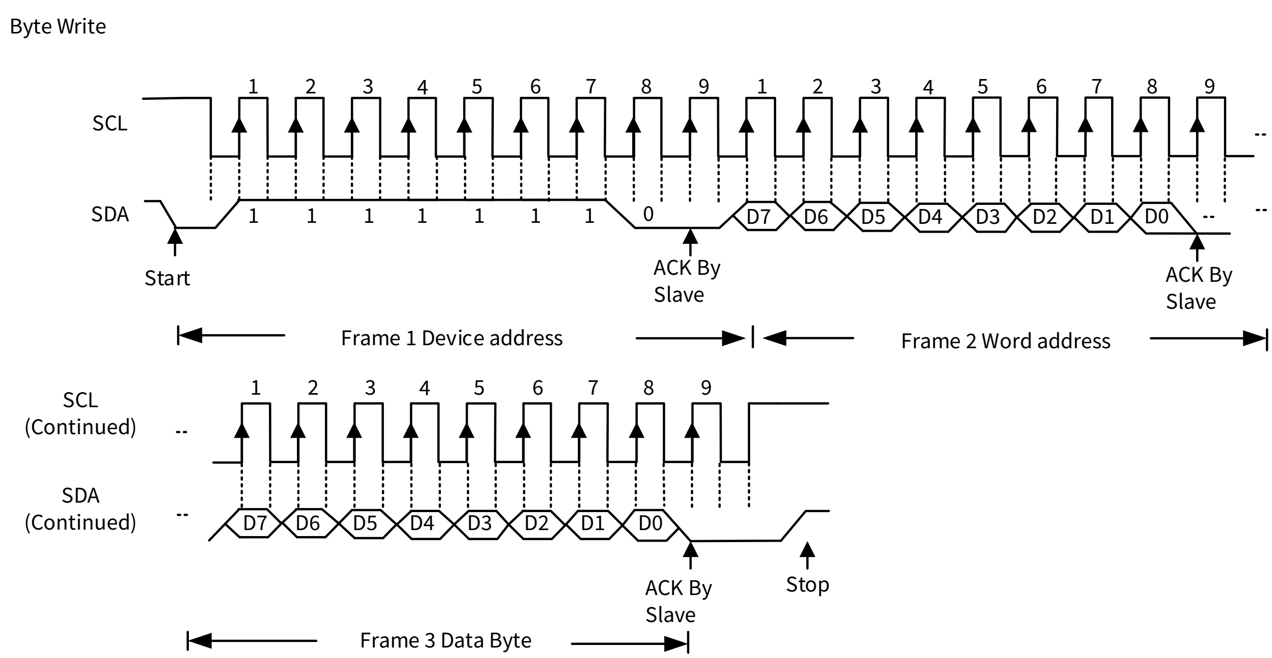



Figure 6.4 I²C Write Byte

Random Read

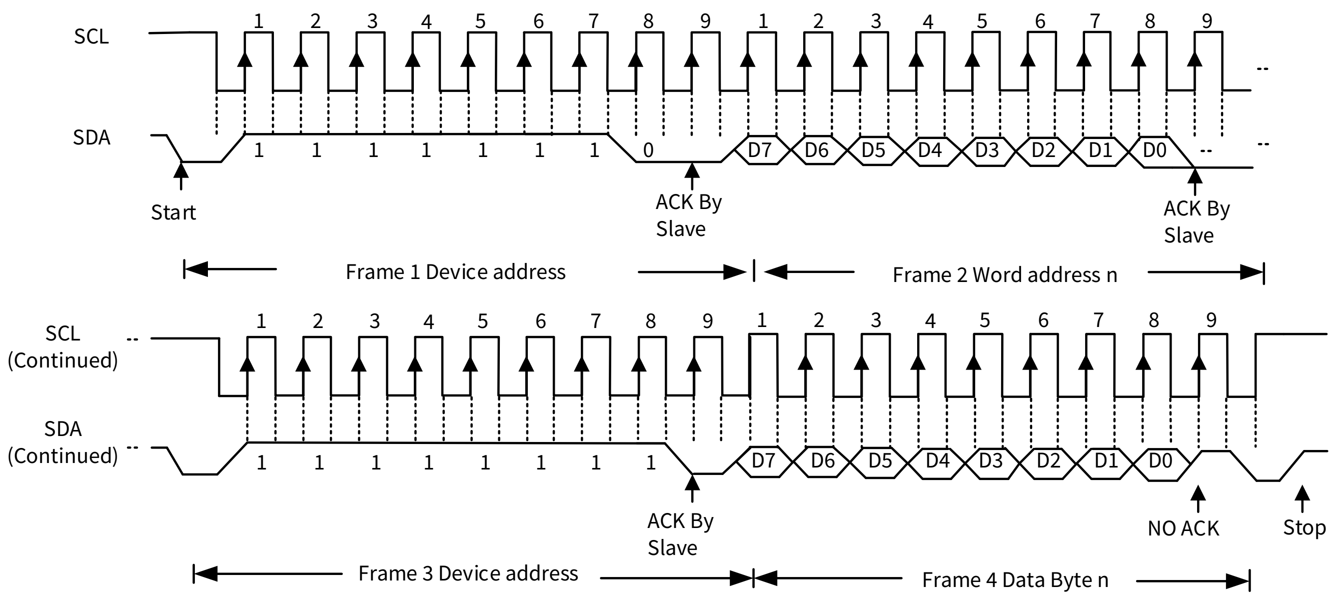



Figure 6.5 $I^2C$ Read Byte

## 6.4. SPI Interface

The falling edge of CSB, in conjunction with the rising edge of SCK, determines the start of framing. Once the beginning of the frame has been determined, timing is straightforward. The first phase of the transfer is the instruction phase, which consists of 16 bits followed by data that can be of variable lengths in multiples of 8 bits. If the device is configured with CSB tied low, framing begins with the first rising edge of SCK.

The instruction phase is the first 16 bits transmitted. As shown in Figure 6.6, the instruction phase is divided into a number of bit fields.

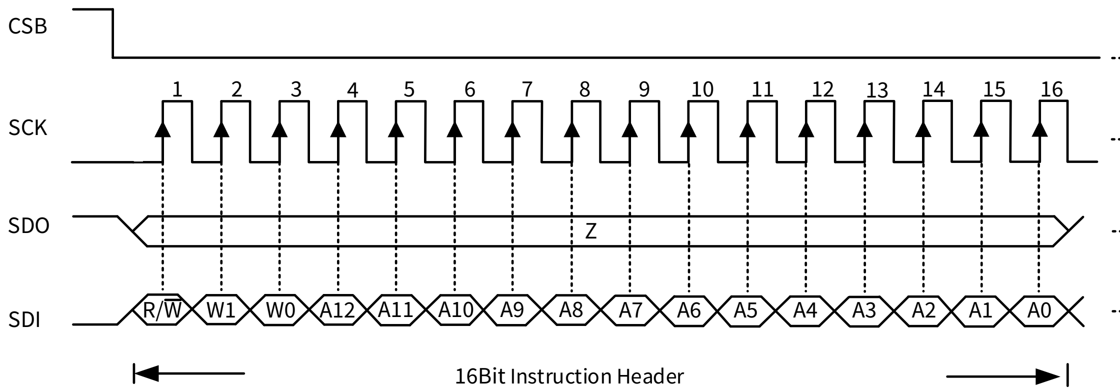



Figure 6.6 Instruction Phase Bit Field

The first bit in the stream is the read/write indicator bit (R/$\overline{W}$). When this bit is high, a read is being requested, otherwise indicates it is a write operation.

W1 and W0 represent the number of data bytes to transfer for either read or write (Table 6.3). If the number of bytes to transfer is three or less (00, 01, or 10), CSB can stall high on byte boundaries. Stalling on a nonbyte boundary terminates the communications cycle. If these bits are 11, data can be transferred until CSB transitions high. CSB is not allowed to stall during the streaming process.

The remaining 13 bits represent the starting address of the data sent. If more than one word is being sent, sequential addressing is used, starting with the one specified, and it either increments (LSB first) or decrements (MSB first) based on the mode setting.

Table 6.3 W1 and W0 settings

| ***W1:W0*** | ***Action*** | ***CSB Stalling*** |
|---|---|---|
| 00 | 1 byte of data can be transferred. | Optional |
| 01 | 2 bytes of data can be transferred. | Optional |
| 10 | 3 bytes of data can be transferred. | Optional |
| 11 | 4 or more bytes of data can be transferred. CSB must be held low for entire sequence; otherwise, the cycle is terminated. | No |

Data follows the instruction phase. The amount of data sent is determined by the word length (Bit W0 and Bit W1). This can be one or more bytes of data. All data is composed of 8-bit words.

Data can be sent in either MSB-first mode or LSB-first mode (by setting ‘LSB_first’ bit). On power up, MSB-first mode is the default. This can be changed by programming the configuration register. In MSB-first mode, the serial exchange starts with the highest-order bit and ends with the LSB. In LSB-first mode, the order is reversed. (Figure 6.7)

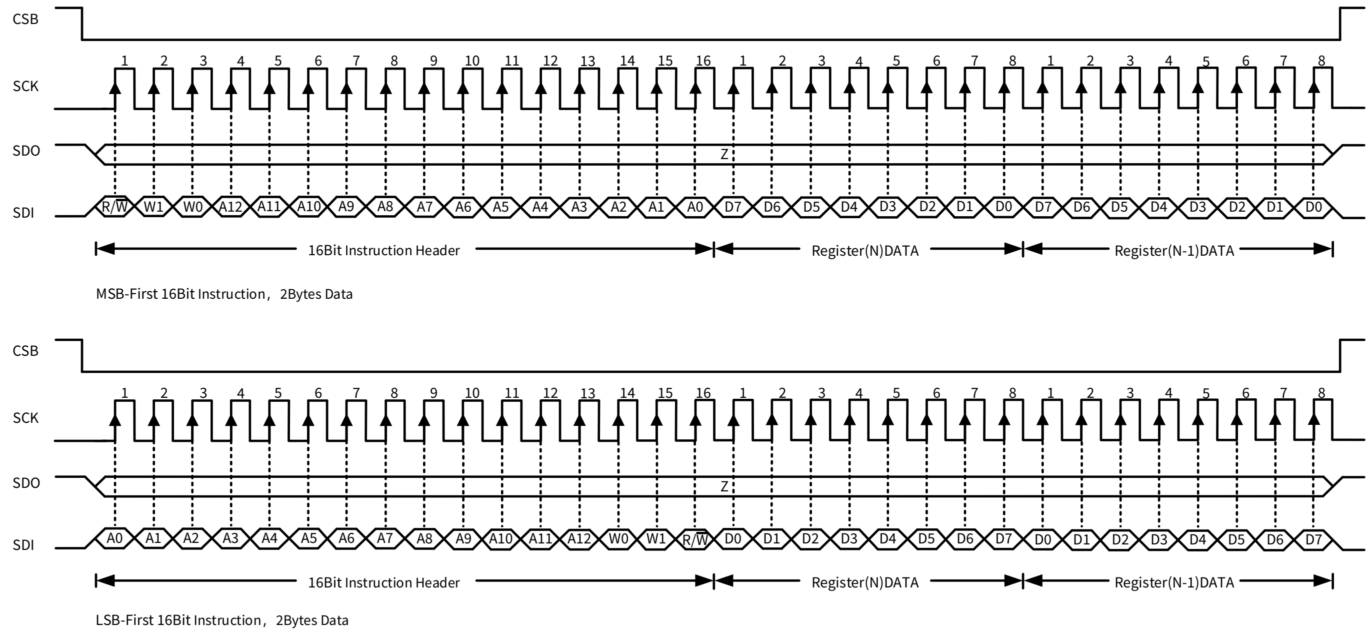



Figure 6.7 MSB First and LSB First Instruction and Data Phases

Byte Write

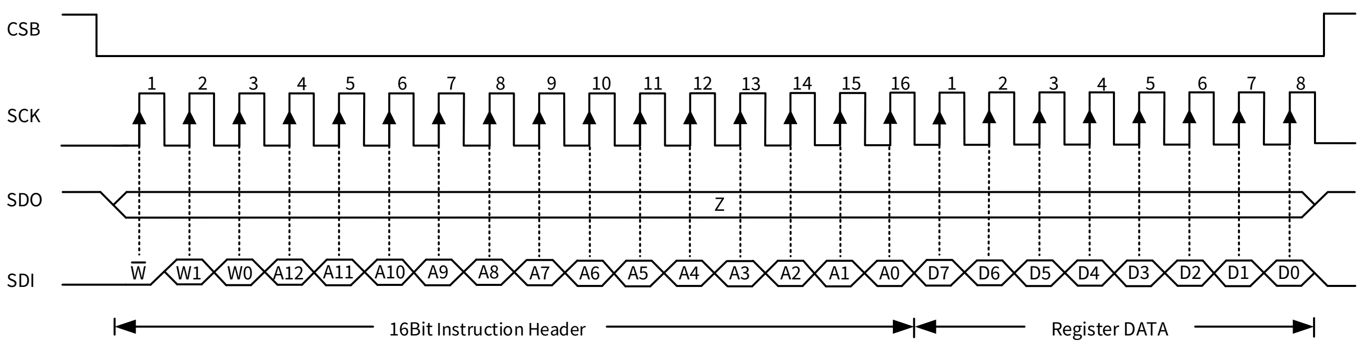



Figure 6.8 SPI Write Byte

Byte Read

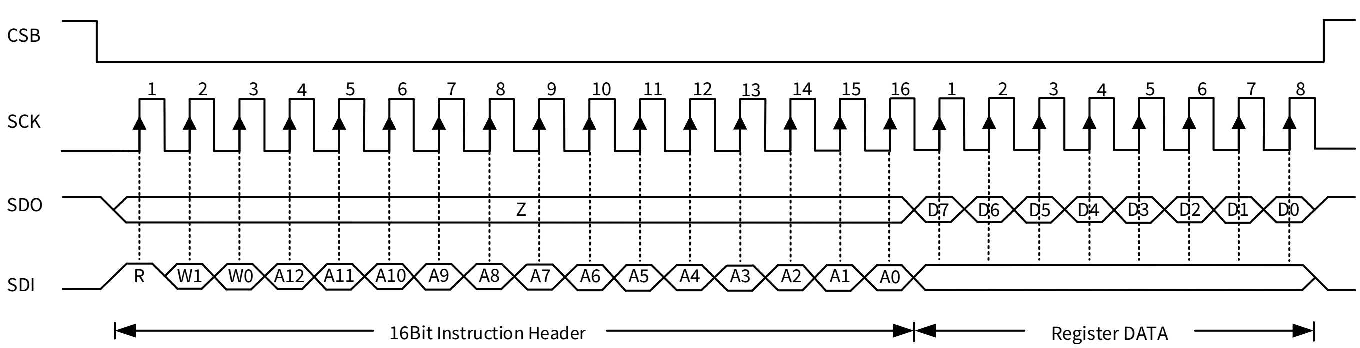



Figure 6.9 SPI Read Byte

# 7. Typical Application

## 7.1. Application Circuit

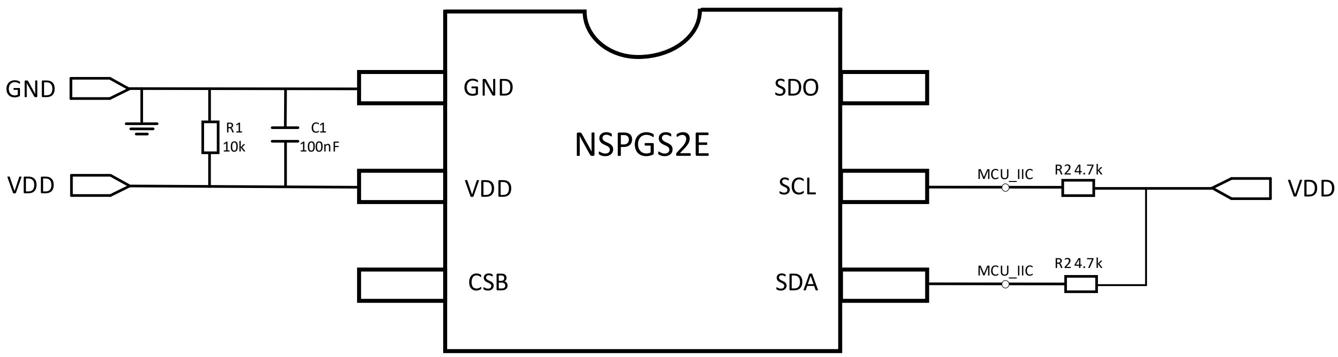



Figure 7.1 $I^2C$ Output Application Circuit

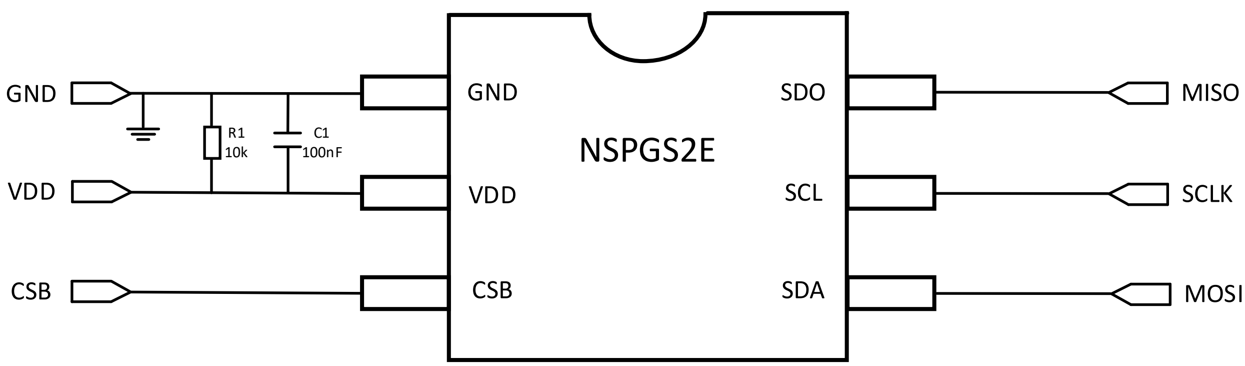



Figure 7.2 SPI Output Application Circuit

# 8. Package Information

## 8.1. Package Size

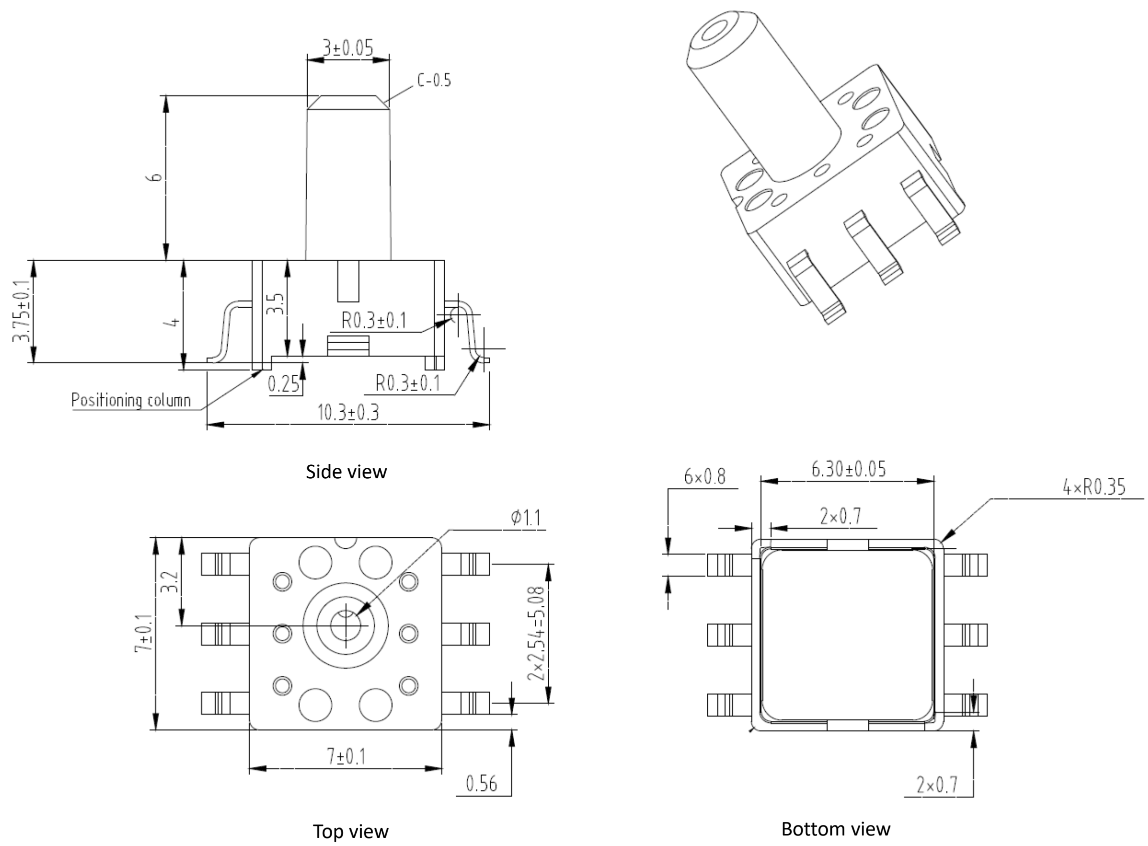



Figure 8.1 Package Outline mm

## 8.2. Recommended Footprint

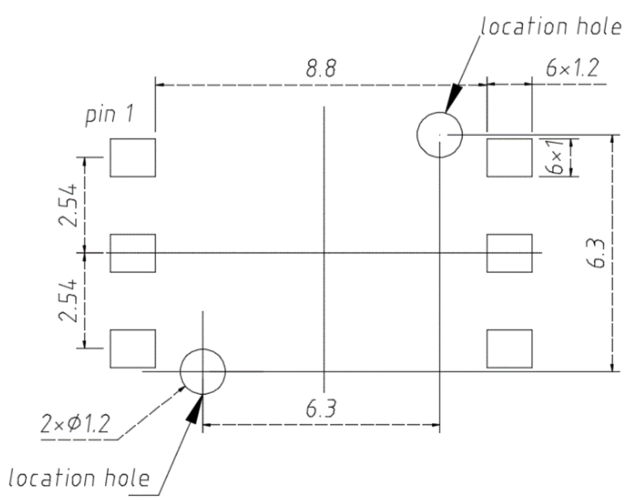



Figure 8.2 Footprint mm

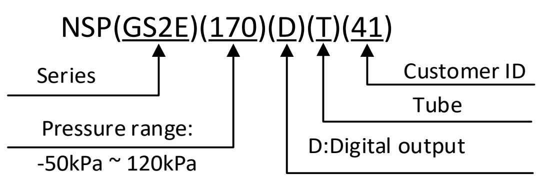
# 9. Order Information

| Product Type | Output Type | Pressure Range | | Output Range | | Gain and Offset | | Supply Voltage | Accuracy@-40~115°C | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $P_L$ | $P_H$ | $O_L$ | $O_H$ | A | B | | Initially | Full Life |
| NSPGS2E170DT41 | $I^2C$/SPI | -50.00kPa | 120.00kPa | 838861 | 7549746 | 0.00471 | 0.33529 | 5.0V | ±1.5% | ±2.5% |

Please scan the following QR code or visit the download link for complete part number list.

https://www.novosns.com/Public/Uploads/uploadfile4/NSPGS2E.pdf

Naming Convention:



# 10. Soldering Parameters

## 10.1. Reflow Soldering (SMD Terminal)

Table 10.1 Soldering Parameters

| ***Reflow Condition*** | | ***Lead–free Assembly*** |
|---|---|---|
| Pre Heat | Temperature Min ($T_s$(min)) | 150°C |
| | Temperature Max ($T_s$(max) | 200°C |
| | Time (min to max) ($t_s$) | 60 – 180 secs |
| Average ramp up rate (Liquidus Temp ($T_L$) to peak | | 3°C/second max |
| TS (max)to TL – Ramp-up Rate | | 3°C/second max |
| Reflow | Temperature ($T_L$) (Liquidus) | 217°C |
| | Time (min to max) ($t_L$) | 60 – 150 seconds |
| Peak Temperature ($T_P$) | | 260°C |
| Time within 5°C of actual peak Temperature ($t_p$) | | 20 – 40 seconds |
| Ramp-down Rate | | 6°C/second max |
| Time 25°C to peak Temperature ($T_P$) | | 8 minutes max |
| Do not exceed | | 260°C |

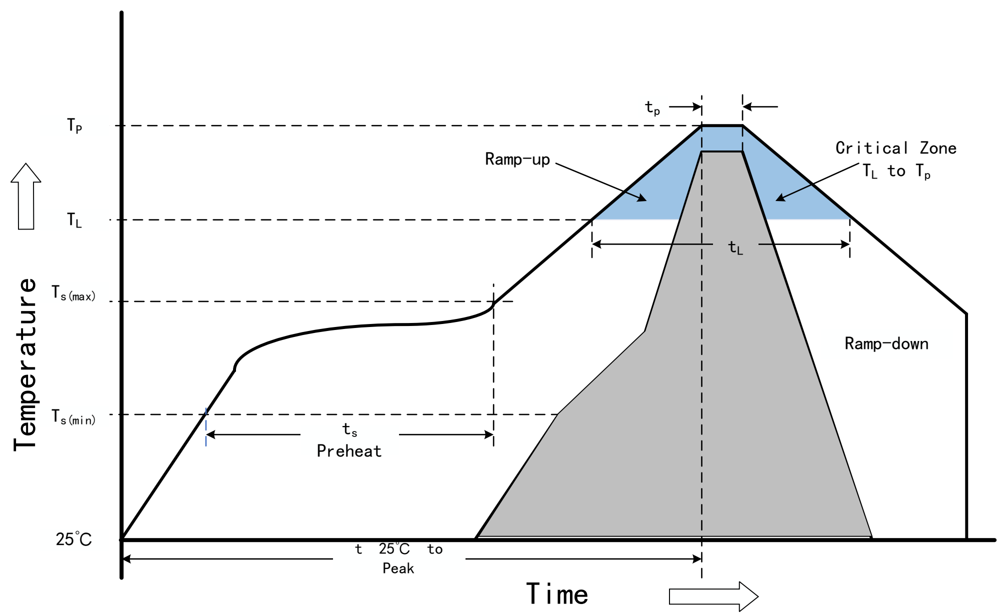



Figure 10.1 Reflow Soldering Curve

## 10.2. Manual Soldering

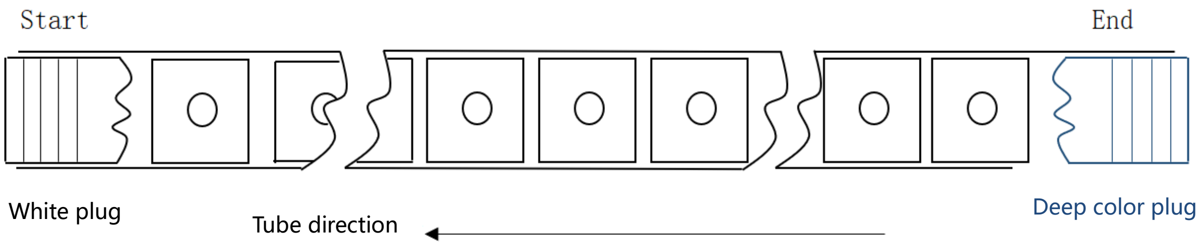
· Raise the temperature of the soldering tip between 260 °C and 300 °C and solder within 5 seconds.

· Use a flattened soldering tip when performing rework on the solder bridge.

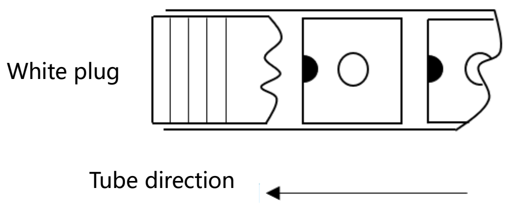
· Complete rework in one time.

# 11. Packing Information

This series product using tube package, each tube contains 70ea devices. Each tube has a deep color plug at the bottom and a white plug at the top, as follows:

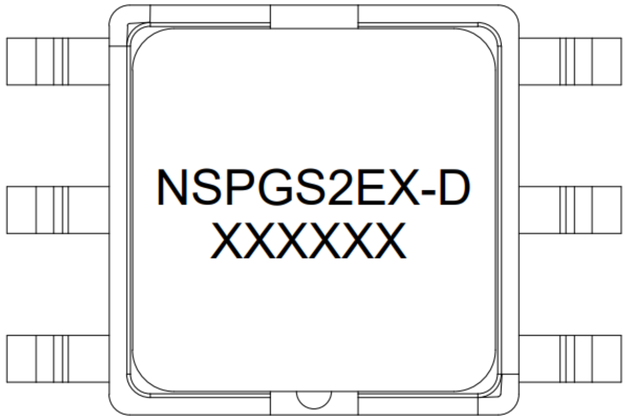



Pin1 point faces to the white plug at the top:




Minimum ordering quantity (MOQ): 1400EA.
Standard pack quantity (SPQ): 700EA.

# 12. Identification Code




NSPGS2EX: Product series.
D: $I^2C$/SPI
XXXXXX: Product serial number.

# 13. Revision History

| Revision | Description | Date |
|---|---|---|
| 1.0 | Release Version. | 2025/3/28 |

## Notes：

### 1. I²C code

```
#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 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 NSPGS2E170DT41_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 NSPGS2E170DT41_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 NSPGS2E170DT41_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);
    pBuffer[2]=IIC_Receive(NACK);
    IIC_Stop();
}

void main()
{
        uchar PData[3]={0,0,0};
        while(1)
        {
                NSPGS2E170DT41_Write_Byte(0x30,0x0A);
                while(1)                                        //Check whether the conversion ends
                {
                        if(number<=50)
                        {
```

```
                number++;
                delay_ms(1);
                NSPGS2E170DT41_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;
            }
        }
        NSPGS2E170DT41_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)Pdata/8388607-0.33529)/0.00471;      //PCode=(AxP+B)*8388607
P=(PCode/8388607-B)/A
                                                                 //A=0.00471, B=0.33529
                                                    //PNormalized=PCode/8388607
    }
}
```

## 2. SPI code

```
u8 REG06=0,REG07=0,REG08=0;
u8 number=1;
u8 PData[3]={0};
u32 PCode=0,Pdata=0;
float Pressure=0.0;
void NSPGS2E170DT41_SPI_Init(void)
{
	SPI_PORT_GPIO_Config();
	CSB(1);
	SCLK(0);
	SDI(1);
}

void NSPGS2E170DT41_SPI_Write_OneByte(u8 addr,u8 val)
{
	u8 i=0; u16 dat;

	dat=0x0000+addr;
	CSB(0);
	delay_us(2);

	for(i=0;i<16;i++)
	{
		SCLK(0);
		if(dat&0x8000)
			SDI(1);
		else
			SDI(0);
		delay_us(2);
		dat<<=1;
		SCLK(1);
		delay_us(2);
	}

	for(i=0;i<8;i++)
	{
		SCLK(0);
		if(val&0x80)
			SDI(1);
		else
			SDI(0);
		delay_us(2);
		val<<=1;
		SCLK(1);
		delay_us(2);
	}

	SCLK(0);
	CSB(1);
	delay_us(2);

}
u8 NSPGS2E170DT41_SPI_Read_OneByte(u8 addr)
{
	u8 i=0;   u16 dat; u8 val=0;

	dat=0x8000+addr;
	CSB(0);
	delay_us(2);
```

```
	for(i=0;i<16;i++)
	{
		SCLK(0);
		if(dat&0x8000)
		{
			SDI(1);
		}
		else
		{
			SDI(0);
		}
		delay_us(2);
		dat <<= 1;
		SCLK(1);
		delay_us(2);
	}

	for(i=0;i<8;i++)
	{
		SCLK(0);
		val<<=1;
		if(SPI_MISO)
			val++;
		delay_us(2);
		SCLK(1);
		delay_us(2);
	}

	SCLK(0);
	CSB(1);
	delay_us(2);

	return val;
}

void NSPGS2E170DT41_SPI_Read_3Byte(u8 addr,u8* pBuffer)
{
	u8 i=0;   u16 dat; u8 val_1=0,val_2=0,val_3=0;

	dat=0xC000+addr;
	CSB(0);
	delay_us(2);

	for(i=0;i<16;i++)
	{
		SCLK(0);
		if(dat&0x8000)
		{
			SDI(1);
		}
		else
		{
			SDI(0);
		}
		delay_us(2);
		dat <<= 1;
		SCLK(1);
		delay_us(2);
	}

	for(i=0;i<8;i++)
```

```
	{
		SCLK(0);
		val_1<<=1;
		if(SPI_MISO) val_1++;
		delay_us(2);
		SCLK(1);
		delay_us(2);
	}

	for(i=0;i<8;i++)
	{
		SCLK(0);
		val_2<<=1;
		if(SPI_MISO) val_2++;
		delay_us(2);
		SCLK(1);
		delay_us(2);
	}

	for(i=0;i<8;i++)
	{
		SCLK(0);
		val_3<<=1;
		if(SPI_MISO) val_3++;
		delay_us(2);
		SCLK(1);
		delay_us(2);
	}
	pBuffer[0]=val_1;
	pBuffer[1]=val_2;
	pBuffer[2]=val_3;

	SCLK(0);
	CSB(1);
	delay_us(2);
}

void NSPGS2E170DT41_SPI_Read_MultiByte(u8 addr,u8 len,u8 *pBuffer)
{
	u8 i=0,k=0,val=0; u16 dat;

	dat=0xE000+addr;
	CSB(0);
	delay_us(2);

	for(i=0;i<16;i++)
	{
		SCLK(0);
		if(dat&0x8000)
		{
			SDI(1);
		}
		else
		{
			SDI(0);
		}
		delay_us(2);
		dat <<= 1;
		SCLK(1);
		delay_us(2);
	}
```

```
		for(k=0;k<len;k++)
	{
		for(i=0;i<8;i++)
		{
			SCLK(0);
			val<<=1;
			if(SPI_MISO)val++;
			delay_us(2);
			SCLK(1);
			delay_us(2);

		}
		pBuffer[k]=val;

	}

	SCLK(0);
	CSB(1);
	delay_us(2);
}

int main(void)
{
	NSPGS2E170DT41_SPI_Init();
	delay_ms(100);
	NSPGS2E170DT41_SPI_Write_OneByte(0x00,0x81);

	while(1)
	{
		NSPGS2E170DT41_SPI_Write_OneByte(0x30,0x0A);
		while(1)				//Check whether the conversion ends
		{
			if(number<=50)
			{
				number++;
				delay_ms(1);
				if(0x02== NSPGS2E170DT41_SPI_Read_OneByte(0x30))
				{
					number=1;
					break;
				}
			}
			if(number>50)
			{
				number=1;
				break;
			}
	}

	NSPGS2E170DT41_SPI_Read_3Byte(0x08,PData);
	REG08=PData[0];				//Register 0x08
	REG07=PData[1];				//Register 0x07
	REG06=PData[2];				//Register 0x06
	PCode=(REG06*65536+REG07*256+REG08);	//PCode = Data0x06*2^16+ Data0x07*2^8+ Data0x08

	if (PCode>8388607)
		Pdata=PCode-16777215;			//Symbol processing
	else Pdata=PCode;
	Pressure=((float)Pdata/8388607-0.33529)/0.00471; //PCode=(AxP+B)*8388607	P=(PCode/8388607-B)/A
						//A=0.00471, B=0.33529
						//PNormalized=PCode/8388607
```

```
        }
}
```

# 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's 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**