Table of contents

1. IIC Work requirements

1.1. I^{2}C Communication Electrical Characteristics

parameter	marked	minimum value	typical value	maximum value	unit	Remark
Clock frequency	f_{scl}			400	kHz	
Clock low pulse hold time	t_{LOW}	1.3			us	
Clock high pulse hold time	t_{HIGH}	0.6			us	
SDA establishment time	t_{SUDATB}	0.1			us	
SDA hold time	t_{HDDAT}	0.0			us	
build time per start	t_{SUSTA}	0.6			us	
start condition hold time	t_{HDSTAB}	0.6			us	
stop time build time	t_{SUSTO}	0.6			us	
Interval time between two communications	t_{BUF}	1.3			us	

2. Functional description

2.1. register definition

address	bit address	register name	Defaults	describe
0x30	7 - 0	CMD<7:0>	0x03	0x00 : command mode, all EEPROM registers can only be written in command mode enter; 0x01/0x02: single work mode 0x03 : continuous working mode 0x33 : Enter EEPROM programming mode
0x06	7 - 0	PDATA<23:16>	0x00	Signed, 2 's complement: stores calibrated pressure sensor data ; When ' RAW_P'= 1,store the ADC output of the main signal channel ; when ' RAW_P' =0, store the corrected sensor data Code_P = 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<23:16>	0x00	Signed numbers, 2 's complement: When ' RAW_T '= 1 , store the ADC output code of the temperature channel; when ' RAW_T '=0 , store the corrected temperature data. LSB=1/2^16 ℃ Code_T = Data0x09*2^16+ Data0x0A*2^8+ Data0x0B ;
0x0A	7 - 0	TDATA<15:8>	0x00	
0x0B	7 - 0	TDATA<7:0>	0x00	

2.2. Data normalization calculation (range [-1 , 1])

Pressure Conversion : Take's Complement

1. After power on, directly read the three calibrated register data of 0x06 , 0x07 and 0x08 to form a 24bit pressure AD value. The highest bit is the sign bit. When the sign bit value is " 1 ", it means "negative", and the sign bit value is " 0 " means "positive".

2. Perform the following operations on the read AD value to calculate the pressure value :

eg : Suppose the decimal values read by REG0x06 , REG0x07 , REG0x08 are x , y , z

Pressure AD (Code) value: m=x*2^16+y*2^8+z

Positive and negative processing: if m>2^23 , it is a negative value, normalized value =(m-2^24)/2^23 ; if m<2^23 , it is a positive value, normalized value = m/2^23 ;

Temperature Conversion : Take Complement Code

1. Read the calibrated register data of 0x09 , 0x0A , and 0x0B to form a 24bit temperature AD value. The highest bit is the sign bit. When the sign bit value is " 1 ", it means "negative", and when the sign bit value is " 0 " Represents "positive".

2. Perform the following operations on the read AD value to calculate the temperature value :

eg : Suppose the decimal values read by REG0x09 , REG0x0A , REG0x0B are x , y , z

Temperature AD (Code) value: m=x*2^16+y*2^8+z

Positive and negative processing:

if m>2^23 , it is a negative value, temperature value (° C) =(m-2^24)/2^16+25 ;

if m<2^23 , it is a positive value , temperature value (° C) =m/2^16+25 ;

2.3. Digital Output Transfer Function

The pressure value can be converted to a digital output register value using the following equation:

$$Y = K\mathrm{x} + B$$

Among them , Code is the chip register value; P is the actual pressure value, the unit is kPa ;

Table 2.3.1 Typical comparison table of digital output

Example part number	Pressure Range kPa		digital output code		transfer function coefficient	
	P_L	P_H	O_L	O_H		
NSA2860X/2862X	-100	100	838861	7549746		
			Numeric normalized value			
	Y_1	Y_2	X_1	X_2	K	B
	-100	100	0.1	0.9	250	-125

Combined with the Code to pressure conversion formula in 2.3 , the measured pressure value can be calculated; Example: When the values of registers 0x06 , 0x07 , and 0x08 are 0x40 , 0x00 , and 0x00 respectively , substitute them into the transfer function coefficient parameter calculation, and normalize the value = (64*2^16+0+0) / 2^23 = 0.5 , P(kPa) =(0.5*(250))-125 , the final pressure value is about 0kPa .

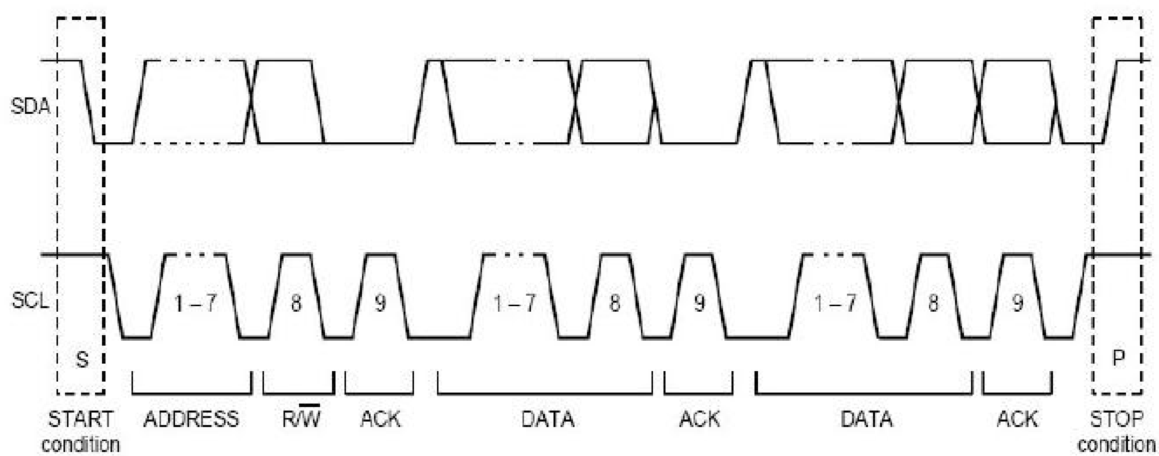
2.4. I²C Communication Interface Protocol

The I²C bus uses SCL and SDA as signal lines. These two lines are both connected to VDD through pull-up resistors , and are kept high when not communicating. The I²C device addresses of this series of products are as follows:

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

Table 2.4.1 I²C address

The I²C communication protocol has special start (S) and stop (P) conditions. When SCL is at high level, the falling edge of SDA marks the beginning of data transmission.
The I²C master sequentially sends the slave's address (7 bits) and the read / write control bits. When the slave device recognizes this address, it generates an acknowledge signal and pulls SDA low in the ninth cycle. After getting the response from the device, the master device continues to send the 8 -bit register address, and continues to send or read data after getting the response. SCL is at a high level, and a rising edge occurs on SDA to mark the end of I²C communication. In addition to the start and end flags, when SCL is high
The data transmitted by SDA must remain stable. The value transmitted by SDA can be changed while SCL is low .
All data transmission in I²C communication takes 8 bits as the basic unit, and a response signal is required



after every 8 -bit data transmission to keep the transmission going.

Figure 2.4.2 I²C protocol

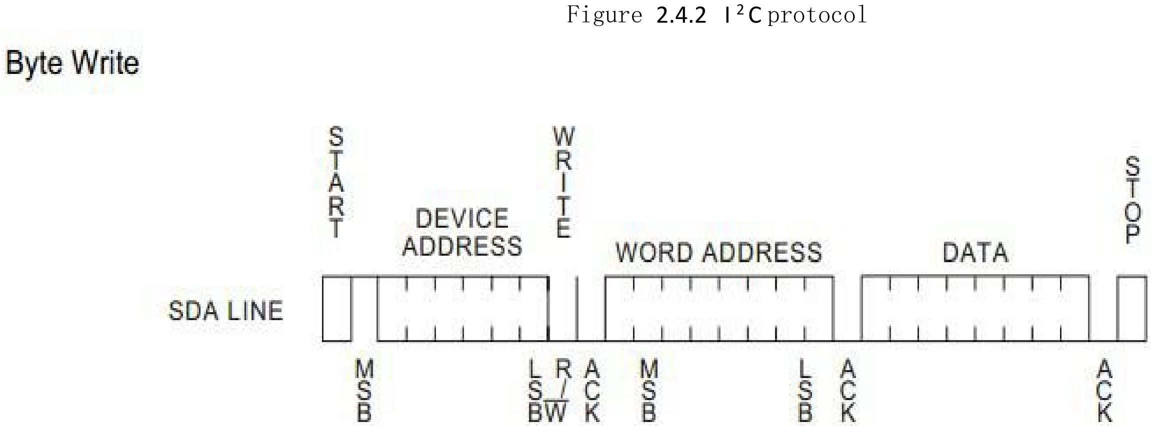


Figure 2.4.3 I²C write timing

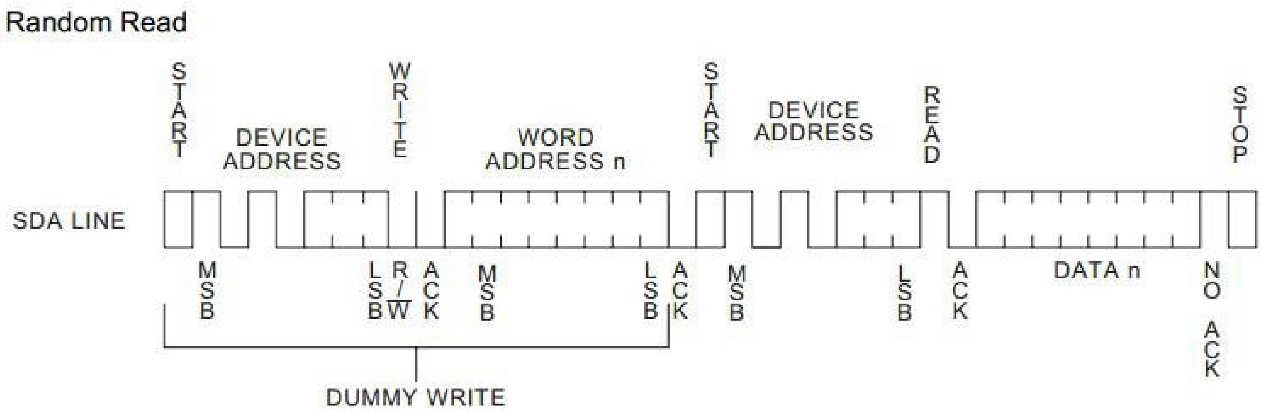


Figure 2.4.4 I²C read timing

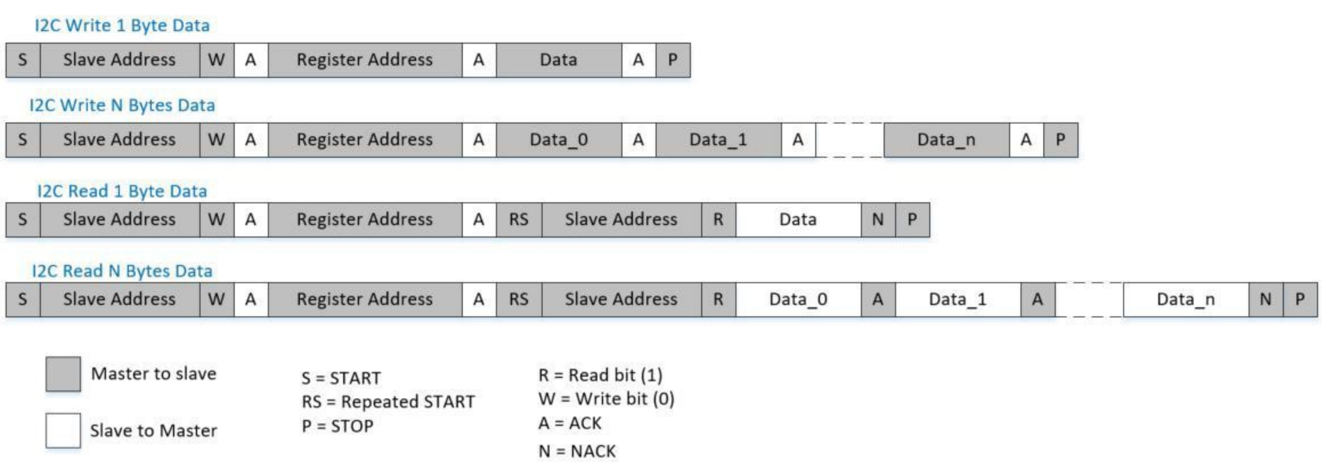


Figure 2.4.5 Read register logic

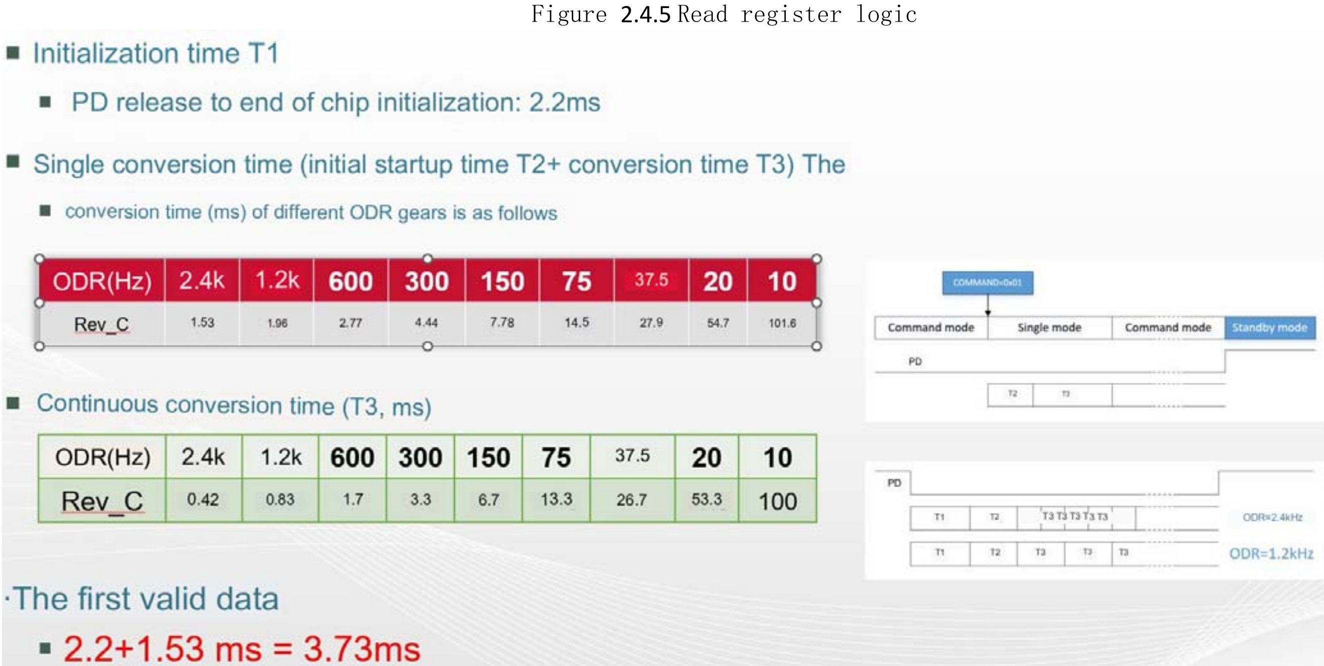


Figure 2.4.6 NSA2862X read data parameter configuration for the first time

3. revise history

revision	Description	date
0.1	initial version	2022/10/20

Notes:

1. I²C routine

```
void IIC_Init(void)      //IIC initialization
{
    SCL_H;
    SDA_H;
    SCL_W;
    SDA_W;
}

void IIC_Start(void)     //IIC Start Signal
{
    SDA_W;
    SCL_H;
    SDA_H;
    delay10us();
    SDA_L;
    delay10us();
}

void IIC_Stop(void)      //IIC Stop Signal
{
    SCL_L;
    delay10us();
    SCL_H;
    SDA_W;
    SDA_L;
    delay10us();
    SDA_H;
    delay10us();
}

void IIC_ACK(void)       //IIC Ack Signal
{
    SDA_W;
    SDA_L;
    SCL_H;
    delay10us();
    SCL_L;
}

void IIC_NACK(void)      //IIC NAck Signal
{
    SDA_W;
    SDA_H;
    SCL_H;
    delay10us();
    SCL_L;
```

```
}

uchar IIC_Wait_ACK(void)        //IIC Ack wait for judgment
{
    int ErrTime=0;
    SDA_R; SCL_H;
    delay10us();
    while(Read_SDA)
    {
        ErrTime++;
        if(ErrTime>200)
        {
            IIC_Stop();
            return 1;
        }
    }
    SCL_L;
    SDA_W;
    SDA_L;
    delay10us();
    return 0;
}

void IIC_Send(uchar IIC_Data)   //IIC send
{
    uchar  i;
    SDA_W;
    SCL_L;
    delay10us();
    for(i=0;i<8;i++)
    {
        if((IIC_Data&0x80)>>7)
            SDA_H;
        else
            SDA_L;
        IIC_Data<<=1;
        SCL_H;
        delay10us();
        SCL_L;
        delay10us();
    }
}

uchar IIC_Receive(uchar ACK)    //IIC receive
{
    uchar i,Receive_Data=0x00;
    SDA_R;
    for(i=0;i<8;i++)
    {
```

```
            SCL_L;
            delay10us();
            SCL_H;
            Receive_Data<<=1;
            if(Read_SDA==1)
                Receive_Data++;
            else
                ;
            delay10us();
    }
    SCL_L;
    delay10us();
    if(ACK==0x01)
            IIC_ACK();
    else
            IIC_NACK();
    return Receive_Data;
}

void IIC_Write_Byte(uchar WriteAddr,uchar WriteData)      //IIC single write data
{
    uchar flag;
    IIC_Start();
    IIC_Send(0xDA|0x00);
    IIC_Wait_ACK();
    IIC_Send(WriteAddr);
    IIC_Wait_ACK();
    IIC_Send(WriteData);
    IIC_Wait_ACK();
    IIC_Stop();
}

void IIC_Read_Byte(uchar ReadAddr, uchar *pBuffer)        //IIC IIC single read data
{
    IIC_Start();
    IIC_Send(0xDA|0x00);
    IIC_Wait_ACK();
    IIC_Send(ReadAddr);
    IIC_Wait_ACK(); IIC_Start();
    IIC_Send(0xDA|0x01);
    IIC_Wait_ACK();
    pBuffer[0]=IIC_Receive(0);
    IIC_Stop();
}

void IIC_Read_3Byte(uchar ReadAddr,uchar *pBuffer)        //IIC continuous read data (3byte)
{
    IIC_Start();
    IIC_Send(0xDA|0x00);
```

```
        IIC_Wait_ACK();
        IIC_Send(ReadAddr);
        IIC_Wait_ACK(); IIC_Start();
        IIC_Send(0xDA|0x01);
        IIC_Wait_ACK();
        pBuffer[0]=IIC_Receive(1);
        pBuffer[1]=IIC_Receive(1);
        pBuffer[2]=IIC_Receive(0);
        IIC_Stop();
}

Void Main()
{
        Float Pressure ,Temperature ;
        Float k ,b;        //Calculate the transfer function coefficient according to the measuring range
        uChar PData[3]={0,0,0};
        uChar TData[3]={0,0,0};
        /*************************************************************************************/
        #if 1              //NSA2862X， 1: and disable OWI; 0: not disable OWI
        Delay_3ms();
        #else
        Delay_25ms();
        #else if
        /*************************************************************************************/
        IIC_Init();
        #if 1              //NSA2862X， 1: Single conversion command: low power consumption mode; 0: Continuous conversion command
        IIC_Write_Byte(0x30,0x01);
        #else
        IIC_Write_Byte(0x30,0x03);
        #else if
        /*************************************************************************************/
        Delay_1ms();       //ODR_P&ODR_T is set to 2.4K rate, other rates see Figure 2.4.6  configuration below
        /*****************Read the temperature first, then the pressure*****************************/
        IIC_Read_3Byte(0x09,TData);
        Temperature = ((TData[0]*2^16 + TData[1]*2^8 + TData[2]) /2^16) +25
        IIC_Read_3Byte(0x06,PData);
        Pressure = ((PData[0]*2^16 + PData[1]*2^8 + PData[2]) /2^23) *k +b
}
```