1. IIC Work Requirements

1.1. I²C Electrical and Communication Characteristics

Parameters	Marking	Minimum Value	[illegible]	Maximum Value	Unit	备注
Clock Frequency	f_{scl}			400	kHz	
Clock Low Pulse Duration	t_{LOW}	1.3			us	
Clock High Pulse Duration	t_{HIGH}	0.6			us	
SDA Setup Time	t_{SUDAT}	0.1			us	
SDA Hold Time	t_{HDDAT}	0.0			us	
Duration of start condition maintenance	t_{SUSTA}	0.6			us	
Start condition hold time	t_{HDSTA}	0.6			us	
Time of establishment for stop time	t_{SUSTO}	0.6			us	
Interval time between two communications	t_{BUF}	1.3			us	

2. Function description

2.1. Definition of registers

Address	Bit address	Register name	Default value	Description
0x30	7 –4	Sleep_time<3:0>	4’b 0000	0000b：0ms，0001b：62.5ms...1111b：937.5ms，valid only in sleep mode
	3	Sco	1’b 0	1："Start data collection; automatically reset to 0 when collection ends
	2 –0	Measurement_ctrl<2:0>	3’b 000	000b: Single Temperature Measurement Mode 001b: Single Sensor Signal Measurement Mode 010b: Combined Measurement Mode (performs a temperature measurement followed immediately by a sensor signal measurement) 011b: Sleep Mode (periodically performs a combined measurement mode, with the interval time determined by 'sleep_time') 100b: Continuous Temperature Measurement Mode 101b: Continuous Sensor Signal Measurement Mode
0x06	7 –0	PDATA<23:16>	0x00	Signed integer, 2's complement: stores calibrated pressure sensor data; when 'RAW_P' = 1, stores the ADC output of the main signal channel; when 'RAW_P' = 0, stores the calibrated 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 integer, 2's complement: When 'RAW_T' = 1, stores the ADC output code of the temperature channel; when 'RAW_T' = 0, stores the calibrated temperature data. LSB= 1/2^8℃ Code_T ＝ Data0x09*2^ 16+ Data0x0A*2^8；
0x0A	7 － 0	TDATA<15:8>	0x00	

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

Pressure Conversion: Two's Complement

1 、After powering on, directly read the calibrated register data from 0x06, 0x07, and 0x08 to form a 24-bit pressure ADC value. The most significant bit is the sign bit, where a value of "1" indicates "negative" and "0" indicates "positive."

2 、Perform the following calculations to determine the pressure value:

eg： if the decimal values read from REG0x06, REG0x07, and

REG0x08 are x, y, and z respectively, the pressure ADC (Code) value

is: m=x*2^16+y*2^8+z

Sign Handling:

If m > 2^23, it indicates a negative value, and the normalized value is (m - 2^24) / 2^23.

If m < 2^23, it indicates a positive value, and the normalized value is m / 2^23.

Temperature Conversion: Two's Complement

1 、Read the calibrated register data from 0x09 and 0x0A to form a 24-bit temperature ADC value. The most significant bit is the sign bit, where a value of "1" indicates "negative" and "0" indicates "positive."

2 、Perform the following calculations to determine the temperature value:

eg：Assume that the decimal values read from

REG0x09 and REG0x0A are x and y. The temperature

ADC (Code) value is：

m=x*2^16+y*2^8

Sign Handling：
if m>2^23 ， it is a negative value. The temperature in degrees Celsius
(°C) is calculated as (m-2^24)/2^8；
if m<2^23 ， it is a positive value. The temperature in degrees Celsius
(°C) is calculated as m/2^8；

2.3. Digital Output Transfer Function

Using the following equation, the pressure value can be converted to a digital output register value:

$$Y = Kx + B$$

Where Code is the chip register value, and P is the actual pressure value in kPa；

Table 2.3.1 Typical Digital Output Reference Table

Example Part Number	Pressure Range kPa		Digital Output code		Transfer Function Coefficients	
	P_L	P_H	O_L	O_H		
NSA2300/2302	-100	100	838861	7549746		
			Digital Normalized Value			
	Y_1	Y_2	X_1	X_2	K	B
	-100	100	0.1	0.9	250	-125

Combining the Code-to-pressure conversion formula from section 2.3, the measured pressure value can be calculated；

eg：if the values of the 0x06, 0x07, and 0x08 registers are 0x40, 0x00, and 0x00 respectively, then: Using the transfer function coefficient parameters:

Normalize the value: = (64*2^16+0+0) / 2^23 = 0.5 ， P(kPa) =(0.5 *(250))- 125， Thus, the final pressure value is approximately 0 kPa.

2.4. I²C Communication Interface Protocol

I²C bus uses SCL (Serial Clock Line) and SDA (Serial Data Line) as signal lines. Both lines are connected to VDD via pull-up resistors and remain high when not communicating. The I2C device addresses for this series of products are as follows:

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

Table 2.4.1: I2C Address

In the I2C communication protocol, a special Start (S) and Stop (P) condition are used. Data transmission begins when SDA falls while SCL is high. The master device sends the 7-bit address of the slave device along with a read/write control bit. Upon recognizing the address, the slave sends an acknowledgment signal by pulling SDA low during the ninth clock cycle. After receiving this acknowledgment, the master continues by sending 8-bit register addresses and, upon acknowledgment, continues to send or read data. The communication ends with a Stop condition, where SDA rises while SCL is high. Data must be stable when SCL is high, and can change when SCL is low. All data in I2C communication is transferred in 8-bit units, with an acknowledgment bit required after every 8 bits to continue the transfer.

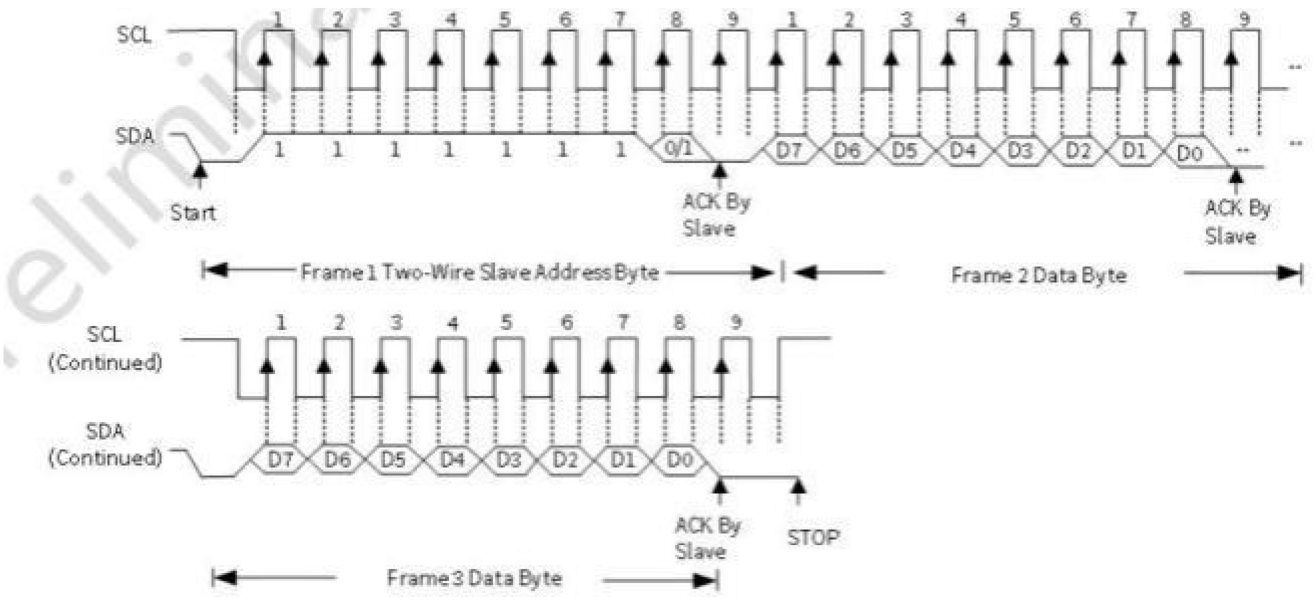

图 2.4.2 I²C 协议

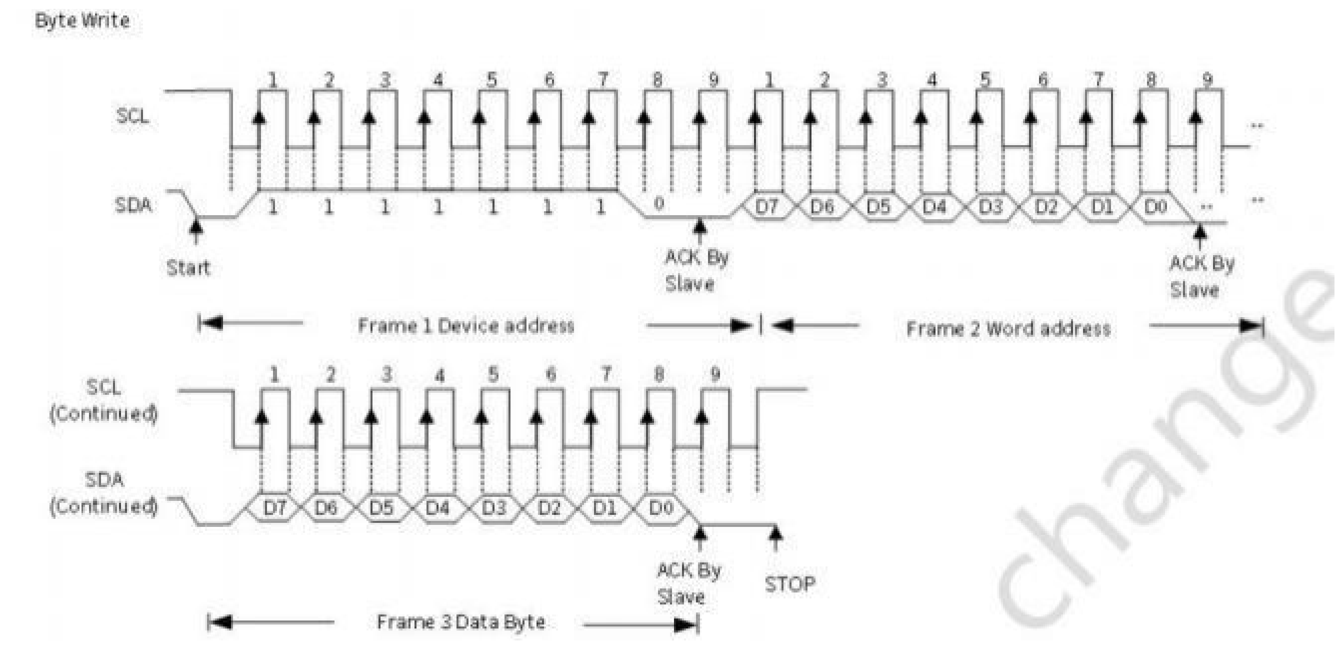

Figure 2.4.3 I²C write timing

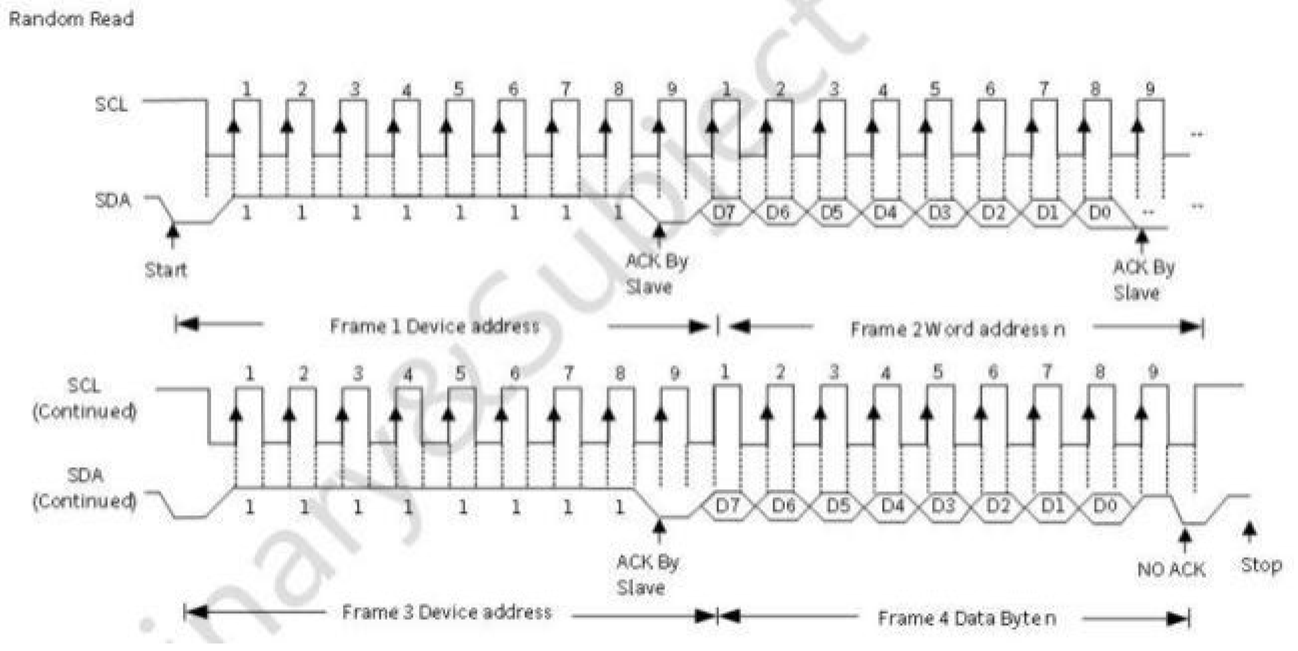

Figure 2.4.4 I²C read timing

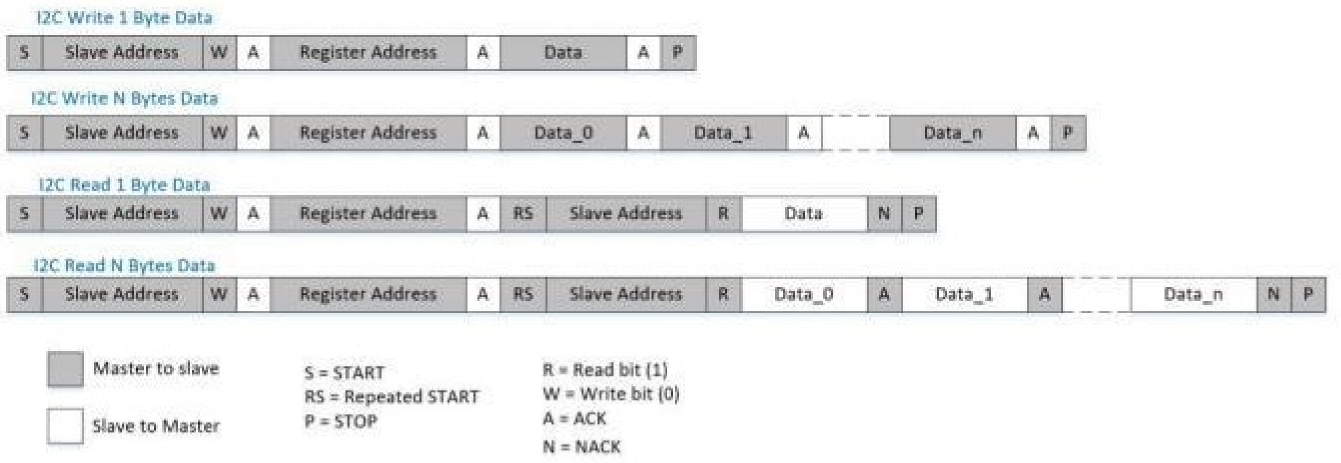
I2C Write 1 Byte Data
S | Slave Address | W | A | Register Address | A | Data | A | P
I2C Write N Bytes Data
S | Slave Address | W | A | Register Address | A | Data_0 | A | Data_1 | A | Data_n | A | P
I2C Read 1 Byte Data
S | Slave Address | W | A | Register Address | A | RS | Slave Address | R | Data | N | P
I2C Read N Bytes Data
S | Slave Address | W | A | Register Address | A | RS | Slave Address | R | Data_0 | A | Data_1 | A | Data_n | N | P
Master to slave
Slave to Master
S = START
RS = Repeated START
P = STOP
R = Read bit (1)
W = Write bit (0)
A = ACK
N = NACK

Figure 2.4.5 Register Read Logic

2.5. Change I2C Address(**NSA2302 Only**)

1 、Read the value of 0xA4 and store it as REGA4; Write the value REGA4 = REGA4 | 0x08 to 0xA4;

2 、Write I2C Address<6:0> | 0x80 to 0xA3.

For example, if the I2C address is 0xDA (1101 1010 for write) and 0xDB (1101 1011 for read), the address to write is 0x6D (0110 1101). Then, combine 0x6D (0110 101) with 0x80 (1000 0000), resulting in writing 0xED (1110 1101) to 0xA3.

3 、Write 0x40 to 0x6A, write 0x6A to 0x6C, and then wait 1 second for EEPROM programming to complete.

4 、To reset the chip and activate the new I2C address, either power cycle the device or write 0x24 to 0x00. After this, the new address will be effective, and both the new address and the wildcard addresses 0xFF/0xFE can access the NSA2302.

3. Revision History

Revision	Description	Date
0.1	Initial Version	2023/6/7

Notes:

1. I2C Example Code

```
void  IIC_Init(void)        //IIC Initialization
#define ACK  1
#define  NACK 0
 unsigned  char number=1;
 unsigned char  REG30[1]={0};
 unsigned char CalData[5]={0,0,0,0,0}
 Float Pressure ,Temperature ;
 Float k ,b;                        //Calculate the transfer function coefficients based on the range.

void IIC_Start(void)        //IIC Start signal: When SCL is high, SDA
                                     transitions from high to low
{
       IIC_SCL(1);          //SCL outputs high
      SDA_OUT(1);           //SDA outputs high
       Delay_us(2);
      SDA_OUT(0);           //SDA outputs low
       Delay_us(2);
}

void IIC_Stop(void)        //IIC Stop  signal: When SCL is high, SDA
                                     transitions from low to 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)          //IIC Ack signal (low level)
{
      SDA_OUT(0);
       IIC_SCL(1);
       Delay_us(2);
       IIC_SCL(0);
}

void IIC_NACK(void)        //IIC  NACK signal (high level)
{
      SDA_OUT(1);
       IIC_SCL(1);
       Delay_us(2);
       IIC_SCL(0);

}
```

```
unsigned char IIC_Wait_ACK(void)          //IIC Ack  wait check, return 0 indicates success; otherwise, error.
{

    int ErrTime=0;
    SDA_IN();
    IIC_SCL(1);                   //SDA Set to input mode
    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)   //IIC Send
{
    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);
    }
}

unsigned char IIC_Receive(uchar ACK)     //IIC Receive
{

    uchar i, Receive_Data=0;
    SDA_IN();
                                  //SDA Set to input mode
    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 IIC_Write_Byte(unsigned char WriteAddr,unsigned char WriteData)
// IIC Single write operation
{
        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(unsigned char ReadAddr, unsigned char *pBuffer)
// IIC Single read operation
{
        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(NACK);
        IIC_Stop();
}

void IIC_Read_5Byte(unsigned char ReadAddr,unsigned char *pBuffer)
//IIC Continuous read operation (5byte)
{
        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(ACK);
```

```
        pBuffer[1]=IIC_ Receive(ACK);
        pBuffer[2]=IIC_ Receive(ACK);
        pBuffer[3]=IIC_ Receive(ACK);
        pBuffer[4]=IIC_ Receive(NACK);
        IIC_Stop();
}

void Main()
{
    /*********************************************************************************/
    #if 1                              //NSA2862X， 1: OWI disabled; 0: OWI not disabled
    Delay_3ms();
    #else
    Delay_25ms();
    #else if
    /*********************************************************************************/
    while(1)
    {
            IIC_Write_Byte(0x30,0x0A);         //Combination conversion mode
            While(1)
            {
                    if(number<=50)
                    {
                            number++;
                            Delay_ms(1);
                            IIC_Read_Byte(0x30, REG30);
                            if(0x02 == REG30[1])
                            {
                                    Number=1;
                                    REG02[0]=REG02[1]=0;
                                    break;

                            }
                    }
                    if(number>50)
                    {
                            number=1;
                            //can add error messages, such as serial port errors
                            break;

                    }
            }
    }
    IIC_ Read_5Byte(0x06, CalData);
    Pressure = CalData[0] * 65536 + CalData[1] * 256 + CalData[2];
    if(Pressure > 8388607)
            Pressure = ((Pressure - 16777216) /8388607) *k +b         //Pressure value, with units according to the
                                                                         transfer function specification.
    else
            Pressure = (Pressure /8388607) *k +b
    Temperature = CalData[3]*2^ 16 + CalData[4]*2^8
    if(Temperature > 8388607)
            Temperature = (Temperature - 16777216) /256                // ℃
    else
            Temperature = Temperature /256
}
```