GZP6810D

Thermal Differential Pressure Sensor

Digital output(I2C)

Datasheet

Version：V1.0

Issued Date：2025.03.21

Table of Contents

Document Revision History

Revision	Description	Date
V1.0	Initial version	2025.03.21

1 Product Description

1.1 Features

- -500 to 500Pa range
- Excellent accuracy and very few offset drift
- High reliability and long-term stability
- Optimal signal-to-noise ratio
- Reliable performance
- Digital output
- Fast response time

1.2 Applications

- Portable ventilators, small household oxygen generators
- Continuous Positive Airway Pressure (CPAP) equipment
- Anesthesia delivery during labor
- Intensive care equipment
- Filter monitoring
- Residual pressure monitoring system for fire protection
- Medical breathing apparatus
- HVAC (Heating, Ventilation, and Air Conditioning) system

2 Function Description

The GZP6810 sensor series is a series of digital differential pressure sensors from Sencoch, designed specifically for high-precision micro- differential pressure applications. The sensor measures the pressure of air and non-corrosive gases with extremely high accuracy , capable of measuring very small pressures , and exhibits excellent measurement accuracy and zero drift. The GZP6810 series features a digital two-wire I2C interface for easy connection to microprocessors.

The sensor's superior performance is based on Sencoch's patented sensor technology. Differential pressure is measured by a thermal sensor element that measures the gas flowing through it. The proven technology is well-suited for high-quality mass production and is ideal for demanding and cost-sensitive OEM applications.

2.1 Electrical Characteristics

Power supply : Unless otherwise specified, the following parameters were tested at 5V±0.1V. Reference temperature: 25℃ Relative Humidity 40% ~ 60% RH.

Tab.1 Electrical Characteristics

Project		Min.	Typ.	Max.	Unit
Pressure	Accuracy			3	% Rd
	Zero point accuracy		0.3		Pa
	Zero-point repeatability		0.1		Pa
	Repeatability		0.5		% Rd
	Response time		10		ms
	Data refresh rate		1000		Hz
Temperature	Measurement range	- 40		8 5	°C
	Accuracy (-10 to 60)		2		
	Accuracy (-40 to 85)		3		
	Repeatability		0.3		
Operating current			18	20	mA
power supply voltage		3.0		5.5	V
Power-on time (time for the sensor to become ready)			25		ms
SCL frequency			200	4 00	KHz
Working pressure				1	Bar
Burst pressure		5			Bar
Compensated temperature		0		50	℃
Operating temperature		-20		65	℃
Storage temperature		-40		85	℃
Electrical interface		2.0 mm 4 -pin long needle			
Overall Material		silicon FR4 , PBT , epoxy resin , silicone, lead-free solder			

2.2 Structure Specification

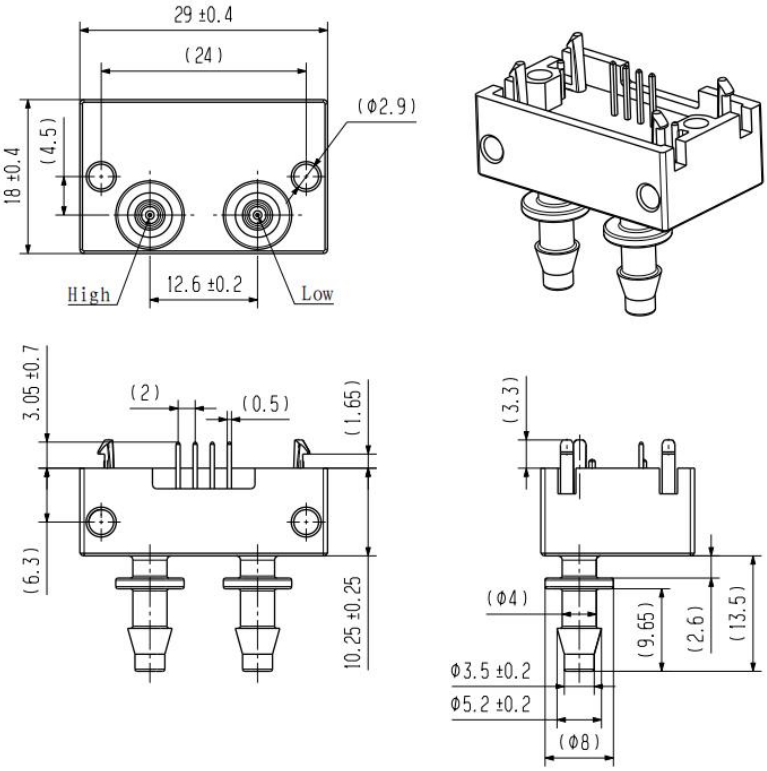

Fig.1 Product dimensions

2.3 Electrical Connections

A and B are vents, and both A and B can be used as air inlets and outlets depending on the situation. When vent A is the air inlet, the differential pressure is positive; when vent B is the air inlet, the differential pressure is negative.

Tab.2 Pin Definition

Pin number	Pin Name	Describe
1	SDA	Serial data (I2C interface)
2	GND	Grounding
3	VDD	VDD power supply
4	SCL	Serial clock (I2C interface)

3 Order Guide

GZP 6810 500Pa WX

Tab.3 Order rules

GZP	Pressure Sensor Series
6812	Product Series
D	Output type A: Analog output D: IIC output
500Pa	Pressure range:±500Pa
WX	Company int±500Paerior code

4 Application Circuit

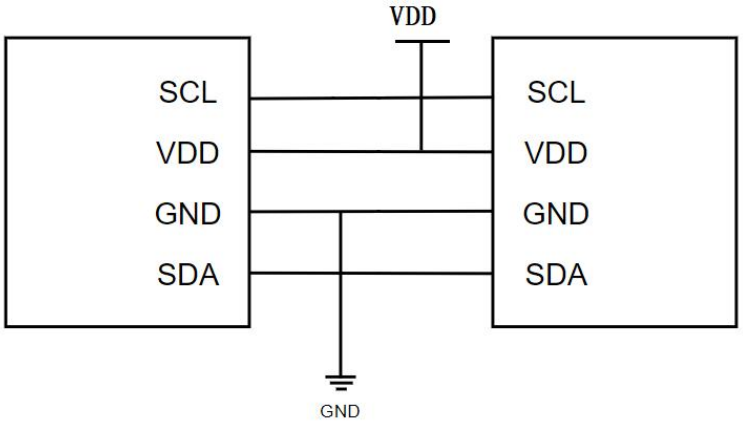

Fig.3 Application Circuit Diagram

Please confirm the electrical definitions before assembly.

This product does not have reverse connection protection; please pay attention to the power polarity during assembly.

5 I²C Communication Protocol

The I2C bus uses SCL and SDA as signal lines, which are internally pulled up to VDD by a 4.7k resistor and remain high when not communicating. It uses a standard 7-bit addressing method, where the slave address is transmitted starting with the first byte after the start signal. The GZP6810's 7-bit I2C address is 0x25.

5.1 I2C Timing Diagram

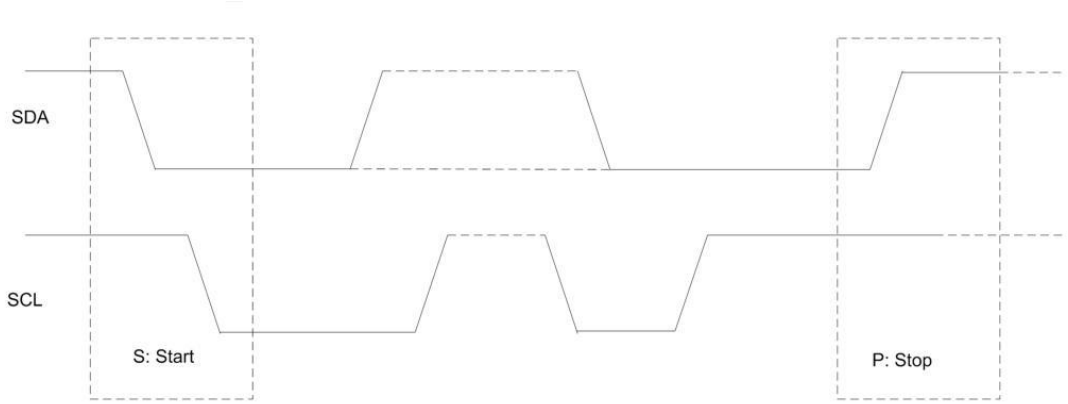

Fig.4 I2C Timing Diagram

The I2C communication protocol has specific start (S) and stop (P) conditions. When SCL is high, the falling edge of SDA is a critical condition.

Data transmission begins. The I2C master device sequentially sends the slave device's address (7 bits) and read/ write control bits . When the slave device recognizes this address...

After receiving the address , the master device generates an acknowledgment signal and pulls SDA low during the ninth cycle. Upon receiving the acknowledgment from the slave device, the master device continues to send 8 bits of register data.

The register address is used to continue sending or reading data after receiving an acknowledgment. When SCL is high, a rising edge on SDA activates the I2C flag.

Communication ends. Besides the start and end flags, the data transmitted by SDA must remain stable when SCL is high. When SCL is low...

The value transmitted by SDA can change. All data transmissions in I2C communication are based on 8-bit units, and a delay is required after every 8 bits of data transmission.

A response signal is required to keep transmission going.

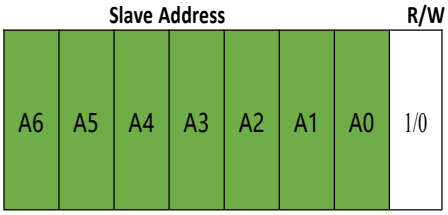

Fig.5 Standard I2C 7-bit addressing format

5.2 Master device writes data to slave device

The master device writes multiple data transmission formats to the slave device as shown in the figure below. After the master sends the write address , the slave responds to the address. The master sends multiple 8-bit data, and the slave sends back response signals in sequence. After sending all the data, the master sends a stop signal to indicate that the data transmission is over.

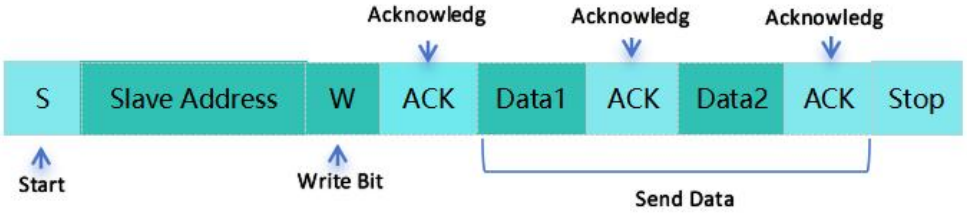

Fig.6 Data writing format

5.3 Master device read data from slave device

After the master sends the read address, the slave sends a response signal. Then, the slave sequentially sends back the corresponding data according to the SCL signal provided by the slave. Note that after sending back the data, the master needs to respond to the data. The slave decides whether to continue sending data based on the master's response signal. When the master sends NACK, it indicates that the master no longer needs to read data. Finally, the master sends a transmission end signal.

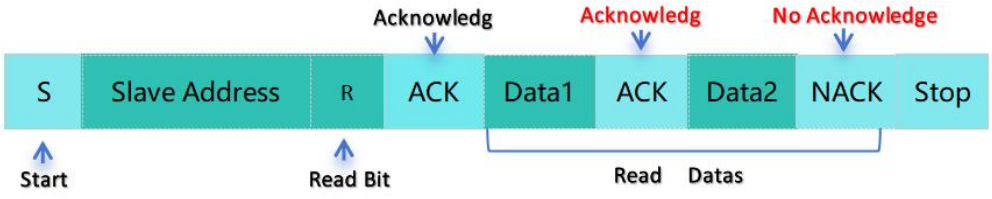

Fig.7 Read data format

6 GZP6810 Commands

Instruction Description	Instruction	Return from machine
Continuous measurement command	0x361E	Byte1: High pressure 8 bits Byte2: Low pressure 8 bits Byte3: CRC Byte4: Temperature 8 bits high Byte5: Low temperature 8 bits Byte6: CRC Byte7: Pressure scaling factor, 8 bits higher Byte8: Low 8 bits of the pressure scaling factor Byte9: CRC
Product type read command	0xE102	Byte1:0x68 Byte2:0x10 Byte3: 0xA0 (CRC)

6.1 Continuous Measurement Command

Meaning	Address	Continuous measurement command	
Sending format	Byte 1	Byte 2	Byte 3
	0x4A	0 x 36	0 x 1 E

After the master unit sends a continuous measurement command, the slave unit transmits the current pressure and temperature values back to the master unit. Subsequent read commands are only required, and the slave unit automatically remembers the previous command and continues transmitting the current real-time pressure and temperature values. The pressure and temperature data received by the master unit are signed 16-bit data, amplified by a scaling factor. After receiving the corresponding data, the master unit needs to divide it by the corresponding scaling factor to obtain the actual measured value.

Example: Host sends: 4A 36 1E

Slave response: 08 60 0D 13 66 DF 00 3C 39

0x0860 represents the raw pressure data, and 0x1366 represents the raw temperature data. Their corresponding CRC8 checksums are 0x0D and 0xDF, respectively. The specific CRC8 algorithm is detailed in Chapter 6.3.

According to the calculation formula in the table below, the measured pressure value is 0x0860/60 = 2144/60 = 35.7Pa , and the measured temperature value is 0x1366/200 = 4966/200 = 24.83℃.

Pressure difference calculation formula	Differential pressure = raw differential pressure data / 60
Temperature calculation formula	Temperature = Raw temperature data / 200

6.2 Product Type Reading Command

Host sends: 4A E1 02

Slave return: 68 10 A0

0x6810 is the product type code of GZP6810, and 0xA0 is the CRC8 check value of 0x6810.

6.3 CRC8 Check

There are generally three standards for CRC: CRC8, CRC16, and CRC32 . The GZP6810 uses CRC8. Taking the polynomial x8 + x5 + x4 + 1 (0x31) as an example, the CRC8 checksum bytes are generated by the CRC algorithm shown in the table below.

Property	**Value**
Length	8 bit
Polynomial	0x31(x8 + x5 + x4 + 1)
Initial value	0xFF
Does the input need to be reversed?	False
Does the output need to be reversed?	False
Final XOR value	0x00
example	CRC(0x1ABC) = 0xC3

The C language code for calculating the CRC code is as follows:

```
//*********************************************************
//Function name: Calc_CRC8
//Function: CRC8 calculation, initial value: 0xFF, polynomial: 0x31( x8 + x5 + x4 + 1)
// Parameters: u8 *data: the first CRC checksum; u8 Num : the length of the CRC checksum data.
//Returns: crc : the calculated crc8 value
//*********************************************************
u8Calc_CRC8 ( u8 )              * data , u 8 Len )
```

```
    {
    u8 bit, byte, crc = 0xFF;
    for( byte = 0; byte < Len; byte++)
    {
    crc^ = (data[ byte ] );
    for( bit    = 8 ;bit > 0; --bit)
    {
    if( crc & 0x80 ) crc = ( crc << 1 )^ 0x31;
    else crc =( crc << 1);
    }
    }
    return crc ;
}
```

7 Precautions for Use

(1) The only pressure medium that can be used directly is dry, non-corrosive gas. Other media, especially corrosive media , are not suitable.

When used in media containing moisture or foreign matter, it may cause malfunctions and damage, so please avoid using it in the above-mentioned environments;

(2) Regarding the operating pressure, please use it within the rated pressure range. Using it outside the range will cause damage.

(3) Reflow soldering may damage the sensor. When soldering using other methods, the pressure vents must be protected to prevent foreign objects from entering.

(4) Since the product does not have internal reverse connection protection, please ensure that the voltage connected to each pin is correct.

(5) Due to the possibility of damage caused by static electricity, please ensure that any charged objects on the table and personnel are grounded during use.

It allows static electricity in the surrounding area to discharge safely.

Example Program: Product Pressure and Temperature Reading Example Program

```
/***Port configuration section, users can configure according to their own needs***/

#define IIC_SCL_GPIO_PORT GPIOB
#define IIC_SCL_GPIO_PIN GPIO_PIN_6
#define IIC_SCL_GPIO_CLK_ENABLE()           do{ __HAL_RCC_GPIOB_CLK_ENABLE(); }while(0)
#define IIC_SDA_GPIO_PORT                    GPIOB
#define IIC_SDA_GPIO_PIN                     GPIO_PIN_7
#define IIC_SDA_GPIO_CLK_ENABLE()          do{ __HAL_RCC_GPIOB_CLK_ENABLE(); }while(0)
#define IIC_SCL(1)       GPIO_WritePin(IIC_SCL_GPIO_PORT,IIC_SCL_GPIO_PIN, GPIO_PIN_SET)
#define IIC_SCL(0)       GPIO_WritePin(IIC_SCL_GPIO_PORT,IIC_SCL_GPIO_PIN, GPIO_PIN_RESET)
#define IIC_SDA(1)     GPIO_WritePin( IIC_SDA_GPIO_PORT,IIC_SDA_GPIO_PIN, GPIO_PIN_SET)
#define IIC_SDA(0)     GPIO_WritePin(IIC_SDA_GPIO_PORT,IIC_SDA_GPIO_PIN, GPIO_PIN_RESET)
#define IIC_READ_SDA GPIO_ ReadPin ( IIC_SDA_GPIO_PORT, IIC_SDA_GPIO_PIN)

void IIC_Init (void)
{
      GPIO_InitTypeDef gpio_init_struct ;

IIC_SCL_GPIO_CLK_ENABLE ( );              SCL pin clock enable
IIC_SDA_GPIO_CLK_ENABLE ( );             / / SDA pin clock enable

      gpio_init_struct.Pin = IIC_SCL_GPIO_PIN;
      gpio_init_struct.Mode = GPIO_MODE_OUTPUT_PP; // Push-pull output
      gpio_init_struct.Pull = GPIO_PULLUP;      / / Pull up
      gpio_init_struct.Speed = GPIO_SPEED_FREQ_HIGH;      / / high speed
      HAL_GPIO_Init(IIC_SCL_GPIO_PORT, &gpio_init_struct);       // SCL

      gpio_init_struct.Pin = IIC_SDA_GPIO_PIN;
      gpio_init_struct.Mode = GPIO_MODE_OUTPUT_OD;              // Open-Drain output
      HAL_GPIO_Init(IIC_SDA_GPIO_PORT, &gpio_init_struct);       //    SDA
      IIC_Stop ( ) ;                                                    Stop all devices on the bus .
}

static void IIC_Delay (void)
```

```
{
    delay_us ( 3) ;
}

void IIC_Start (void)
{
IIC_ SDA( 1);
IIC_SCL ( 1);
    IIC_Delay ( ) ;
IIC_ SDA( 0);          / / START signal : When SCL is high , SDA changes from high to low , indicating the start signal.
    IIC_Delay ( ) ;
IIC_SCL ( 0);
    IIC_Delay ( ) ;
}

void IIC_Stop (void)
{
IIC_ SDA( 0);          / / STOP signal : When SCL is high , SDA changes from low to high , indicating a stop signal.
    IIC_Delay ( ) ;
IIC_SCL ( 1);
    IIC_Delay ( ) ;
IIC_ SDA( 1);
    IIC_Delay ( ) ;
}

uint8_t IIC_Wait_Ack ( void )
{
    uint8_t Waitime = 0;
    uint8_t Rack = 0;

    IIC_SDA(1); // The host releases the SDA line (at this time, external devices can pull the SDA line low).
    IIC_Delay ( ) ;
```

```
   IIC_Delay ( ) ;        //After the 8th falling edge, you need to wait ~9us, otherwise there is a possibility
of data loss.
IIC_SCL ( 1);      When SCL =1, the slave device can return ACK .
       IIC_Delay ( ) ;
   IIC_Delay ( ) ;
       while (IIC_READ_SDA) // wait for response
{
              Waitime ++;
              if ( Waitime > 250)
{
                     IIC_Stop ( ) ;
Rack = 1;
                     break ;
}
}

IIC_SCL ( 0); // SCL =0, end ACK check
       IIC_Delay ( ) ;
       return Rack;
}

void IIC_Ack (void)
{
IIC_SDA ( 0); // When SCL 0 -> 1, SDA = 0, indicating a response.
       IIC_Delay ( ) ;
IIC_SCL ( 1); // Generate a clock
       IIC_Delay ( ) ;
IIC_SCL ( 0);
       IIC_Delay ( ) ;
IIC_SDA ( 1); // Host releases SDA line
       IIC_Delay ( ) ;
}

void IIC_Nack (void)
{
```

```
IIC_SDA ( 1); // When SCL 0 -> 1, SDA = 1, indicating no response.
    IIC_Delay ( ) ;
IIC_SCL ( 1); // Generate a clock
    IIC_Delay ( ) ;
IIC_SCL ( 0);
    IIC_Delay ( ) ;
}

void IIC_Send_Byte (uint8_t Data)
{
    uint8_t t;
    for (t = 0; t < 8; t++)
{
IIC_SDA ( (Data & 0x80) >> 7); // Send the most significant bit first
IIC_SCL ( 1);
IIC_SCL ( 0);
Data <<= 1; // Left shift by 1 bit for the next transmission
}
IIC_SDA ( 1); // Transmission complete , host releases SDA line
}

uint8_t IIC_Read_ Byte ( uint8_t Ack )
{
    uint8_t i , ReceData = 0;
    for ( i = 0; i < 8; i ++ ) // Receive 1 byte of data
{
        ReceData <<= 1 ; // The most significant bit is output first, so the first received data bits need
to be shifted left.
IIC_SCL ( 1);
        if (IIC_READ_SDA)
{
            ReceData ++;
}
IIC_SCL ( 0);
        delay_us ( 2) ;    // Minimum delay of 2μs, otherwise data read will fail.
```

```
}
    if (! Ack )
{
        IIC_Nack ( ) ; // Send nACK
}
    else
{
        IIC_Ack ( ) ; // Send ACK
}
    return ReceData ;
}

uint8_t GZP6810_Trig_ ReadData( uint16_t Cmd,uint8_t *p)
{
    uint8_t i=0,ACK=0,Ture=1,False=0;
    IIC_Start();
    IIC_Send_Byte(0x4a);
    ACK = IIC_Wait_Ack();
    IIC_Send_Byte(Cmd>>8);
    ACK = IIC_Wait_Ack();
    IIC_Send_Byte(Cmd);
    ACK = IIC_Wait_Ack();
    IIC_Stop();
    delay_ms(1);
    IIC_Start();
    IIC_Send_Byte (0x4B);
    ACK = IIC_Wait_Ack();
    for(i=0;i<9;i++)
    {
        if ( i == 8 ) p[i]=IIC_Read_Byte(False);
        else p[i]=IIC_Read_Byte(Ture);
    }
    IIC_Stop();

  return ACK;
```

```
}
uint8_t GZP6810_ReadData(uint8_t *p)
{
        uint8_t i=0,ACK=0,Ture=1,False=0;
        IIC_Start();
        IIC_Send_Byte (0x4b);
        ACK = IIC_Wait_Ack();
        for(i=0;i<9;i++)
        {
                if ( i == 8 ) p[i]=IIC_Read_Byte(False);
                else p[i]=IIC_Read_Byte(Ture);
        }
        IIC_Stop();
        return ACK;
}
void    main(void)
{
    float Pressure,Temperature;                         //Pressure, Temperature
    uint8_t p[9],IIC_Ack= 0;
       HAL_Init ( ); // Initialize the HAL library
      sys_stm32_clock_init ( RCC_PLL_MUL9); // Set the clock to 72MHz
       delay_init ( 72 ); // Delay initialization
       usart_ init ( 460800); // Initialize the serial port to 115200
    IIC_Init ( ) ;
    delay_ms ( 5) ;
    IIC_Ack = GZP6810_Trig_ ReadData( 0x361E,p);
    delay_ms ( 40) ;
    while( 1)
       {
              IIC_Ack = GZP6810_ ReadData( p);
               Pressure =( float)(((int16_t)(p[0]*256)+p[1])/60 .0 ); // Pressure proportionality coefficient is
60
              Temperature = ( float)((int16_t)((p[3]*256)+p[4])/200.0); //Temperature scaling factor is 200
              if ( IIC_Ack ) printf("i2c ack error\n");
              else
```

```
                {
                    printf("Press is %f\n",Pressure);
              printf("Temp is %f\n",Temperature);
                }
            delay_ms(5);
        }
}
```

Safety Precautions

This product is made of semiconductor components for general electronic equipment (communication equipment, measuring equipment, working machinery, etc.). Products using these semiconductor components may malfunction and fail due to external interference and surges, so please confirm the performance and quality under actual use. To be on the safe side, please perform safety design on the device (setting of protection circuits such as fuses and circuit breakers, multiple devices, etc.) so that life, body, property, etc. will not be harmed in the event of a malfunction. To prevent injuries and accidents, please be sure to comply with the following matters:

·The driving current and voltage should be used below the rated values.

Please wire according to the electrical definition . In particular, reverse connection of the power supply may cause accidents due to circuit damage such as heat, smoke, and fire, so please be careful.

·Be careful when fixing the product and connecting the pressure inlet .

Warranty and Disclaimer

The information in this sheet has been carefully reviewed and is believed to be accurate; however, no responsibility is assumed for inaccuracies. Furthermore, this information does not convey to the purchaser of such devices any license under the patent rights to the manufacturer. Sencoch Technology reserves the right to make changes without further notice to any product herein. Sencoch Technology makes no warranty, representation or guarantee regarding the suitability of its product for any particular purpose, nor does Sencoch Technology assume any liability arising out of the application or use of any product or circuit and specifically disclaims any and all liability, including without limitation consequential or incidental damages. Typical parameters can and do vary in different applications. All operating parameters must be validated for each customer application by customer’s technical experts. Sencoch Technology does not convey any license under its patent rights nor the rights of others.