GZP6877D-C

型压力传感器

数字输出

受控文件

产品规格书

版本号： V1.0

文件发行日期：2025.01.13

目录

1. 产品描述 4
1.1 产品特点 4
1.2 应用领域 4
2. 功能描述 5
2.1 引脚定义说明 5
2.2 框图 6
2.3 精度 6
3.技术指标 8
3.1 最大额定参数 8
3.2 性能指标 8
3.3 电气特性 9
4.应用电路 9
5.I^2C 通讯协议 10
6.寄存器描述 12
7.工作模式说明 15
8.外形结构 16
9.选型指南 17
10.常用量程型号 17
11.使用注意事项 18
11.1 焊接 18
11.2 清洗要求 19
11.3 存储与运输 19
11.4 其他使用注意事项 19
12.包装信息 20
安全注意事项 21
IIC Example Code（附件：IIC 代码案例） 22

文件修订历史

修订	描述	日期
V1.0	初始版本	2025.01.13

公司保留在不另行通知的情况下对其所包含的规格进行更改的权利。
产品规格书版权及产品最终解释权归芯感智所有。

1. 产品描述

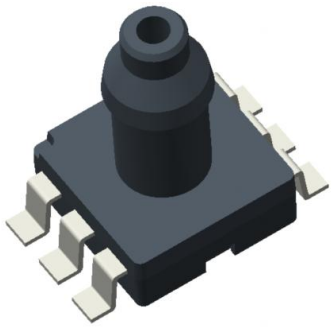
GZP6877D-C 型压力传感器采用 SOP6 封装形式，倒钩管的设计可以保证安装的密封性。内有封装的压力传感器与信号调理芯片，对传感器的偏移、灵敏度、温漂和非线性进行数字补偿。采用 24 位 ADC，并且调理芯片内置温度传感器，可以输出高精度的压力值和温度值。同时提供 IIC 通讯协议接口，抗干扰能力强。所有测量数据都经过充分校准和温度补偿。

GZP6877D-C 型压力传感器响应快、精度高，具有良好的线性以及长期稳定性。

GZP6877D-C 型压力传感器集成度高、可靠性好，易安装，便于 SMT，广泛用于医疗电子、汽车电子、运动健身器材等领域。

1.1 产品特点

- 测量范围-100kPa...0～0.5kPa...1000kPa
- 表压型
- 气嘴带防脱结构
- 电源电压: 2.7V-5.5V
- 适用于无腐蚀性的气体
- IIC 输出



1.2.应用领域

- 呼吸机、肺活量计、负压伤口治疗、血压监护、睡眠窒息治疗等医疗领域
- 空气流量检测、供暖通风与空气调节、气动设备、压力开关等工业领域
- 生物科学、小家电、消费电子、运动健身器材、消防器材、物联网等领域
- 气体流量仪表、气体排放、变风量调节等领域

2. 功能描述

本产品用先进的微机电原理制作，核心技术为基于硅压阻效应的 MEMS 压力传感器芯片和高性能的信号调理 AISC 芯片，硅微压阻式 MEMS 压力传感器 芯片通过四个应变敏感电阻构成的惠斯通电桥，输出信号被 ASIC 芯片放大、温度补偿和线性化，传递函数的线性化和温度补偿由 ASIC 中的数字处理电路实现，通过多项式补偿算法和多个温度下的多点压力标定技术，实现了全工作温度范围内的高精度压力测量。

2.1 引脚定义说明

压力传感器引脚配置如图 1。

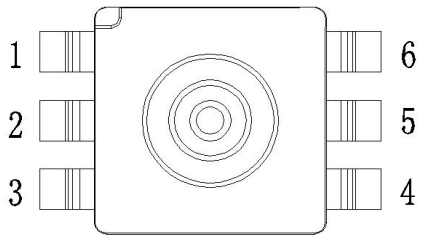


图 1. 引脚示意图

表 1. 引脚对应关系

序号	描述	备注
1	NC	悬空引脚
2	VDD	电源输入正极
3	NC	悬空引脚
4	SDA	数据输出脚
5	SCL	信号输出脚
6	GND	电源输入负极

表 1

注意：

1. 装配前请确认好电气定义
2. NC 脚不要有任何的电气连接，否则可能会造成产品功能失效
3. 焊装过程中做好防静电保护
4. 过载电压(6.5Vdc)可能烧毁电路芯片
5. 请在 VDD 和 GND 之间加上 0.1uf 电容
6. 本产品无反接保护，装配时请注意电源极性

2.2 框图

传感器功能框图如图 2 所示

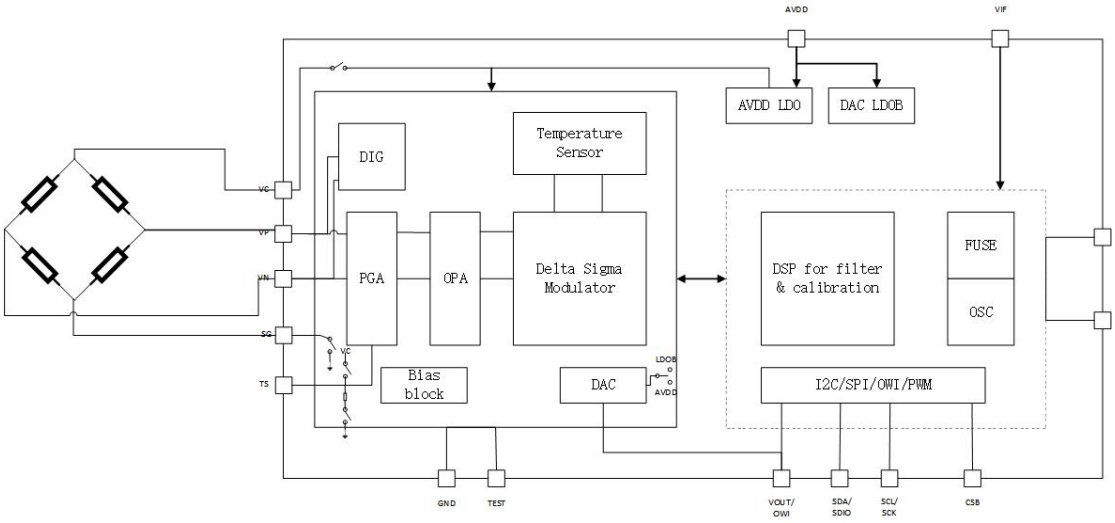


图 2 传感器功能框图

2.3 精度

GZP6877D-C 型压力传感器的精度受由其线性、重复性、迟滞的误差组成。用传递函数计算的值是传感器的规定值，也是理论值。传感器的误差等于传感器在规定输入压力下的实际输出值与规定的输出值的差值。

2.3.1 综合误差

综合误差在产品的精度基础上，还包括更多的误差源：

压力漂移:指定压力范围内，零点和满量程的实际输出电压和规定输出电压的输出偏差。

温度效应:在温度范围内的，零点和满量程在不同温度下的输出偏差

综合误差采用误差带表示，该误差带由三段线段组成，数据如图 3 和表 2 所示。

温度(℃)	**误差(Fs)**
-20~100	±3.0%
0~60	±1.5%

* 压力量程不同，综合误差不同，请咨询客服获取更多细节

表 2 综合误差

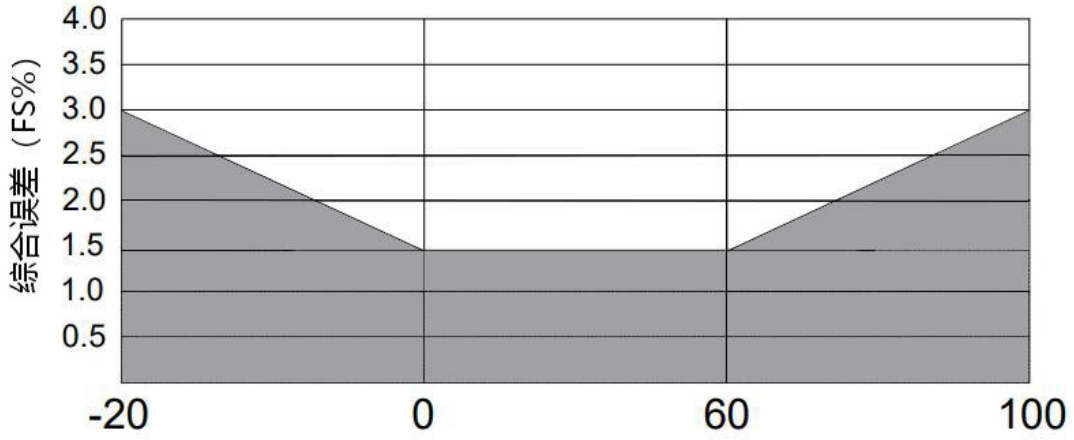


图 3 综合误差与温度关系

3.技术指标

传感器以下指标均在供电电源(3.3±0.25)V DC 和温度 25℃下测得

3.1 最大额定参数

传感器最大额定参数如表 3 所示

参数	最小值	典型值	最大值	单位	备注
供电电压	-0.3		6.5	V	VDD/VDDIO
数字引脚电压	-0.3		VDDIO+0.3	V	25℃
ESD 防护		±2		kV	HBM
储存温度	-40		125	℃	
工作温度	-30		105	℃	

表 3

3.2 性能指标

传感器性能指标如表 4 所示

项目	数值	单位
精度*	±1	%Span
响应时间	5ms@OSR P=1024X	ms
SDA/SCL 上拉电阻	4.7	K ohm
零点温度漂移	±0.03	%FS/℃
满程温度漂移	±0.03	%FS/℃
过载压力	4×（量程≤60kPa）	Rated
	2.5×（60kPa<量程≤200kPa ）	
	1.5×（量程>200kPa）	
破坏压力	5×（量程≤60kPa）	
	3×（60kPa<量程≤200kPa ）	
	2×（量程>200kPa）	
补偿温度	0～60（可定制）	℃

* 量程不同，精度不同，请咨询客服获取更多细节。

表 4

3.3 电气特性

传感器电气特性如表 5 所示

参数	最小值	典型值	最大值	单位	备注
供电电压	2.7		5.5	V	
待机电流		1		uA	
LDO 输出		2.8		V	SEL LDAC=1
		4.8		V	SEL LDAC=0
分辨率		21		Bits	
输出数据分辨率	24			Bits	LSB=(1/2^23)*VEXT
内置温度传感器			±1	℃	@25℃
			±1	℃	0 to 85 ℃
时钟脉冲频率			400	KHz	I2C 通讯

表 5

4.应用电路

芯片推荐应用电路见图 4

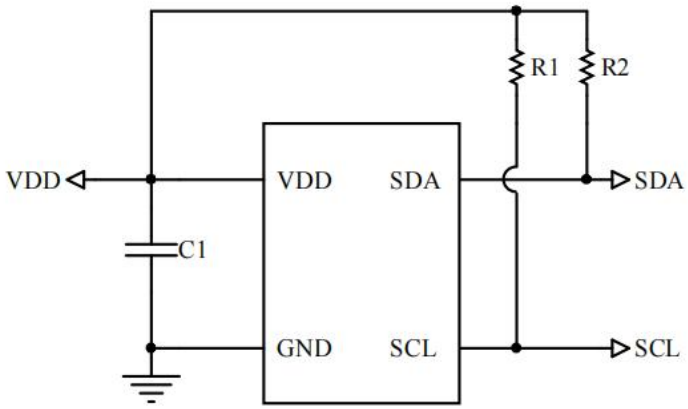


图 4

注:C1 推荐值为 100nF;R1,R2 推荐值为 4.7k

5.I²C 通讯协议

I²C 总线使用 SCL 和 SDA 作为信号线，这两根线都通过上拉电阻（典型值 4.7K）连接到 VDD，不通信时都保持为高电平。I²C 设备地址为 0x58。

传感器 I²C 通讯参数特性如表 6 所示

标识	参数	条件	最小值	最大值	单位
fscl	时钟频率			400	KHz
tLOW	时钟低脉冲维持时间		1.3		us
tHIGH	时钟高脉冲维持时间		0.6		us
tSUDAT	SDA 建立时间		0.1		us
tHDDAT	SDA 保持时间		0.0		us
tSUSTA	每次开始时的建立时间		0.6		us
tHDSTA	开始条件保持时间		0.6		us
tSUSTO	停止条件建立时间		0.6		us
tBUF	两次通讯间隔时间		1.3		us

表 6

- I²C 时序图

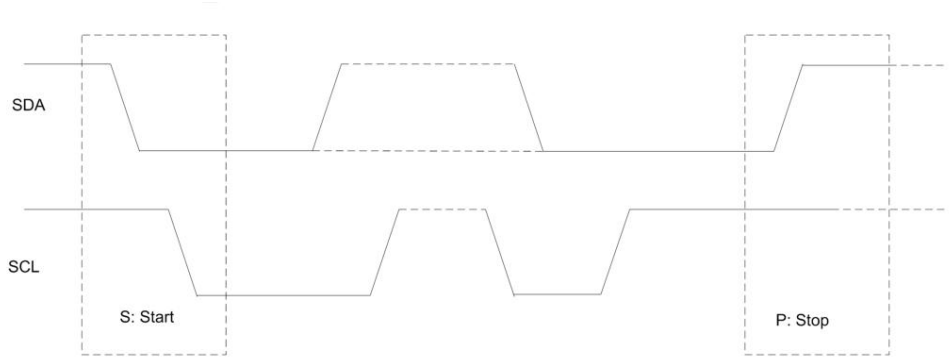


图 5.IIC 时序图

I²C 通讯协议有着特殊的开始(S)和终止(P)条件。当 SCL 处于高电平同时，SDA 的下降沿标志数据传输开始。I²C 主设备依次发送从设备的地址（7 位）和读/写控制位。当从设备识别到这个地址后，产生一个应答信号并在第九个周期将 SDA 拉低。得到从设备应答后，主设备继续发

送 8 位寄存器地址，得到应答后继续发送或读取数据。SCL 处于高电平，SDA 发生一个上升沿动作标志 I^2C 通信结束。除了开始和结束标志之外，当 SCL 为高时 SDA 传输的数据必须保持稳定。当 SCL 为低时 SDA 传输的值可以改变。I^2C 通信中的所有数据传输以 8 位为基本单位，每 8 位数据传输之后需要一位应答信号以保持继续传输。

- I^2C 协议

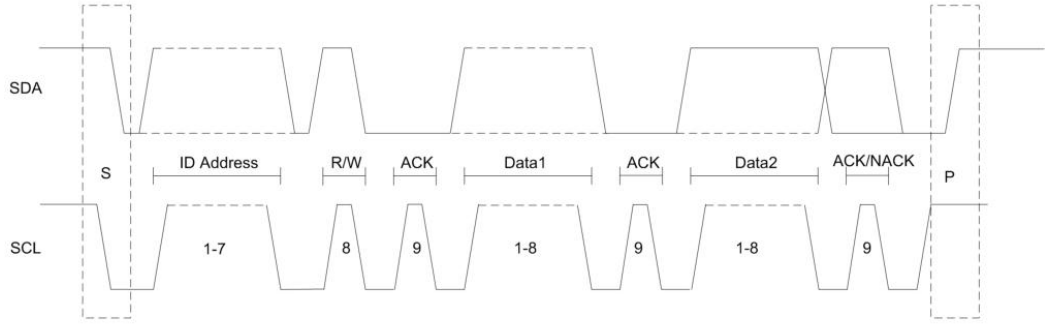


图 6. I^2C 协议

6.寄存器描述

表 7 寄存器描述

地址	描述	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	默认值
0x00	ID	R	ID<7:0>								0x58
0x01	Chip_Control	R/W	reserved<7:6>		data_ready	reserved	data_out	measurement_ctrl	Active<1:0>		0x05
0x02	CFG_OSR	R/W	OSR_T<7:5>			OSR_P<4:2>			MODE[1:0]		OTP
0x03	CFG_MEAS	R/W	reserved<7:6>		T_SB[5:3]			PT_R[2:0]			OTP
0x04	P_data	R	Data out<23:16>								0x00
0x05	P_data	R	Data out<15:8>								0x00
0x06	P_data	R	Data out<7:0>								0x00
0x07	T_data	R	Temp out<15:8>								0x00
0x08	T_data	R	Temp out<7:0>								0x00
0x24	CFG_OPER	R/W	reserved<7:1>						DAC_EN		

表 7

Reg0x00

I^2C 设备地址，默认地址为 0x58。

Reg0x01

芯片控制寄存器

Active<1:0>:00,芯片掉电；01，芯片启动；

measurement_ctrl：0，压力测量；1，温度测量；

data_out：0，输出校准数据；1，输出原始数据；

data_ready：0，数据转换未完成；1，数据转换完成。

Reg0x02

MODE[1:0]: 00: Idle mode 睡眠模式，01: Normal mode 正常模式，10: One shot mode 单次采集模式，11：Normal mode, cyclic measurement 正常模式循环测量

OSR_P[4:2]（压力过采样）： 000: over sampling x 256

001: over sampling x 512

010: over sampling x 1024

011: over sampling x 2048

100: over sampling x 4096

101: over sampling x 8192

110: over sampling x 16384

111: over sampling x 32768

OSR_T[7:5]（温度过采样）： 000: over sampling x 256

001: over sampling x 512

010: over sampling x 1024

011: over sampling x 2048

100: over sampling x 4096

101: over sampling x 8192

110: over sampling x 16384

111: over sampling x 32768

Reg0x03

PT_R[2:0]：000: 64/1，001: 32/1，010: 16/1，011: 8/1，100: 4/1，101: 1/1，Others: 128/1 （正常模式下，压力/温度测量比）

T_SB[5:3]：000: 0ms，001: 62.5ms，010: 125ms，011: 250ms，100: 500ms，101: 750ms，110: 1000ms，111: 2000ms（正常模式下的待机时间设置）

Reg0x04-Reg0x06

压力数据寄存器

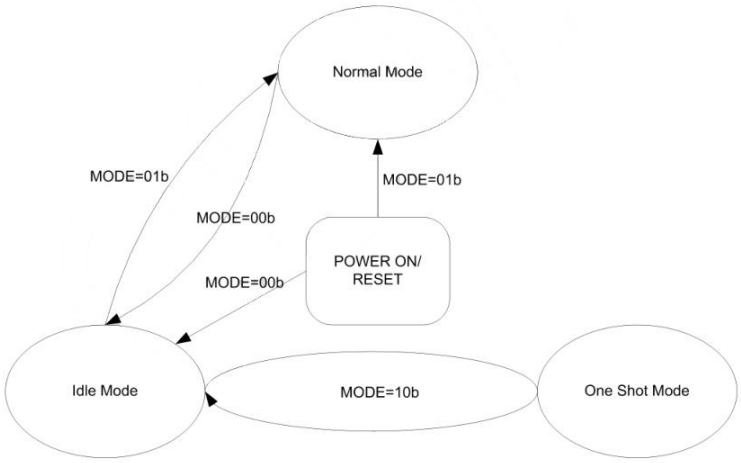
Reg0x07-Reg0x08

温度数据寄存器

Reg0x24

DAC_EN: 0:禁用 DAC，1:启用 DAC

7.工作模式说明



Normal Mode：通过向 MODE 寄存器(0x02[1:0])写入 01b，可以开启正常模式；压力传感器和温度传感器的信号在一个预定的频率下周期性的输出测量数据，产品会在一定的时间后进入休眠状态；

One Shot Mode：通过向 MODE 寄存器(0x02[1:0])写入 10b，可以开启单触发模式；用户可以通过清除或置位 measurement_ctrl(0x01[2])来指定是测量温度还是压力信号。在单触发测量完成后,将进入空闲模式以等待下一个指令。

Idle Mode:空闲模式

8.外形结构（单位：mm）

传感器外型尺寸参照图 7。(未注公差按±0.1mm 执行)

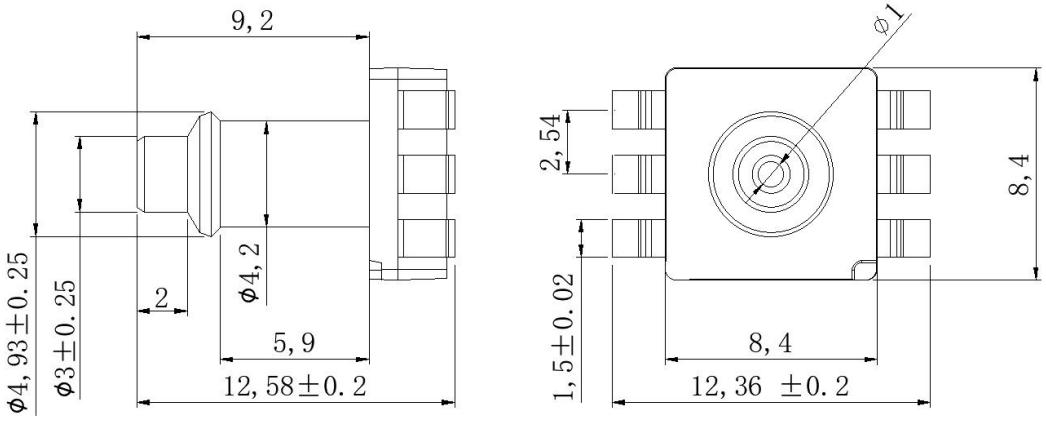


图 7

建议焊盘尺寸图 8

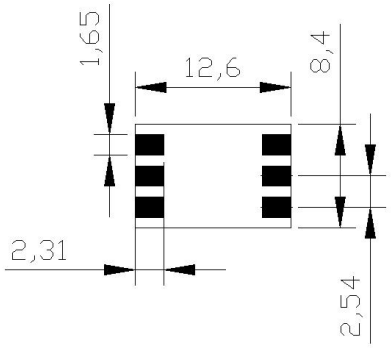


图 8

9.选型指南

GZP 6877 D-C - 001MPP F01

GZP	芯感智
6877	产品大类
D	输出类型 A：模拟输出 D：IIC 输出
C	通讯协议方式
001MPP	压力量程: 001 表示所测压力数值 压力单位: KP: KPa MP:MPa HP:HPa PS: PSI BA:Bar 压力类型: P: 正压(如 0~1) W：负压到正压(如-1~1) 故 001MPP 表示 0 到 1MPA 的测量压力
F01 (附属项)	包装方式 F01：料管 B01：编带
XXX（如有）	定制项

表 9 选型指南

10.常用量程型号

压力量程	**型号**
0 ~ 200kPa	GZP6877D-C201KPP F01
0 ~ 700kPa	GZP6877D-C701KPP F01
0 ~ 1000kPa	GZP6877D-C001MPP F01(B01 可选)
-2.5 ~ 0kPa	GZP6877D-C2.5KPN F01
更多定制量程及参数,请咨询我司客服或代理商	

表 10. 常用量程表

11.使用注意事项

11.1 焊接

由于本产品为热容量较小的小型构造，因此请尽量减少来自外部的热量的影响。否则可能会因热变形而造成破损，引起特性变动。请使用非腐蚀性的松香型助焊剂。另外，由于产品暴露在外，因此请注意不要使助焊剂侵入内部。

1） 手焊接

· 请使用头部温度在 260 ~ 300 ℃（30 W）的电烙铁 在 5 秒以内实施作业。

· 在端子上施加负载进行焊接的情况下，由于输出可能会发生变化，因此请注意。

· 请保持电烙铁头洁净。

2） 回流焊接（SMD 端子型）

推荐的回流炉温度设置条件如下所示

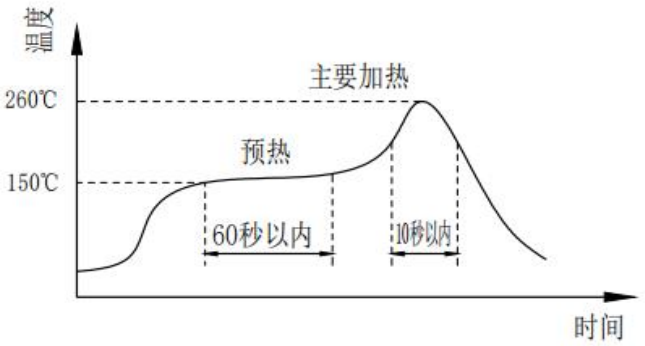


图 9. 回炉温度设置条件

3） 印刷板的翘度相对于整个传感器应保持在 0.05mm 以下，请对此进行管理。

4） 安装传感器后，对基板进行切割弯折时，请注意不要使焊接部产生应力。

5） 由于传感器的端子为外露构造，因此金属片等触摸端子后，会引发输出异常。请注意不要用金属片或者手等触摸。

6） 焊接后，为了防止基板的绝缘恶化而实施涂层时，请注意不要使传感器上面附着药剂。

11.2 清洗要求

1） 由于产品为开放型，因此请注意不要使清洗液侵入内部。

2） 使用超声波进行清洗时，可能会使产品发生故障，因此请避免使用超声波进行清洗。

11.3 存储与运输

1） 本产品为非防滴构造，因此请勿在可能溅到水等的场所中使用。

2） 请勿在产生凝露的环境中使用。另外，附着在传感器芯片上的水分冻结后，可能会造成传感器输出的变动或者破坏。

3） 压力传感器的芯片在构造上接触到光后，输出会发生变动。尤其是通过透明套等施加压力时，请避免使光接触到传感器的芯片。

4）正常包装的压力传感器可通过普通输送工具运输。请注意： 产品在运输过程中防止潮湿、冲击、晒伤和压力。

11.4 其他使用注意事项

1） 安装方法错误时，会造成事故，因此请注意。

2） 请避免采用超声波等施加高频振动的使用方法。

3） 能够直接使用的压力媒介仅为空气和无腐蚀性气体，除此以外的媒介，尤其是在腐蚀性介质和含有异物的媒介中使用时，会造成故障和破损，因此请避免在上述环境中使用。

4） 压力导入口的内部配置有压力传感器芯片。从压力导入口插入针等异物后，会造成芯片破损和导入口堵塞，因此请绝对避免上述操作。另外，使用时请避免堵塞大气导入口。

5） 关于使用压力，请在额定压力的范围内使用。在范围外使用时，会造成破损。

6） 由于可能因静电而造成破坏，因此使用时请注意以下事项。

请将桌子上的带电物，作业人员接地，以使周围的静电安全放电。

12.包装信息

料管信息（单位为毫米）

每管数量:58 PCS

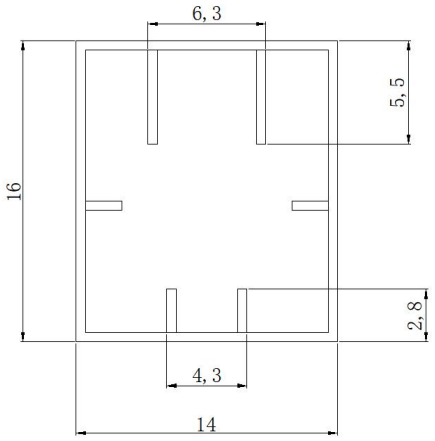
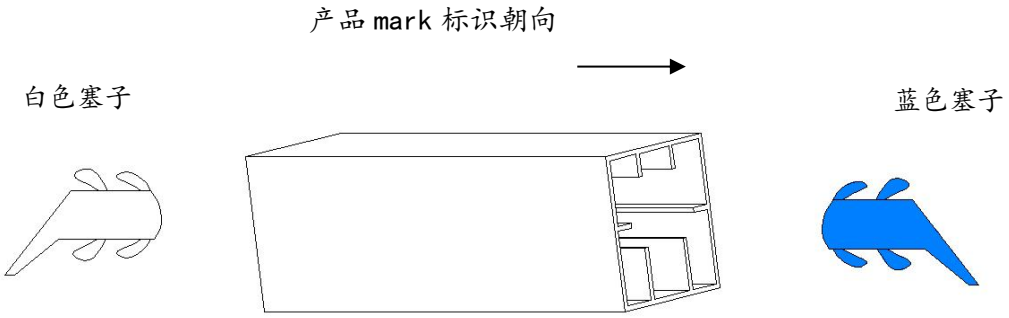
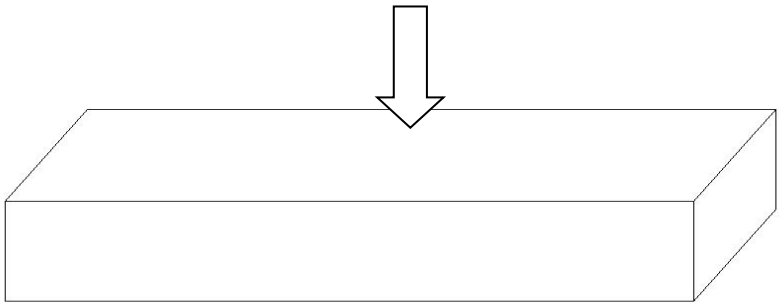


图 10.料管截面示意图



(520mm x 14mm x 16mm，58PCS)



(纸盒：530mm x 145mm x 53mm，1856PCS)

图 11.包装

安全注意事项

本产品是使用一般电子设备用（通信设备，测量设备，工作机械等）的半导体部品而制成的。使用这些半导体部品的产品，可能会因外来干扰和浪涌而发生误动作和故障，因此请在实际使用状态下确认性能及品质。为以防万一，请在装置上进行安全设计（保险丝，断路器等保护电路的设置，装置多重化等），一旦发生误动作也不会侵害生命，身体，财产等。为防止受伤及事故的发生，请务必遵守以下事项：

·驱动电流和电压应在额定值以下使用。

·请按照电气定义进行接线。特别是对电源进行逆连接后，会因发热，冒烟，着火等电路损伤引发事故，因此敬请注意。

·对产品进行固定和对压力导入口进行连接时请慎重。

IIC Example Code（附件：IIC 代码案例）

```
#include "stm32f10x.h"

// 定义 SCL 和 SDA 引脚
#define SCL_PIN GPIO_Pin_6
#define SDA_PIN GPIO_Pin_7
#define I2C_GPIO_PORT GPIOB

// 假设量程为 1000，零点为 0，可根据实际情况修改
#define RANGE 1000.0
#define ZERO_POINT 0.0

// 延时函数
void delay_us(uint32_t us) {
    uint32_t count = us * 7;
    while (count--);
}

// I2C 初始化函数
void I2C_Init(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    // 使能 GPIOB 时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    // 配置 SCL 和 SDA 引脚为开漏输出
    GPIO_InitStructure.GPIO_Pin = SCL_PIN | SDA_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_OD;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(I2C_GPIO_PORT, &GPIO_InitStructure);
    // 初始状态，SCL 和 SDA 为高电平
    GPIO_SetBits(I2C_GPIO_PORT, SCL_PIN | SDA_PIN);
}

// 产生 I2C 起始条件
void I2C_Start(void) {
    GPIO_SetBits(I2C_GPIO_PORT, SDA_PIN);
    delay_us(5);
    GPIO_SetBits(I2C_GPIO_PORT, SCL_PIN);
    delay_us(5);
    GPIO_ResetBits(I2C_GPIO_PORT, SDA_PIN);
    delay_us(5);
    GPIO_ResetBits(I2C_GPIO_PORT, SCL_PIN);
    delay_us(5);
}

// 产生 I2C 停止条件
void I2C_Stop(void) {
    GPIO_ResetBits(I2C_GPIO_PORT, SDA_PIN);
    delay_us(5);
    GPIO_SetBits(I2C_GPIO_PORT, SCL_PIN);
    delay_us(5);
    GPIO_SetBits(I2C_GPIO_PORT, SDA_PIN);
    delay_us(5);
```

```
}

// I2C 发送一个字节
void I2C_SendByte(uint8_t byte) {
    for (uint8_t i = 0; i < 8; i++) {
        if (byte & 0x80) {
            GPIO_SetBits(I2C_GPIO_PORT, SDA_PIN);
        } else {
            GPIO_ResetBits(I2C_GPIO_PORT, SDA_PIN);
        }
        byte <<= 1;
        delay_us(5);
        GPIO_SetBits(I2C_GPIO_PORT, SCL_PIN);
        delay_us(5);
        GPIO_ResetBits(I2C_GPIO_PORT, SCL_PIN);
        delay_us(5);
    }
}

// I2C 读取一个字节，并发送 ACK 或 NACK
uint8_t I2C_ReadByte(uint8_t ack) {
    uint8_t byte = 0;
    GPIO_SetBits(I2C_GPIO_PORT, SDA_PIN);
    for (uint8_t i = 0; i < 8; i++) {
        byte <<= 1;
        GPIO_SetBits(I2C_GPIO_PORT, SCL_PIN);
        delay_us(5);
        if (GPIO_ReadInputDataBit(I2C_GPIO_PORT, SDA_PIN)) {
            byte |= 0x01;
        }
        GPIO_ResetBits(I2C_GPIO_PORT, SCL_PIN);
        delay_us(5);
    }
    if (ack) {
        GPIO_ResetBits(I2C_GPIO_PORT, SDA_PIN); // 发送 ACK
    } else {
        GPIO_SetBits(I2C_GPIO_PORT, SDA_PIN); // 发送 NACK
    }
    delay_us(5);
    GPIO_SetBits(I2C_GPIO_PORT, SCL_PIN);
    delay_us(5);
    GPIO_ResetBits(I2C_GPIO_PORT, SCL_PIN);
    delay_us(5);
    return byte;
}

// I2C 等待从设备应答
uint8_t I2C_WaitAck(void) {
    uint8_t ack = 0;
    GPIO_SetBits(I2C_GPIO_PORT, SDA_PIN);
    GPIO_SetBits(I2C_GPIO_PORT, SCL_PIN);
    delay_us(5);
    if (GPIO_ReadInputDataBit(I2C_GPIO_PORT, SDA_PIN)) {
```

```
            ack = 1;
        } else {
            ack = 0;
        }
        GPIO_ResetBits(I2C_GPIO_PORT, SCL_PIN);
        delay_us(5);
        return ack;
}
// 从指定从机地址和寄存器地址读取多个字节
void I2C_ReadBytes(uint8_t slave_addr, uint8_t reg_addr, uint8_t *data, uint8_t len) {
        I2C_Start();
        // 发送从机地址，写操作
        I2C_SendByte(slave_addr << 1);
        I2C_WaitAck();
        // 发送寄存器地址
        I2C_SendByte(reg_addr);
        I2C_WaitAck();
        I2C_Start();
        // 发送从机地址，读操作
        I2C_SendByte((slave_addr << 1) | 0x01);
        I2C_WaitAck();
        for (uint8_t i = 0; i < len; i++) {
            if (i == len - 1) {
                // 最后一个字节，发送 NACK
                data[i] = I2C_ReadByte(0);
            } else {
                // 发送 ACK
                data[i] = I2C_ReadByte(1);
            }
        }
        I2C_Stop();
}
int main(void) {
        uint8_t received_data[5];
        uint8_t temp_coeff_data[2];
        uint32_t pressure_data;
        uint16_t temperature_data;
        float shiftN;
        int EOFFout;
        float actual_pressure,actual_temperature;
        I2C_Init();
        // 从机地址 0x58，寄存器地址 0x04，读取 5 个字节
        I2C_ReadBytes(0x58, 0x04, received_data, 5);
        // 组合压力数据
        pressure_data = ((uint32_t)received_data[0] << 16) | ((uint32_t)received_data[1] << 8) |
(uint32_t)received_data[2];
        // 组合温度数据
        temperature_data = ((uint16_t)received_data[3] << 8) | (uint16_t)received_data[4];
        // 从 0x20 寄存器读取两个字节作为温度系数
        I2C_ReadBytes(0x58, 0x20, temp_coeff_data, 2);
        // 计算 shiftN
        shiftN = (float)temp_coeff_data[1] / 10.0;
```

```
    // 根据温度系数[0]设置 EOFFout
    switch (temp_coeff_data[0]) {
        case 0x0C:
            EOFFout = 4096;
            break;
        case 0x8C:
            EOFFout = -4096;
            break;
        case 0x0D:
            EOFFout = 8192;
            break;
        case 0x8D:
            EOFFout = -8192;
            break;
        case 0x0E:
            EOFFout = 16384;
            break;
        case 0x8E:
            EOFFout = -16384;
            break;
        default:
            // 可以添加默认处理，这里暂时设置为 0
            EOFFout = 0;
            break;
    }

    // 处理压力数据
    if (pressure_data > 8388608) {
        pressure_data -= 16777216;
    }
    // 处理温度数据
    if (temperature_data > 32768) {
        temperature_data -= 65536;
    }
    // 计算实际压力
    actual_pressure = ((float)pressure_data / (1 << 21)) * RANGE + ZERO_POINT;
    // 计算实际温度
    actual_temperature = ((float)(temperature_data - EOFFout) / (1 << (int)shiftN)) + 25;
    while (1) {
        // 可在此处添加处理接收到的压力、温度和温度系数数据的代码
        // 例如打印数据
        // 注意：使用 printf 需要先配置串口
        // printf("Pressure: %f, Temperature: %f,\n",actual_pressure,actual_temperature);
    }
}
```