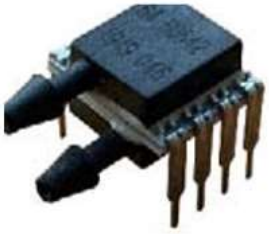
LW50 系列 压力传感器

硅陶瓷系列
数字模拟放大输出



应用领域

- 呼吸机、麻醉机
- 雾化器
- HVAC 滤清器堵塞检测
- 肺活量计
- 风道静压
- HVAC（暖通空调）变送器
- 医院室内气压控制
- VAV 调节系统

产品概述

LW50 高精度硅玻璃纤维系列为压阻硅压力传感器，可提供指定满量程压力范围和温度范围读取压力的放大输出或 24 位数字输出。 LW50 系列通过使用板载专用集成电路 (ASIC) 针对传感器偏移、灵敏度、温度效应和非线性进行了充分校准和温度补偿。经校准的模拟压力输出值会在 1 kHz 左右更新，数字输出会在 20 kHz 左右更新。LW50 系列在 0°C 到 60°C 的温度范围内进行校准。该传感器模拟输出可在 3.3Vdc 或 5.0Vdc 的单电源条件下工作，数字输出可在 1.8V 到 3.6V 的单电源条件下工作。这些传感器测量绝压、差压和表压。绝压型号的传感器具备内部真空参照以及与绝压成比例的输出值。差压型号的传感器允许向感应模片的任意一侧加压。表压型号的传感器以大气压力为参考，提供与大气压力变化成比例的输出值。LW50 压力传感器适用于无腐蚀性、非离子气体 (例如空气和其他干燥气体)。提供的选件可延伸这些传感器的性能，使其适用于无腐蚀性、非离子的液体。 所有产品均遵循 ISO 9001 标准设计和制造。

产品特点

- 多样化的封装：LW50 系列压力传感器为硅压阻精密压力传感器，采用模块化设计，具有多种封装类型可供选择（侧面供气、DIP 型、SMT 型），可满足客户不同安装环境的需要
- 工作电压较低，能耗极小，模拟供电电压可以为 3.3V 或 5V 输入，数字为 1.8V 到 3.6V
- 业界领先的长期稳定性：通过压力敏感芯片的优选和封装工艺的技术处理，作为微压力传感器与业内其他传感器相比表现出色，具有优异的长期稳定性
- 还提供了模拟输出选项
- 绝压、差压和表压类型
- 达到 0.25% FSS BFSL（满刻度量程最佳直线）的极高精确度
- 总误差带为 5% 的满刻度量程最大值
- 所有这些产品都同样具备业界领先的性能规格
- 模拟数字放大输出提供 10% 到 90% 输出或 5% 到 95% 输出
- 在 0℃ 到 60℃ 的温度范围内进行精密 ASIC 调节和温度补偿
- 符合 RoHS 标准
- 压力口特点：直径 3.175MM 倒钩状压力口可以稳固的和 2.38MM 内径的压力管牢固连接测试压力
- 客户定制：精度、总偏差和温度补偿范围以及信号输出方式等可根据客户需求定制，非标准产品请联系我们

标准压力范围量程（英寸水柱，PA）

1 PSI	表压、差压	模拟放大，数字输出
2 PSI	表压、差压	模拟放大，数字输出
5 PSI	表压、差压	模拟放大，数字输出
15 PSI	表压、差压、绝压	模拟放大，数字输出
30 PSI	表压、差压、绝压	模拟放大，数字输出
50 PSI	表压、差压、绝压	模拟放大，数字输出
100 PSI	表压、差压、绝压	模拟放大，数字输出
150 PSI	表压、差压、绝压	模拟放大，数字输出

100 pa	表压、差压	模拟放大，数字输出
1 Kpa	表压、差压	模拟放大，数字输出
10 Kpa	表压、差压	模拟放大，数字输出
30 Kpa	表压、差压、绝压	模拟放大，数字输出
100 Kpa	表压、差压、绝压	模拟放大，数字输出
500 Kpa	表压、差压、绝压	模拟放大，数字输出
1000 Kpa	表压、差压、绝压	模拟放大，数字输出

额定值

参数	最小值	最大值	单位
电源电压 (Vsupply)	-0.3	6.0模拟 / 3.3数字	VDC
任意引脚上的电压	-0.3	Vsupply +0.3	V
ESD 敏感度 (人体模式)	4		Kv
储存温度	-40	125	℃
焊接时间和温度 铅焊料温度 (SIP、DIP) 回流峰值温度 (SMT)	最多 5 秒，在 250℃ 时 最多 15 秒，在 250℃ 时		

性能参数

参数	最小值	典型值	最大值	单位	注释
电源电压 (Vsupply) (根据所选料号) 3.3 5.0（模拟）	 1.8数字/2.模拟 4.75	 3.3 5.0	 3.6 5.25	 V	 2
电源电流 3.3 Vdc 电源 5.0 Vdc 电源		 2.1 3.0		 mA mA	
补偿温度范围	0		60	℃	3
工作温度范围	-40		125	℃	4
精度			±0.25	% FSS	5、7
位置灵敏度			±0.15	% FSS	
综合偏差 (TEB)	-5%		5%	% FSS	6
过载压力		3		倍	9
爆破压力		5		倍	10

环境规格

参数	特性	注释
湿度：仅干燥气体	0% 到 95% RH，非冷凝	
振动	MIL-STD-202F，方法 214，条件 F (20.7g 随机)	
冲击	MIL-STD-202F，方法 213B，条件 F	
寿命	100 万次循环	8
回流焊	J-STD-020D.1 MSL 湿度灵敏度级别 1	

被测介质接触材料

盖子	PPS	PPS
基材	高温玻璃纤维板	高温玻璃纤维板
粘合剂	环氧树脂、硅树脂	环氧树脂、硅树脂
电子组件	陶瓷、玻璃、焊料、硅	硅、玻璃、金、焊料

注释：

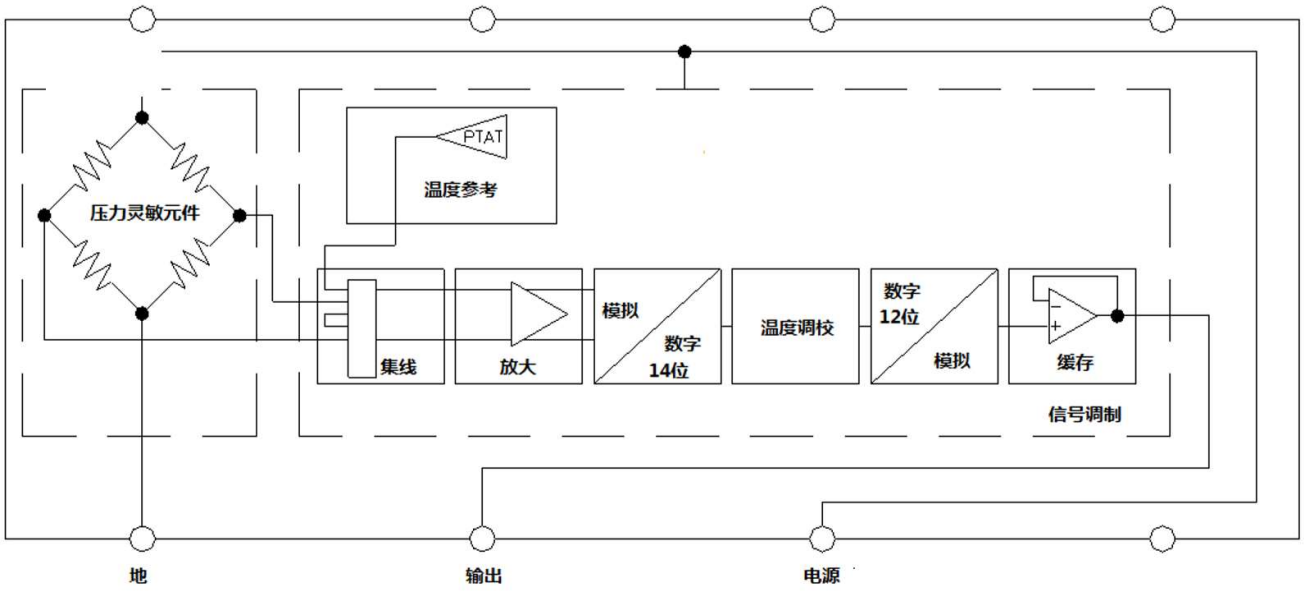
1、 额定值是设备在不损坏的前提下所能承受的最大极限。
2、 该传感器不受反向极性保护。将错误的引脚与电源连接或者接地可能会导致电气故障。
3、 补偿温度范围是指传感器可以在特定的性能限制下产生与压力成比例的输出的温度范围。
4、 工作温度范围是指传感器可以产生与压力成比例的输出的温度范围，但不一定在特定性能限制范之内。
5、 精度：相对适用于在 25°C 时的压力范围内所测输出的最佳直线 (BFSL) 的最大输出偏差。包括所有因压力非线性、压力滞后和不重复性造成的误差。
6、 综合偏差：相对整个补偿温度和压力范围内理想传递函数的最大偏差。包括所有因偏置、满刻度量程、压力非线性、压力滞后、可重复性、偏置热效应、量程热效应和热滞后造成的误差。
7、 满刻度量程 (FSS) 是指在压力范围最大限制值 (Pmax.) 和最小限制值 (Pmin.) 处测得的输出信号之间的代数差。
8、 寿命可能因传感器使用的特定应用而有所变化。
9、 过压：可安全施加到产品的最大压力，使产品在压力返回到工作压力范围时规格保持不变。施加过高的压力可能会对产品造成永久损坏。除非另有规定，否则这适用于工作温度范围内任何温度下的所有可用压力口。
10、爆破压力：可施加到产品的任意压力口而不造成压力媒介脱离的最大压力。在经受任何超过爆破压力的压力之后，产品将不能正常工作。
11、客户定制请联系零壹智控业务人员。

注意：

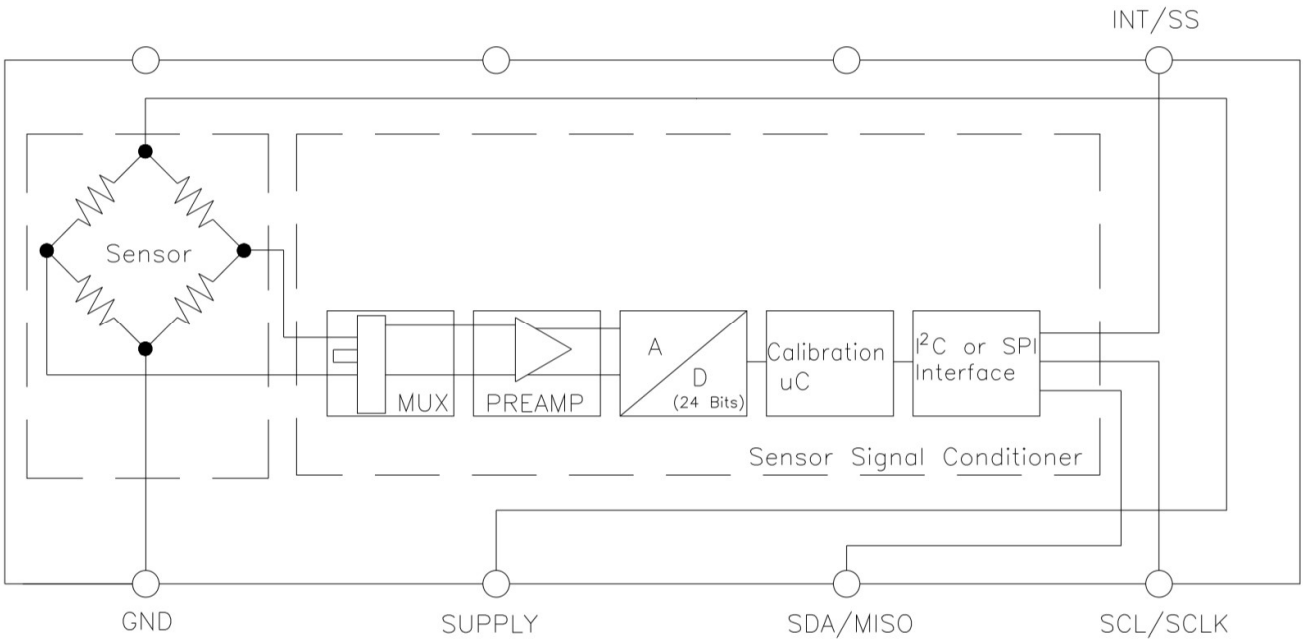
产品损坏

- 确保液体介质仅用于压力口 A；压力口 B 与液体不相容。
- 确保液体介质不含颗粒。所有 LW50 传感器均为终端密封设备，颗粒会在传感器内积聚，造成设备损坏或影响传感器输出。
- 建议将传感器的压力口 A 朝下放置，这样系统中的颗粒就不容易进入并停留在压力传感器内。
- 确保液体介质在干燥时不会产生残留物。传感器内的堆积物可能会影响传感器输出。清洗终端密封的传感器十分困难，而且无法有效地去除残留物。
- 确保液体介质与接液材料相容。不相容的液体介质只会降低传感器的性能，并可能导致传感器故障。
- 不遵循这些说明可能会导致产品损坏。

等效电路



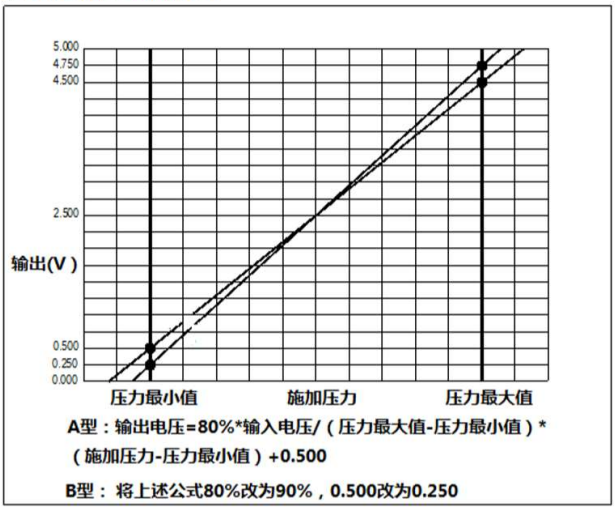
LW50 模拟放大输出等效电路



LW50 数字输出等效电路

压力和温度输出对应公式

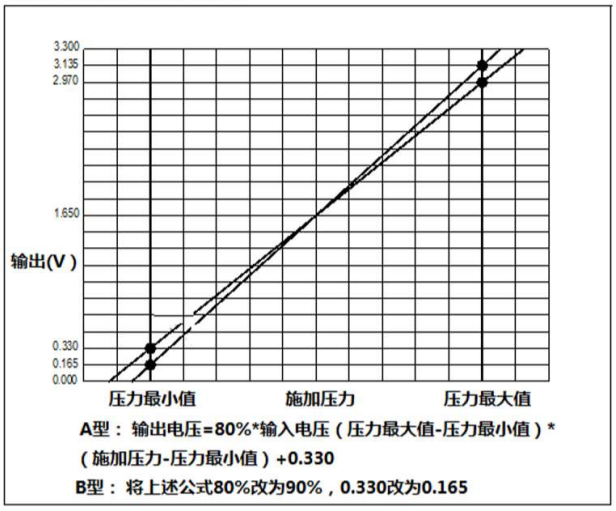
压力转换方程，电压输入5V



A型：输出电压=80%*输入电压/（压力最大值-压力最小值）*（施加压力-压力最小值）+0.500

B型： 将上述公式80%改为90%，0.500改为0.250

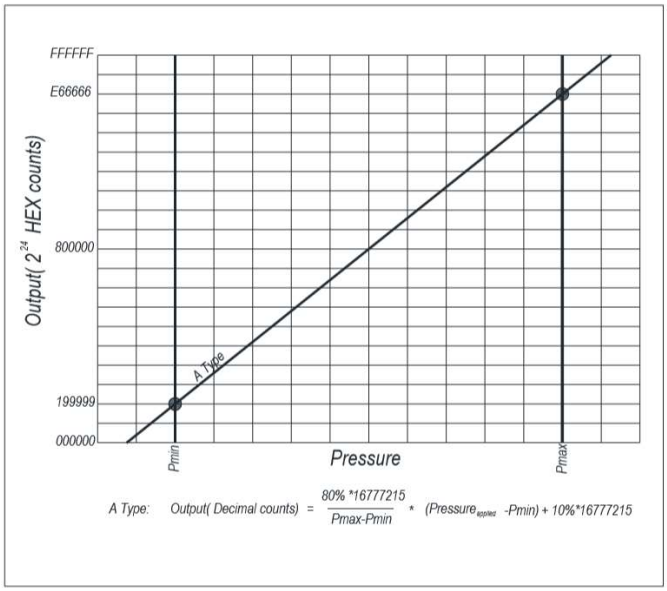
压力转换方程，电压输入 3.3V

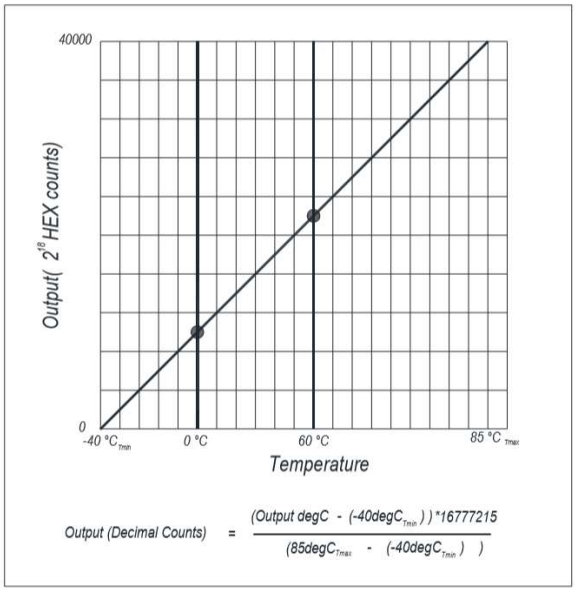


A型： 输出电压=80%*输入电压（压力最大值-压力最小值）*（施加压力-压力最小值）+0.330

B型： 将上述公式80%改为90%，0.330改为0.165

模拟计算公式





数字计算公式

压力和温度变换（模拟输出）

压力类型	说明
差压	输出与施加到各压力口的压力差成比例
表压	输出与施加到压力和大气（环境）压力之间的差值成比例
绝压	输出与施加压力和真空压力之间的差值成比例

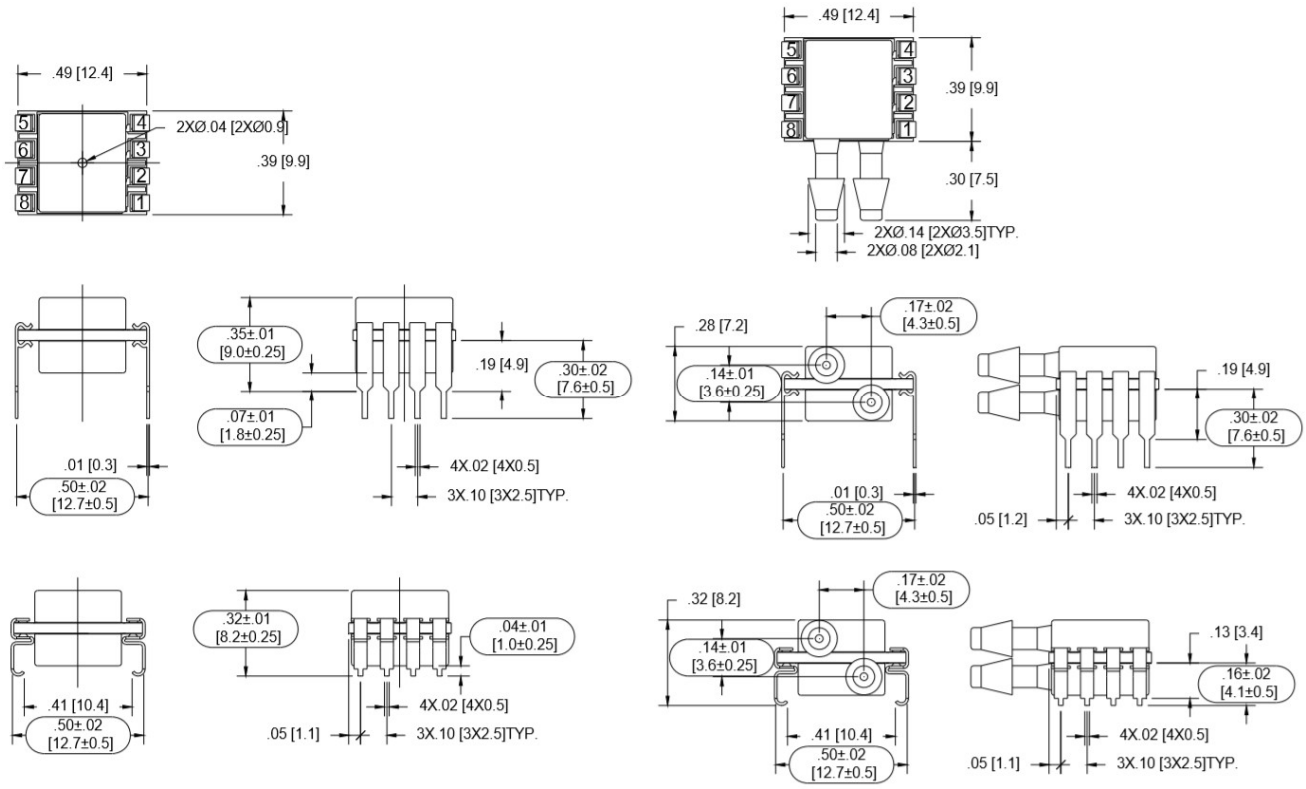
输出百分比

输出百分比（%）	模拟放大（5V）	模拟放大（3.3V）	十进制数字输出（3V）
0	0	0	0
5	0.25	0.165	838861
10	0.5	0.33	1677722
50	2.5	1.65	8388608
90	4.5	2.97	15099494
95	4.75	3.135	15938355
100	5	3.3	16777216

脚位定义

输出类型/脚位	Pin 1	Pin 2	Pin 3	Pin 4	Pin 5	Pin 6	Pin 7	Pin 8
模拟输出	空	电源	地	信号	空	空	空	空
数字输出	地	电源	SDA/MOSI	SCL/SCLK	SS	空	空	MISO

尺寸（毫米）



订购指南

LW50 A-SS 3 A X XXXX X X X

Model
LW50

Output Type
A = 模拟
D = 数字

封装类型
DS = 双盖，测出气孔
MM = 双盖，单孔出气

供电电压
3 = 3.3 Vdc
5 = 5.0 Vdc（仅模拟）

输出类型
A = 10% to 90%
B = 5% to 95%

压力类型
N = 空
I = I2C
S = SPI

湿度保护
C = 果冻胶保护
留空 = 无

焊针类型
P = 通孔
S = 表面贴装
L = 和 PCB 表面平行

压力类型
D = 差压
G = 表压
A = 绝压

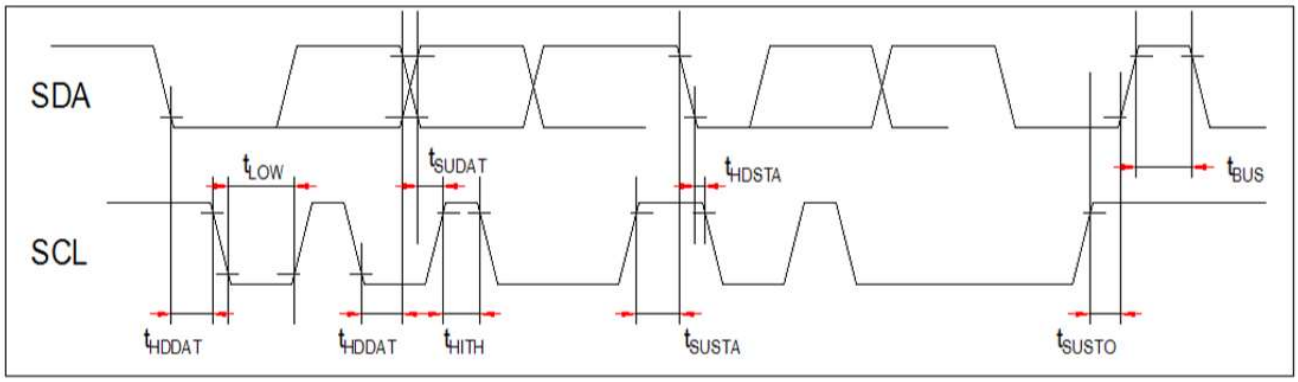
压力范围
002 m = 2 mBar
004 W = 4 InH2O
010 K = 10 Kpa
010 P = 10 Psi
020 B = 20 Bar
100 A = 100 Pascal

零壹智能控制技术（深圳）有限公司
www.linkwon.com.cn
Email：sales1@bill-well.com
深圳市 南山区 南光路 17号 现代城华庭 4栋25A
Tel：0755-2664 7945 / 189 2389 8109　Fax：0755-2641 9680

I2C SPI INTERFACE PARAMETERS & TIMING DIAGRAM

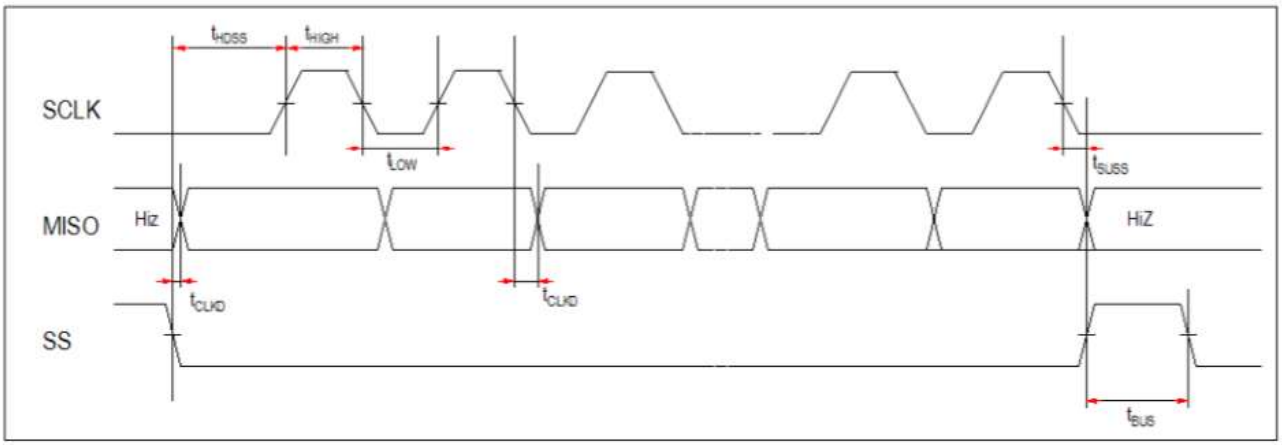
I2C INTERFACE PARAMETERS					
PARAMETERS	SYMBOL	MIN	TYP	MAX	UNITS
SCLK CLOCK FREQUENCY	FSCL	100		400	KHz
START CONDITION HOLD TIME RELATIVE TO SCL EDGE	tHDSTA	0.1			uS
MINIMUM SCL CLOCK LOW WIDTH @1	tLOW	0.6			uS
MINIMUM SCL CLOCK HIGH WIDTH @1	tHIGH	0.6			uS
START CONDITION SETUP TIME RELATIVE TO SCL EDGE	tSUSTA	0.1			uS
DATA HOLD TIME ON SDA RELATIVE TO SCL EDGE	tHDDAT	0			uS
DATA SETUP TIME ON SDA RELATIVE TO SCL EDGE	tSUDAT	0.1			uS
STOP CONDITION SETUP TIME ON SCL	tSUSTO	0.1			uS
BUS FREE TIME BETWEEN STOP AND START CONDITION	tBUS	2			uS
@1 COMBINED LOW AND HIGH WIDTHS MUST EQUAL OR EXCEED MINIMUM SCL PERIOD.					

I2C INTERFACE TIMING DIAGRAM



SPI INTERFACE PARAMETERS					
PARAMETERS	SYMBOL	MIN	TYP	MAX	UNITS
SCLK CLOCK FREQUENCY	FSCL	50		800	KHz
SS DROP TO FIRST CLOCK EDGE	tHDSS	2.5			uS
MINIMUM SCL CLOCK LOW WIDTH @1	tLOW	0.6			uS
MINIMUM SCL CLOCK HIGH WIDTH @1	tHIGH	0.6			uS
CLOCK EDGE TO DATA TRANSITION	tCLKD	0		0.1	uS
RISE OF SS RELATIVE TO LAST CLOCK EDGE	tSUSS	0.1			uS
BUS FREE TIME BETWEEN RISE AND FALL OF SS	tBUS	2			uS
@1 COMBINED LOW AND HIGH WIDTHS MUST EQUAL OR EXCEED MINIMUM SCLK PERIOD.					

SPI INTERFACE TIMING DIAGRAM



ADC Conversion Times for a Single Analog-to-Digital Conversion

Resolution (Bits)	Conversion Time in µs (typical)
12	140
13	185
14	250
15	335
16	470
17	640
18	890
19	1250
20	1760
21	2460
22	3480
23	4890
24	6940

I2C / SPI 例 程

I2C （LW50D-DS3AI-001DP）

```
#include "I2C.h"
#include "main.h"
#include "stm32l0xx_hal.h"

float Pdisplay=0;
u32 Tdisplay=0;
float Pmax=6894.757;
```

```
float Pmin=-6894.757;
float Pspan=13421772.8;
u32 Pvalue=0;
float Tmax=85;
float Tmin=-40;
u32 Tspan=13421773;
u32 Tvalue=0;

void delay_us(long int time)
{
  long int i=8*time;
  while(i--);
}

void delay_ms(long int time)//1372@4M 686@2M 343@1M
{
  long int i=1372*time;
  while(i--);
}

void IIC_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, SCL_Pin|SDA_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pins : PAPin PAPin */
  GPIO_InitStruct.Pin = SCL_Pin|SDA_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

```
void SDA_IN()
{
        GPIO_InitTypeDef  GPIO_InitStructure;
        GPIO_InitStructure.Pin = SDA_Pin;
 GPIO_InitStructure.Mode = GPIO_MODE_INPUT;
 GPIO_InitStructure.Pull = GPIO_NOPULL;
 //GPIO_InitStructure.Alternate = GPIO_PuPd_UP;
 GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_MEDIUM;
 HAL_GPIO_Init(GPIOA, &GPIO_InitStructure);
}

void SDA_OUT()
{
        GPIO_InitTypeDef  GPIO_InitStructure;
        GPIO_InitStructure.Pin = SDA_Pin;
 GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
 GPIO_InitStructure.Pull = GPIO_NOPULL;
        GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_MEDIUM;
 HAL_GPIO_Init(GPIOA, &GPIO_InitStructure);
}

void IIC_Start(void)
{
        SDA_OUT();
        IIC_SDA_ON;
        IIC_SCL_ON;
        delay_us(4);
        IIC_SDA_OFF;//START:when CLK is high,DATA change form high to low
        delay_us(4);
        IIC_SCL_OFF;
}

void IIC_Stop(void)
{
         SDA_OUT();//sda
         IIC_SCL_OFF;
         IIC_SDA_OFF;//STOP:when CLK is high DATA change form low to high
         delay_us(4);
```

```
        IIC_SCL_ON;
        IIC_SDA_ON;
        delay_us(4);
}
unsigned char IIC_Wait_Ack(void)
{
        unsigned char ucErrTime=0;
        SDA_IN();
        IIC_SDA_ON;delay_us(1);
        IIC_SCL_ON;delay_us(1);
        while(HAL_GPIO_ReadPin(GPIOA, SDA_Pin))
        {
                ucErrTime++;
                if(ucErrTime>250)
                {
                        IIC_Stop();
                        return 1;
                }
        }
        IIC_SCL_OFF;
        return 0;
}

void IIC_Ack(void)
{
        IIC_SCL_OFF;
        SDA_OUT();
        IIC_SDA_OFF;
        delay_us(2);
        IIC_SCL_ON;
        delay_us(2);
        IIC_SCL_OFF;
}

void IIC_NAck(void)
{
        IIC_SCL_OFF;
        SDA_OUT();
```

```
        IIC_SDA_ON;
        delay_us(2);
        IIC_SCL_ON;
        delay_us(2);
        IIC_SCL_OFF;
}

void IIC_Send_Byte(unsigned char txd)
{
   unsigned char t;
                SDA_OUT();
   IIC_SCL_OFF;
   for(t=0;t<8;t++)
   {
                                if(txd&0x80)
                                {IIC_SDA_ON;}
                                else
                                {IIC_SDA_OFF;}
      txd<<=1;
                delay_us(2);
                IIC_SCL_ON;
                delay_us(2);
                IIC_SCL_OFF;
                delay_us(2);
   }
}
unsigned char IIC_Read_Byte(unsigned char ack)
{
        unsigned char i,receive=0;
        SDA_IN();//SDA
   for(i=0;i<8;i++ )
        {
      IIC_SCL_OFF;
      delay_us(2);
                IIC_SCL_ON;
      receive<<=1;
      if(HAL_GPIO_ReadPin(GPIOA, SDA_Pin))receive++;
                delay_us(1);
```

```
    }
    if (!ack)
        IIC_NAck();//nACK
    else
        IIC_Ack(); //ACK
    return receive;
}

unsigned char temp[7];
void Get_Value()
{
        IIC_Start();
        IIC_Send_Byte(0x50);
        IIC_Wait_Ack();
        IIC_Send_Byte(0xaa);
        IIC_Wait_Ack();
        IIC_Stop();
        delay_ms(17);

        IIC_Start();
        IIC_Send_Byte(0x51);
        IIC_Wait_Ack();
        temp[0]=IIC_Read_Byte(1);
        temp[1]=IIC_Read_Byte(1);
        temp[2]=IIC_Read_Byte(1);
        temp[3]=IIC_Read_Byte(1);
        temp[4]=IIC_Read_Byte(1);
        temp[5]=IIC_Read_Byte(1);
        temp[6]=IIC_Read_Byte(0);
        IIC_Stop();

            if(temp[0]==0x40)
            {
                Pvalue=temp[1]*256*256+temp[2]*256+temp[3];
                Tvalue=temp[4]*256*256+temp[5]*256+temp[6];
            }

            Tdisplay=(Tvalue-1677721.6)/Tspan*(Tmax-Tmin)+Tmin;
```

```
        Pdisplay=(Pvalue-1677721.6)/Pspan*(Pmax-Pmin)+Pmin;
}


float pressure;
float filt(unsigned char N)
{
        u8 count;
        float sum=0;
        float value_buf[N];
        for (count=0;count<N;count++)
        {
                Get_Value();
                value_buf[count] = Pdisplay;
                sum += value_buf[count];
        }
        pressure=sum/N;

        return  pressure;
}
```

SPI

```
#include "main.h"
#include "stm32l0xx_hal.h"
#include "SPI.h"
#include "delay.h"

//MODE   0  0
void spi_write(u8 spi_dat)
{
   unsigned char n;
   for(n=0;n<8;n++)
   {
      OLED_SCLK_Clr();
      if(spi_dat&0x80)
         OLED_SDIN_Set();
      else
         OLED_SDIN_Clr();
```

```
        spi_dat<<=1;
        OLED_SCLK_Set();
    }
    OLED_SCLK_Clr();
}

u8 spi_read()
{
    unsigned char n ,dat;

    for(n=0;n<8;n++)
    {
        OLED_SCLK_Clr();
        dat<<=1;
        if(READ_MISO)
            dat|=0x01;
        else
            dat&=0xfe;
        OLED_SCLK_Set();
    }
    OLED_SCLK_Clr();
    return dat;
}

u8 SPIx_ReadWriteByte(u8 TxData)
{
        u8 i,RxData=0,num=0x80;
        for (i=0;i<0x08;i++)
        {
                OLED_SCLK_Clr();
                if(TxData&num)
                        OLED_SDIN_Set();
                else
                        OLED_SDIN_Clr();
                if(num>0x01)
                        num=num>>1;
                delay_ms(4);
                OLED_SCLK_Set();
```

```
            if(READ_MISO)
                RxData|=0x01;
            if(i<7)
                RxData=RxData<<1;
            delay_ms(4);
        }
        OLED_SCLK_Clr();
        return RxData;
}

unsigned char Soft_Buf_Pressure[7];
void ReadSSDL()
{
        OLED_SCLK_Clr();
        SS_OFF;
        delay_us(5);
 spi_write(0xAA);
        spi_write(0x00);
 spi_write(0x00);
        SS_ON;
 delay_ms(20);
        SS_OFF;
        Soft_Buf_Pressure[0] = SPIx_ReadWriteByte(0xf0);
        Soft_Buf_Pressure[1] = SPIx_ReadWriteByte(0x00);
        Soft_Buf_Pressure[2] = SPIx_ReadWriteByte(0x00);
        Soft_Buf_Pressure[3] = spi_read();
        Soft_Buf_Pressure[4] = spi_read();
        Soft_Buf_Pressure[5] = spi_read();
        Soft_Buf_Pressure[6] = spi_read();
        SS_ON;
        delay_us(5);
}
```