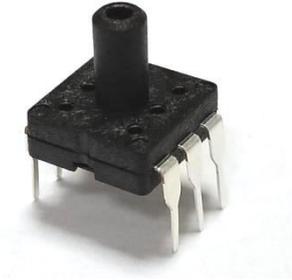


产品规格书

SV-DIP6-010D 压力传感器模块



1、产品描述

SV-DIP6-010D是一种由MEMS压阻式压力传感器芯片和专用信号调理芯片组成的数字输出压差传感器。ASIC包括24位Sigma-Delta ADC、OTP存储器和接口电路。存储在OTP存储器中的校准数据可用于SV-DIP6-010D的校准。压力校准和温度补偿应由外部微控制器实现。每个模块都经过线性校验和温度补偿，使其电压输出信号线性正比于输入压力，并不受工作温度影响

2、产品特点

- 压力范围：0~10kPa
- 待机电流：<0.1 μ A
- 高分辨率：1Pa (RMS, 高分辨模式)
- 数字I²C接口, 24bit ADC
- 工作温度范围：-20 $^{\circ}$ C ~ 85 $^{\circ}$ C
- 可靠性高、稳定性好、长期漂移低

3. 产品应用

- 水位压力测试
- 工业/工程气压控制
- 表压传感器系统
- 医疗仪器

4. 极限参数

表1: 极限参数

参数	条件	最小	最大	单位
存储温度		-40	+105	°C
供电电压	所有的引脚	-0.3	+3.6	V
IO 引脚电压	所有的引脚	-0.3	+3.6	V
ESD		-2	+2	kV
过载			30	kPa

5.性能参数

表2: 性能参数

(Vs=3.3VDC, TA=25°C)

参数	符号	条件			最小值	典型值	最大值	单位
工作温度	TA				-20		85	°C
工作量程	P				0		10	kPa
供电电压	VDD				1.8	3.3	3.6	V
电流消耗 (1 Hz)	I _{dd}	工作模式	OSR_P	OSR_T				μA
		低功耗模式	1024X	2048X		10		
		次低功耗模式	2048X			14		
		标准模式	4096X			19		
		高精度模式	8192X			28		
		超高精度模式	16384X			46		
待机电流	I _{dds} bm	@25°C				0.1		
压力/温度测量时间	T _c	工作模式	OSR_P	OSR_T				ms
		低功耗模式	1024X	2048X		13		
		次低功耗模式	2048X			19		
		标准模式	4096X			31		
		高精度模式	8192X			56		
		超高精度模式	16384X			105		
ADC 分辨率		压力				24		Bit
		温度				16		Bit
压力精度	P_A	0-65°C			-1		+1	%F.S
温度精度	T_A	0~65°C			-1.5		1.5	°C
串行数据时钟	f _{I²C}	I ² C					3.4	MHz



6.工作模式

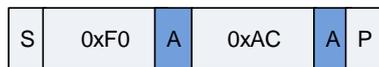
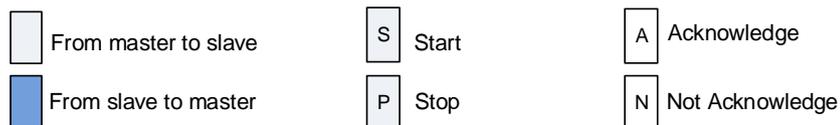
产品上电后仅在接收到相应的 I²C 命令后才会启动一次压力和温度的测量及校准过程，完成测量后自动进入深度休眠状态以节省功耗。在深度休眠时，内部电路中除 I²C 接口外其余电路全部关闭，消耗的电流约为 0.1 μA。

6.1传感器接口

I²C接口

传感器默认的7bits I²C设备地址为0x78

6.2 I²C写入



写命令中的 0xF0 表示默认的 7bits I²C 设备地址为 0x78，最后 1bit 为 0 表示主设备写操作。

6.3 I²C读取

要保证写指令和读指令的时间间隔大于测量的时长才能够读出校准数据，读数格式如下图所示，读命令中的 0xF1 表示默认的 7bits I²C 设备地址为 0x78，最后 1bit 为 1 表示主设备读操作。读到的校准数据共 6 个字节，依次为 1 字节状态字，3 字节电桥校准值，2 字节温度校准值。



图 1: 读出 3 字节电桥原始测量值或温度原始测量值



6.4 I²C 通讯协议

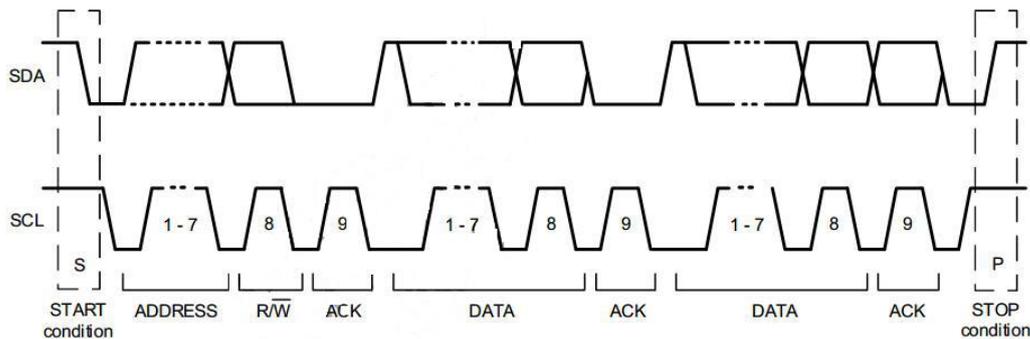


图2: I²C 通讯协议

➤ START Condition

SDA由空闲高状态转换为低状态，这是SCL保持高，这也能在传输过程中重复发送start condition，这预示通讯将会重新开始而没有中间的停止位。

➤ Address Bits

在第一个字节传输过程中，前7-bits提供设备的指定地址，默认为0x78，这个地址的设备将会应答本次的通讯。

➤ Read/Write Direction Bit

在第一个字节传输过程中，最后一比特指出通信方向，0表示主设备写操作，1表示主设备读操作，如果主设备请求读从设备，则主设备将在后来的字节控制SDA线输出数据。

➤ Data Byte

所有其他的字节，除了地址和读、写位，在SDA上传输被认为是通信的数据字节。



➤ Acknowledge or Not Acknowledge Bit

应答位用来告诉发送者字节已经接受到，设备收到数据需要应答每个字节，包括地址字节，在这个时刻，发送数据的总线设备停止驱动SDA线并且SDA线被拉高，不应答一个字节，接受设备不需要做任何事。应答一个字节，接受设备需要把SDA拉低。

一个接受从设备不需要应答，如果从设备不是寻址的设备或者设备不能处理接受的字节，主设备不应答，如果主设备在接受中并且想要结束通讯，如果遇到不应答，设备传输数据需要产生一个停止位。

➤ Stop Condition

SDA 从低状态转换到高状态，而且SCL 保持高，这时结束I²C通信。

7.用户获取数据

发送 0xAC 命令可以获取到利用内部算法校准的数据。发送 0xAC 命令获取校准值的步骤如下：

7.1 发写命令

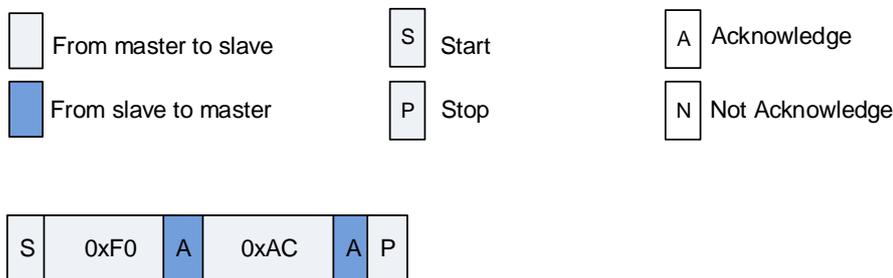


图 3： 写命令



写命令中的 0xF0 表示默认的 7bits I²C 设备地址为 0x78，最后 1bit 为 0 表示主设备写操作。

7.2 等待

发送完写命令后需要等待一段时间再发送读命令，因为内部完成整个测量需要一段时间。等待的时长取决于 OTP (Address: 0x14)的[13:11]压力过采样率和 OTP(Address: 0x14)的[15:14]温度过采样率的设置。

等待的时长不需要计算，可以通过不断的读 IIC 状态字的方式判断出是否采集已经完成。

7.3 读数

要保证写指令和读指令的时间间隔大于测量的时长才能够读出校准数据，读数格式如图 4 所示，读命令中的 0xF1 表示默认的 7bits I²C 设备地址为 0x78，最后 1bit 为 1 表示主设备读操作。读到的校准数据共 6 个字节，依次为 1 字节状态字，3 字节电桥校准值，2 字节温度校准值。

S	0xF1	A	Status	A	BridgeDat [23:16]	A	BridgeDat [15:8]	A	BridgeDat [7:0]	A	TempDat [15:8]	A	TempDat [7:0]	N	P
---	------	---	--------	---	-------------------	---	------------------	---	-----------------	---	----------------	---	---------------	---	---

图 4: I²C 读出 5 字节校准后的电桥和温度值

7.4 换算

读到校准数据后，需要将以百分比形式表示的无符号数进行简单的换算。

为方便理解我们假设读到的校准数据为：0x04 0x9B 0xB0 0xC5 0x56 0xAA

0x04 为状态字 Bit5 为 1 表明最近一次 I²C 忙，需要等待一段时间。如果 Bit5 为 0 表明设备非忙，可以读取数据。关于状态字各比特的详细描述请参见附录。

0x9B 0xB0 0xC5 三个字节为电桥校准值

0x56 0xAA 两个字节为温度校准值

电桥校准值换算：将 0x9B 0xB0 0xC5 转换为十进制数为 10203333，



本次计算假设校准时使用的量程为 0~10kPa，对应的 AD 输出为 2516582.4~14260633.6 (15%AD~85%AD)

根据 P2 输入输出关系校准公式得到：

$$\text{实际压力值} = (10 - 0) / (14260633.6 - 2516582.4) * (10203333 - 2516582.4) + 0 = 6.545 \text{ kPa}$$

温度校准值换算：将 0x56 0xAA 转换为十进制数为 22186，由于读取到的校准数据是以百分比形式表示的，这个百分比在数值上等于我们换算得到的十进制数与 16 bits 无符号数的最大值（65535）之比，所以在换算百分比时可进行如下计算

$$22186 / 65536 * 100\% = 33.85\%$$

$$\text{温度的校准范围规定为 } -40^{\circ}\text{C} \sim 150^{\circ}\text{C} \quad \text{所以校准值} = (150 - (-40)) * 33.85\% - 40 = 24.32^{\circ}\text{C}$$

输入输出关系：

$$\text{Pressure} = \frac{P_{\text{MAX}} - P_{\text{MIN}}}{D_{\text{MAX}} - D_{\text{MIN}}} \cdot (D_{\text{test}} - D_{\text{MIN}}) + P_{\text{MIN}}$$

Pressure: 实际压力值； Dtest: 传感器的数字输出值； P_{MIN}: 传感器零点压力值；
P_{MAX}: 传感器满量程压力值； D_{MIN}: 传感器零点时对应的数字输出值； D_{MAX}: 传感器满量程时对应的数字输出值；



表 4: Status 字节比特位描述

比特位	意义	描述
Bit7	保留	固定为 0
Bit6	上电指示 (Power indication)	1 设备上电 (V _{DDB} on) ; 0 设备掉电
Bit5	忙闲指示(Busy indication)	1 设备忙, 表明最近一次 I ² C 命令所要求读取的数据还未有效。如果设备忙, 新的命令将不被处理。0 表明最近一次 I ² C 命令所要求读取的数据已经准备好被读取
Bit4	保留	固定为 0
Bit[3]	工作状态 (Mode Status)	0 NOR mode 1 CMD mode
Bit2	存储器数据完整性指示 (Memory integrity/error flag)	0 表示 OTP 存储器数据完整性测试 (CRC)通过, 1 表示完整性测试失败。对数据完整性的测试只在上电过程中(POR)计算一次, 所以被写入的新 CRC 值只能在接下来的 POR 之后使用。
Bit1	保留	固定为 0
Bit0	保留	固定为 0

7.5 Get_Cal_S 命令

Get_Cal_S 命令(0xB9~0xBE)与 Get_Cal 命令(0xAC)几乎一样, 除了测量时 ADC 的过采样率的设定有所不同。Get_Cal 命令的过采样率是由 OTP 中的配置决定的, 是固定不变的; 而 Get_Cal_S 命令的过采样率是由命令本身的内容决定的。在用户校准完传感器, 并烧写 OTP 数



据之后，Get_Cal_S 命令允许用户用不同的过样率完成测量，以实现高精度、中等精度或低精度测量。

表5 Get_Cal_S 命令

Command 0xBX(HEX)	Function	Detail
X 的第[3] Bit	测量温度时ADC的过采样率OSR_T	1: 8x 过采样率
X 的第[2:0] Bit	测量外部电桥时ADC的过采样率 OSR_P	001: 16384x 过采样率 100: 2048x 过采样率 010: 8192x 过采样率 101: 1024x 过采样率 011: 4096x 过采样率 110: 512x 过采样率

8.典型应用

8.1 应用电路

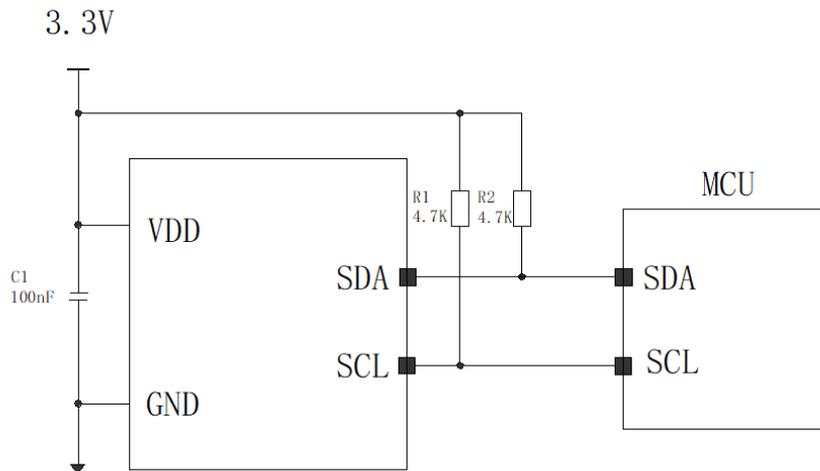


图 5：应用参考电路



8.2 参考应用结构

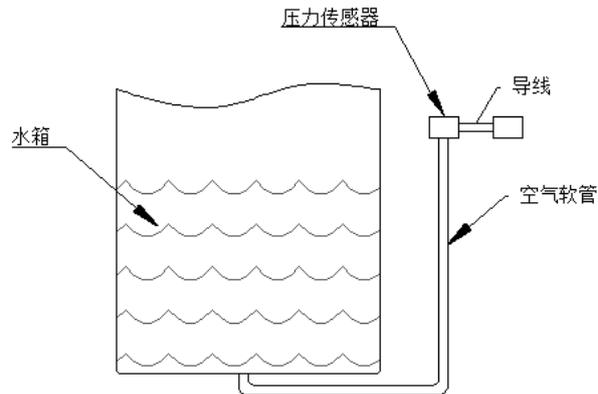


图 6: 参考应用结构

考应用结构图 6 所示，水箱的水位高低检测是根据与水箱内侧相连的水压气管里的空气压力大小来判断。通过一个塑料管子和水位传感器连接起来的放水时水进入塑料管压缩管中空气，水位的高低和管中空气的压力大小呈正比，水位越高，水压就越大，空气压力就越大，压力传感器根据收到压力的大小也会输出与空气压力呈正比的信号给到 CPU，进而可以实时监测水箱水位高度，实现自动补给或输送水箱水量。

压力对应水位高度换算：

$$P_f (\text{Pa}) = \rho gh$$

其中： P_f 为为液体对水箱底部产生的压强； ρ 为液体密度，假设水箱中液体为水，则水的密度 ρ 为 $1.0 \times 10^3 \text{ kg/m}^3$ ； g 为重力加速度 9.8 m/s^2 ； h 为液体高度，由此液体高度 h 可得到：



$$h = \frac{P_f \text{ (Pa)}}{\rho g} = \frac{P \text{ (Pa)}}{\rho g}$$

如传感器量程为 0~5kPa，则对应测量的液体水位高度 h 范围：0~510.2mm；

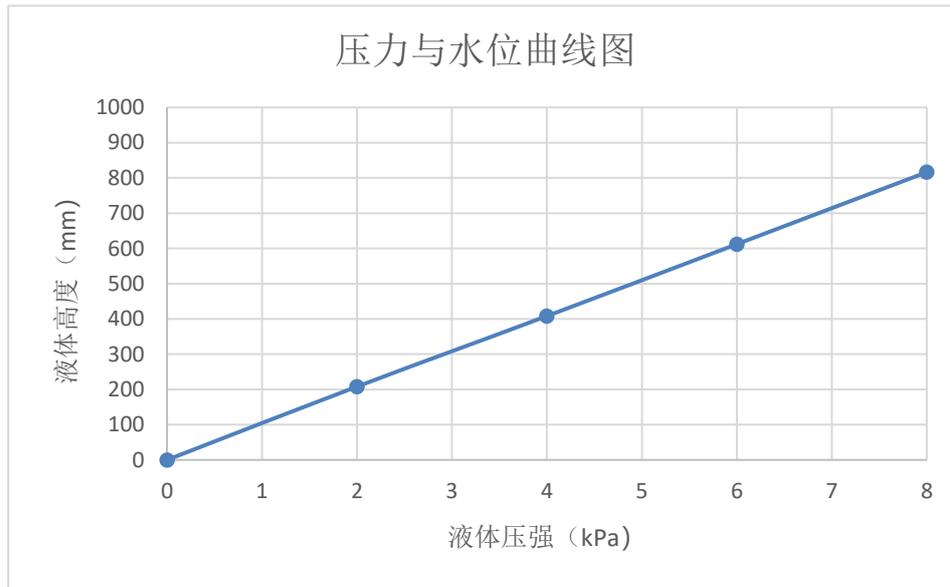


图7：压力与水位曲线图



9.机械特性

9.1引脚配置

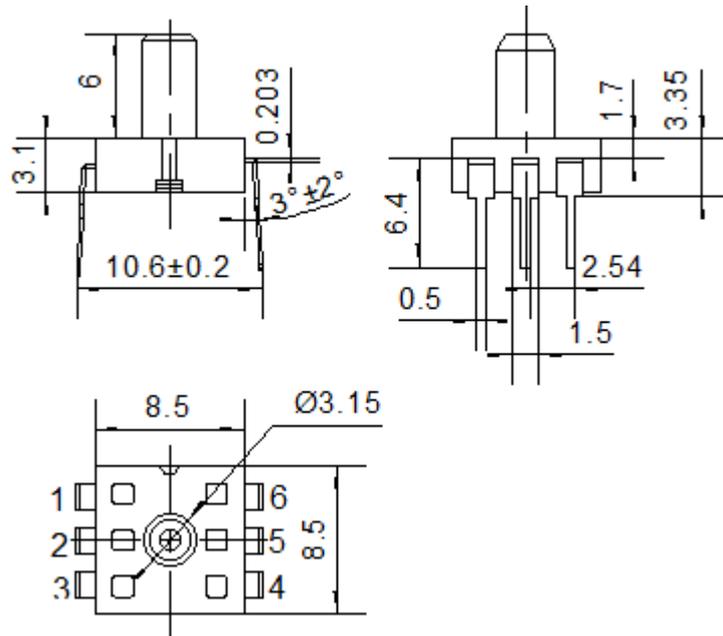


图6: 引脚配置图

表6: 引脚配置

引脚	定义	说明
1	VDD	电源正极
2	SCL	I ² C 通讯时序
3	SDA	I ² C 通讯数据
4	NC	空置不接
5	GND	电源负极
6	NC	空置不接



10.更改版本

版本号	变更内容	变更日期
1.0	新建	2020-6-18
1.1	增加IIC 参考例程	2020-8-18

11.联系方式

华景传感科技（无锡）有限公司

地址：江苏省无锡市新吴区菱湖大道200号F2栋，213135

电话：(86) 0510-85622282

传真：(86) 0510-85622278

邮箱：sales@svsens.com

网站：www.svsens.com



附件：IIC 参考例程

```
#include "3729.h"
#include "3729_IIC.h"

//IIC clock line
sbit SCL = P1 ^ 1;

//IIC data line
sbit SDA = P1 ^ 0;

//Set the input and output mode of IIC data pin
#define Set_SDA_INPUT()
    P1MDOUT &= 0xFE;
    P1 |= 0X01
#define Set_SDA_OUTPUT() P1MDOUT |= 0x01;

////Delay function needs to be defined
void DelayUs(unsigned char i)
{
}

//Start signal
void Start(void)
{
    SDA = 1;
    DelayUs(2);
    SCL = 1;
    DelayUs(2);
    SDA = 0;
    DelayUs(2);
    SCL = 0;
}

//Stop signal
void Stop(void)
{
```



```
Set_SDA_OUTPUT();
SDA = 0;
DelayUs(2);
SCL = 1;
DelayUs(2);
SDA = 1;
DelayUs(2);
}

//Read ACK signal
unsigned char Check_ACK(void)
{
    unsigned char ack;
    Set_SDA_INPUT();
    SCL = 1;
    DelayUs(2);
    ack = SDA;
    SCL = 0;
    Set_SDA_OUTPUT();
    return ack;
}

//Send ACK signal
void Send_ACK(void)
{
    Set_SDA_OUTPUT();
    SDA = 0;
    DelayUs(2);
    SCL = 1;
    DelayUs(2);
    SCL = 0;
    DelayUs(2);
}

//Send one byte
void SendByte(unsigned char byte)
{
    unsigned char i = 0;
```



```
Set_SDA_OUTPUT();
do
{
    if (byte & 0x80)
    {
        SDA = 1;
    }
    else
    {
        SDA = 0;
    }
    DelayUs(2);
    SCL = 1;
    DelayUs(2);
    byte <<= 1;
    i++;
    SCL = 0;
} while (i < 8);
SCL = 0;
}

//Receive one byte
unsigned char ReceiveByte(void)
{
    unsigned char i = 0, tmp = 0;
    Set_SDA_INPUT();
    do
    {
        tmp <<= 1;
        SCL = 1;
        DelayUs(2);
        if (SDA)
        {
            tmp |= 1;
        }
        SCL = 0;
        DelayUs(2);
        i++;
    }
```



```
    } while (i < 8);
    return tmp;
}

//Write a byte of data through IIC
uint8 BSP_IIC_Write(uint8 address, uint8 *buf, uint8 count)
{
    unsigned char timeout, ack;
    address &= 0xFE;
    Start();
    DelayUs(2);
    SendByte(address);
    Set_SDA_INPUT();
    DelayUs(2);
    timeout = 0;
    do
    {
        ack = Check_ACK();
        timeout++;
        if (timeout == 10)
        {
            Stop();
            return 1;
        }
    } while (ack);
    while (count)
    {
        SendByte(*buf);
        Set_SDA_INPUT();
        DelayUs(2);
        timeout = 0;
        do
        {
            ack = Check_ACK();
            timeout++;
            if (timeout == 10)
            {
                return 2;
            }
        }
    }
}
```



```
        }
    } while (0);
    buf++;
    count--;
}
Stop();
return 0;
}

//Read a byte of data through IIC
uint8 BSP_IIC_Read(uint8 address, uint8 *buf, uint8 count)
{
    unsigned char timeout, ack;
    address |= 0x01;
    Start();
    SendByte(address);
    Set_SDA_INPUT();
    DelayUs(2);
    timeout = 0;
    do
    {
        ack = Check_ACK();
        timeout++;
        if (timeout == 4)
        {
            Stop();
            return 1;
        }
    } while (ack);
    DelayUs(2);
    while (count)
    {
        *buf = ReceiveByte();
        if (count != 1)
            Send_ACK();
        buf++;
        count--;
    }
}
```



```
    Stop();
    return 0;
}

// Define the upper and lower limits of the calibration pressure
#define PMIN 0 //Full range pressure for example 0kPa
#define PMAX 10000 //Zero Point Pressure Value, for example 10kPa
#define DMIN 2516582.4 //AD value corresponding to pressure zero, for example 15%AD
#define DMAX 14260633.6 //AD Value Corresponding to Full Pressure Range, for example 85%AD

//The 7-bit IIC address of the 3729 is 0x78
uint8 Device_Address = 0x78 << 1;

//Delay function needs to be defined
void DelayMs(uint8 count)
{
}

//Read the status of IIC and judge whether IIC is busy
uint8 3729_IsBusy(void)
{
    uint8 status;
    BSP_IIC_Read(Device_Address, &status, 1);
    status = (status >> 5) & 0x01;
    return status;
}

void 3729_get_cal(void)
{
    uint8 buffer[6] = {0};
    uint32 Dtest = 0;
    uint16 temp_raw = 0;
    double pressure = 0.0, temp = 0.0;

    //Send 0xAC command and read the returned six-byte data
    buffer[0] = 0xAC;
    BSP_IIC_Write(Device_Address, buffer, 1);
    DelayMs(5);
}
```



```
while (1)
{
    if (3729_IsBusy())
    {
        DelayMs(1);
    }
    else
        break;
}
BSP_IIC_Read(Device_Address, buffer, 6);

//The returned pressure and temperature values are converted into actual values according
to the calibration range
Dtest = ((uint32)buffer[1] << 16) | ((uint16)buffer[2] << 8) | buffer[3];
temp_raw = ((uint16)buffer[4] << 8) | (buffer[5] << 0);
pressure = (PMAX-PMIN)/(DMAX-DMIN)*(Dtest-DMIN)+PMIN;
temp = (double)temp_raw / 65536;
temp = temp * 19000 - 4000;
}
```