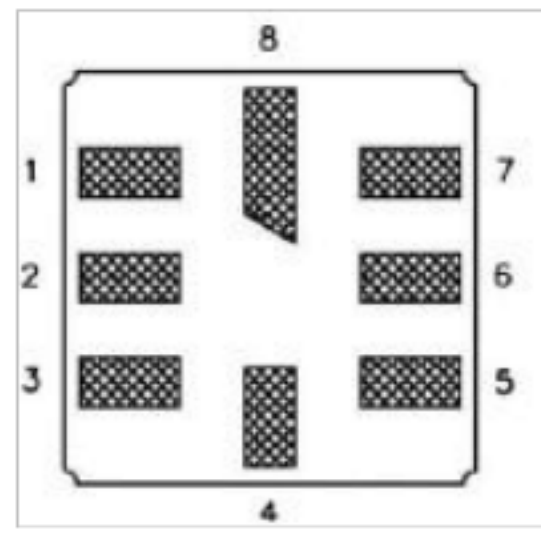
气压传感器（ BMP085 ）

在测量海拔高度时， 传统的做法是通过测量某一高度的大气压力， 再经过换算才能得到高度数据。为了测量大气压力，就得用上气压传感器，下面就来讨论一下气压传感器的应用。

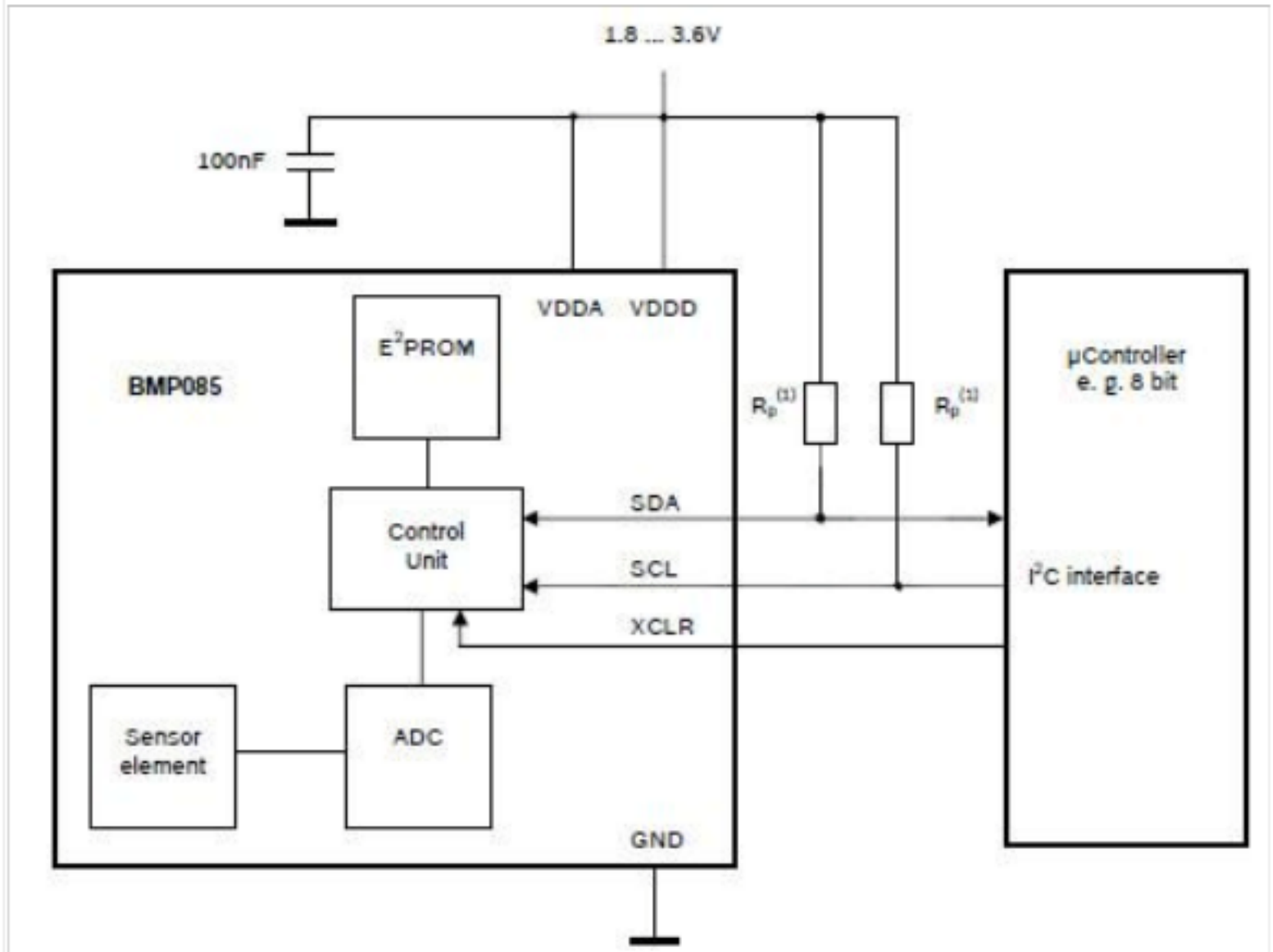
气压传感器是压力传感器中的一种， 它专用于测量气体的绝对压强。 目前市场上能见到的气压传感器有很多种， 下面就以市场上常见的 Bosch 公司推出的 BMP085 来进行讨论。 BMP085 不仅可以实时的测量大气压力，还能测量实时温度。同时它还具有 IIC 总线的接口，便于单片机进行访问。另外它的使用也很方便，不需要太多的操作就可读取到气压及测量数据。

BMP085 采用强大的 8脚陶瓷无引线芯片承载（ LCC）超薄封装，它性能卓越，内置有校准补偿，绝对精度最低可以达到 0.03hPa（0.25米），并且耗电极低，只有 3μA。气压测量范围从300hPa 到1100hPa，换算成高度为海拔 9000米到 500米。下图是其封装外形和引脚排列。



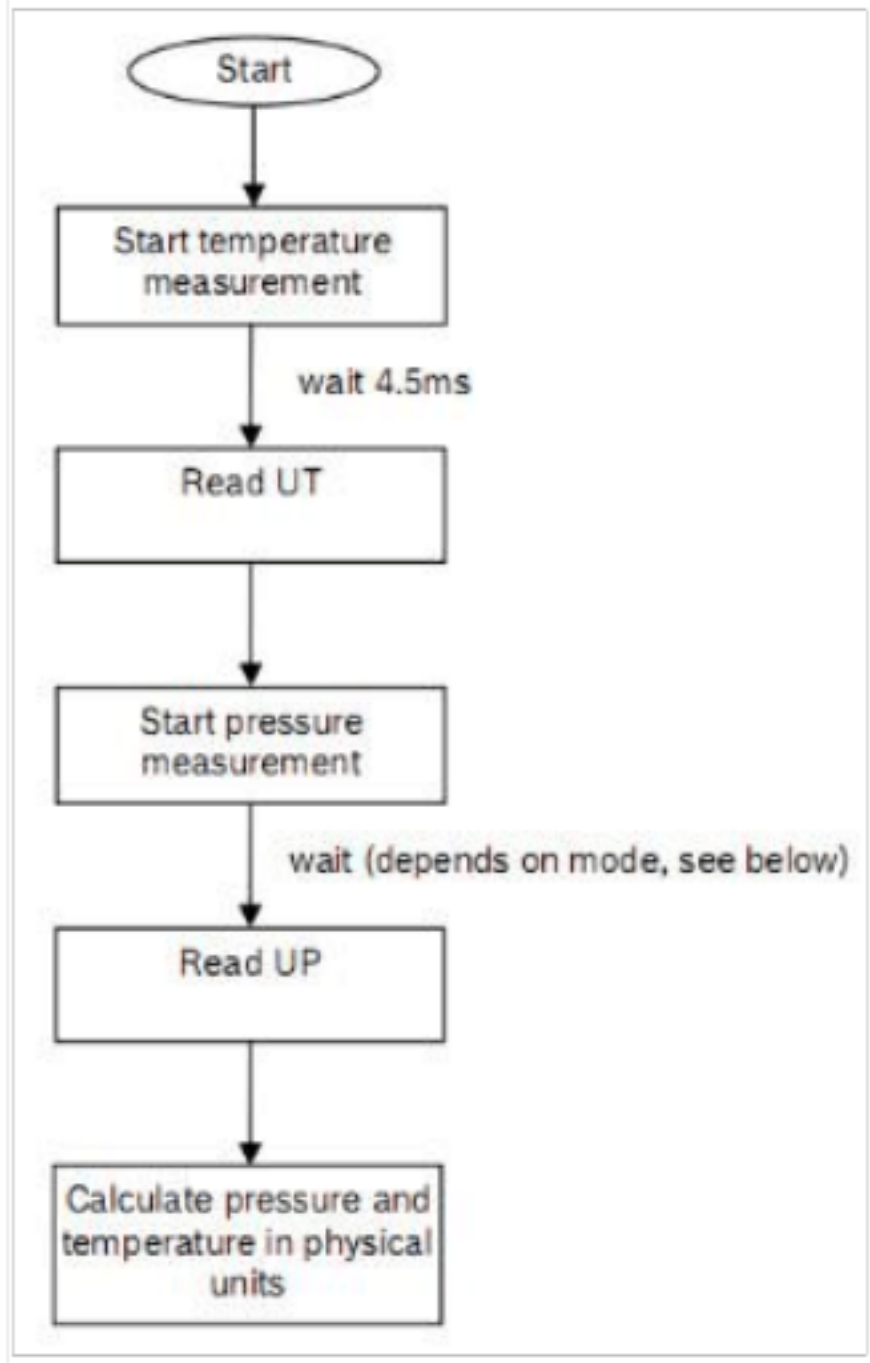
引脚各功能如下： 1脚（ GND ）接电源地， 2脚（ EOC）为完成转换输出， 3脚（ VDDA ）为正电源， 4脚（ VDDD ）为数字正电源， 5脚为空， 6脚（ SCL）为 IIC 的时钟端， 7脚（SDA）为 IIC 的数据端， 8脚（XCLR ）为主清除信号输入端，低电平有效，用来复位 BMP085 和初始化寄存器和控制器，在不用的情况下可以空置。

BMP085 的工作电压为 1.8V~3.6V ，典型工作电压为 2.5V，其与单片机相连的典型电路如下图所示。



从上图中可以看到，　BMP085 内包含有电阻式压力传感器、　AD 转换器和控制单元，其中控制单元包括了　EEPROM 和 IIC 接口。读取　BMP085 时会直接传送没有经过补偿的温度值和压力值。而在　EEPROM 中则储存了 176位单独的校准数据，这些数据将对读取的温度压力值进行补偿。 176位的 EEPROM 被划分为 11个字，每个字 16位，这样就包含有　11个校准系数。每个器件模块都有自己单独的校准系数，　在第一次计算温度压力数据之前，　单片机就应该先读出读出　EEPROM 中的这些校准数据，然后再开始采集数据温度和压力数据。

和所有的　IIC 总线器件一样，　BMP085 也有一个器件的固定地址，根据其数据手册，出厂时默认 BMP085 的从机地址为　0xEE（写入方向），或 0xEF（读出方向）。温度数据　UT 和压力数据 UP 都存储在寄存器的第　0到15位之中，压力数据　UP 的精度还可扩展至　16~19位。



	BMP085 reg adr	
Parameter	MSB	LSB
AC1	0xAA	0xAB
AC2	0xAC	0xAD
AC3	0xAE	0xAF
AC4	0xB0	0xB1
AC5	0xB2	0xB3
AC6	0xB4	0xB5
B1	0xB6	0xB7
B2	0xB8	0xB9
MB	0xBA	0xBB
MC	0xBC	0xBD
MD	0xBE	0xBF

上图中左边是 Bosch 公司技术手册上提供的读取顺序的流程图，右边是 EEPROM 中的校准数据。

从流程图中可以看出，单片机发送开始信号启动温度和压力测量，经过一定的转换时间（4.5ms）后，从 IIC 接口读出结果。 为了将温度的单位换算成℃和将压力的单位换算成 hPa，需要用到 EEPROM 中的校准数据来进行补偿计算，这些数据也可以从 IIC 接口读出。事实上， EEPROM 中的这些校准数据应该在程序初始化的时候就读出，以方便后面的计算。 在同一个采样周期中 BMP085 可以采 128次压力值和 1次温度值， 并且这些值在读取后会被及时更新掉。若不想等待到最大转化时间之后才读取数据，可以有效利用 BMP085 的输出管脚 EOC 来检查转化是否完毕。若为 1表示转换完成，为 0表示转换正在进行中。

要得到温度或气压的值，必须要访问地址为 0xF4的控制寄存器。它根据写入数据的不同，回应的值也不一样，具体如下表所示。

Measurement	Control register value (register address 0xF4)	Max. conversion time [ms]
Temperature	0x2E	4.5
Pressure (osrs = 0)	0x34	4.5
Pressure (osrs = 1)	0x74	7.5
Pressure (osrs = 2)	0xB4	13.5
Pressure (osrs = 3)	0xF4	25.5

从图中可以看出，要获得温度数据，必须先向控制寄存器（地址 0xF4）写 0x2E，然后等待至少 4.5ms，才可以从地址 0xF6 和0xF7 读取十六位的温度数据。同样，要获得气压数据，必须先向控制寄存器（地址 0xF4）写 0x34，然后等待至少 4.5ms，才可以从地址 0xF6和0xF7 读取16位的气压数据，若要扩展分辨率，还可继续读取 0xF8（XLSB ）扩展 16位数据到 19位。获取到的数据还要根据 EEPROM 中的校准数据来进行补偿后才能用， EEPROM 的数据读取可根据上图中的地址来进行，地址从 0xAA~0xBF ，具体的补偿算法可参看官方的数据手册，这里就不赘述了。

下面以一个例子来看一下 BMP085 的具体应用。

例子：利用单片机读取来自 BMP085 的温度和气压数据，并把它们通过 LCD1602 显示出来。

BMP085 的 SDA、SCL 端分别接到 ATMega16 的 TWI 端（PC1、PC0），EOC 和 XCLR 端悬空， LCD1602 的接法与前面的一致。参考代码如下。

```
#include <iom16.h>
//=========================  定义从器件地址和读写方式
==============================
#define rd_device_add 0xef //  即11101111，1110111 是 BMP085 器件的固定地址，最后的 1 表示对从器件进行读操作
#define wr_device_add 0xee //  即11101110，1110111 是 BMP085 器件的固定地址， 最后的 0 表示对从器件时行写操作
//================================TWI  状态定义
==================================
#define START 0x08
#define RE_START  0x10
#define MT_SLA_ACK  0x18
#define MT_SLA_NOACK  0x20
```

```
#define MT_DATA_ACK 0x28
#define MT_DATA_NOACK 0x30
#define MR_SLA_ACK        0x40
#define MR_SLA_NOACK        0x48
#define MR_DATA_ACK 0x50
#define MR_DATA_NOACK 0x58
//============================        常用  TWI  操作定义
==============================
#define Start() (TWCR=(1<<TWINT)|(1<<TWSTA)|(1<<TWEN))
#define Stop()      (TWCR=(1<<TWINT)|(1<<TWSTO)|(1<<TWEN))
#define Wait()      {while(!(TWCR&(1<<TWINT)));}
#define TestAck() (TWSR&0xf8)
#define SetAck()       (TWCR|=(1<<TWEA))
#define SetNoAck()       (TWCR&=~(1<<TWEA))
#define Twi() (TWCR=(1<<TWINT)|(1<<TWEN))
#define Write8Bit(x)       {TWDR=(x);TWCR=(1<<TWINT)|(1<<TWEN);}
//============   引脚电平的宏定义   ===============
#define LCM_RS_1 PORTB_Bit0=1              //RS  脚输出高电平
#define LCM_RS_0 PORTB_Bit0=0              //RS  脚输出低电平
#define LCM_RW_1 PORTB_Bit1=1              //RW  脚输出高电平
#define LCM_RW_0 PORTB_Bit1=0              //RW  脚输出低电平
#define LCM_EN_1 PORTB_Bit2=1              //EN  脚输出高电平
#define LCM_EN_0 PORTB_Bit2=0              //EN  脚输出低电平
#define DataPort PORTA                     //PORTA  为数据端口
#define Busy 0x80                  //忙信号
//==============   定义全局变量   ================
unsigned char ge,shi,bai,qian,wan,shiwan;              //显示变量
unsigned char ReadTemp[2];                              //接收到的温度数据缓冲区
unsigned char ReadPressure[2];                          //接收到的气压数据缓冲区
int ac1;
int ac2;
int ac3;
unsigned int ac4;
unsigned int ac5;
unsigned int ac6;
int b1;
int b2;
```

```
int mb;
int mc;
int md;
//==============   定义显示字符串   ================
const unsigned char str0[]={"         T:    .      C       "};       //显示温度
const unsigned char str1[]={"         P:    .      Kpa     "};       //显示气压
//===============1mS      延时 ====================
void delay_1ms(void)
{
   unsigned int i;
   for(i=1;i<(unsigned int)(8*143-2);i++)
       ;
}
//=============n*1mS      延时 ===============
void delay_nms(unsigned int n)
{
   unsigned int i=0;
   while(i<n)
    {delay_1ms();
      i++;
    }
}
//===============IIC     总线写  n 个字节（成功返回   0，失败返回  1）
====================
unsigned char I2C_Write(unsigned char RomAddress,unsigned char *buf,unsigned char
len)
{
   unsigned char i;
   Start();                                                    //启动  I2C  总线
   Wait();                                                     //等待回应
   if(TestAck()!=START)
       return 1;                                               //若回应的不是启动信
号，则失败返回   1
   Write8Bit(wr_device_add);                                   //写 I2C 从器件地址、写方向
   Wait();                                                     //等待回应
   if(TestAck()!=MT_SLA_ACK)
       return 1;                                               //若回应的不是    ACK  信
```

```
号，则失败返回值   1
   Write8Bit(RomAddress);                                    //写 BMP085 的 ROM  地址
   Wait();                                                   //等待回应
   if(TestAck()!=MT_DATA_ACK)
       return 1;                                             //若回应的不是   ACK
信号则失败返回值   1
   for(i=0;i<len;i++)
      {
         Write8Bit(buf[i]);                                  //写数据到   BMP085 的 ROM  中
         Wait();                                             //等待回应
         if(TestAck()!=MT_DATA_ACK)
            {return 1;}                                      //若回应的不是   ACK  信号则
失败返回值  1
         delay_nms(10);
      }
   Stop();                                                   //停止  I2C  总线
   delay_nms(10);                                            //延时等待   BMP085 写完
   return 0;                                                 //写入成功，返回值   0
}
//====================IIC      总线读  n 个字节（成功返回   0，失败返回  1）
=========================
unsigned char I2C_Read(unsigned char RomAddress,unsigned char *buf,unsigned char
len)
{
   unsigned char i;
   Start();                                                  //启动  I2C  总线
   Wait();                                                   //等待回应
   if(TestAck()!=START)
       return 1;                                             //若回应的不是启动
信号，则失败返回   1
   Write8Bit(wr_device_add);                                 //写  I2C  从器件地址、写方向
   Wait();                                                   //等待回应
   if(TestAck()!=MT_SLA_ACK)
       return 1;                                             //若回应的不是
ACK  信号，则失败返回值   1
   Write8Bit(RomAddress);                                    //写  BMP085 的 ROM  地址
   Wait();                                                   //等待回应
```

```
    if(TestAck()!=MT_DATA_ACK)
        return 1;                                                    //若回应的不是
ACK 信号，则失败返回值   1
    Start();                                                         //重新启动  I2C
总线
    Wait();                                                          //等待回应
    if(TestAck()!=RE_START)
        return 1;                                                    //若回应的不是
重复启动信号，则失败返回    1
    Write8Bit(rd_device_add);                                        //写 I2C 从器件地址、读方向
    Wait();                                                          //等待回应
    if(TestAck()!=MR_SLA_ACK)
        return 1;                                                    //若回应的不
是 ACK 信号，则失败返回值   1
    for(i=0;i<len;i++)
        {
            Twi();                                                   //启动
I2C 读方式
            SetAck();                                                //设置接收自动应答
            delay_nms(10);
            Wait();                                                  //等待回
应
            delay_nms(10);
            *(buf+i)=TWDR;                                           //把连续读取的   len 个字节数
据依次存入对应的地址单元（数组）中
        }
    SetNoAck();                                                      //读数据的最后一位后紧跟的
是无应答
    delay_nms(10);
    Stop();                                                          //停止 I2C 总
线
    return 0;                                                        //成功返回值  0
}
//================    检测 LCD 忙信号子函数  ================
void WaitForEnable(void)
{
        unsigned char val;
```

```
    DataPort=0xff;              //数据线电平拉高
    LCM_RS_0;                   //选择指令寄存器
    LCM_RW_1;                   //选择写方式
    __asm("NOP");               //调用汇编指令延时一个空指令周期，等待稳定
    LCM_EN_1;                   //使能端拉高电平
    __asm("NOP");
    __asm("NOP");               //调用汇编指令延时两个空指令周期，等待稳定
    DDRA=0x00;                  //改变数据线方向成输入
    val=PINA;                   //读取数据
    while(val&Busy)
      val=PINA;                 //当 DB7 位为 1时表示忙，循环检测
    LCM_EN_0;                   //忙信号结束，拉低使能端电平
    DDRA=0xff;                  //改变数据线方向成输出
}
//================    写数据到  LCD  子函数 ================
void LcdWriteData(unsigned char dataW)              //写数据  dataW  到 LCD  中
{
    WaitForEnable();            //检测忙信号
    LCM_RS_1;                   //选择数据寄存器
    LCM_RW_0;                   //选择读方式
    __asm("NOP");               //调用汇编指令延时一个空指令周期，等待稳定
    DataPort=dataW;             //把显示数据送到数据线上
    __asm("NOP");               //调用汇编指令延时一个空指令周期，等待稳定
    LCM_EN_1;                   //使能端拉高电平
    __asm("NOP");
    __asm("NOP");               //调用汇编指令延时两个空指令周期，等待稳定
    LCM_EN_0;                   //拉低使能端，执行写入动作
}
//================    写命令到  LCD  子函数 ================
void LcdWriteCommand(unsigned char CMD,unsigned char Attribc)          //写命令  CMD  到
LCD  中，  Arribc  为 1时检测忙信号，否则不检测
{
    if(Attribc)
      WaitForEnable();          //检测忙信号
    LCM_RS_0;                   //选择指令寄存器
    LCM_RW_0;                   //选择写方式
    __asm("NOP");               //调用汇编指令延时一个空指令周期，等待稳定
```

```
    DataPort=CMD;                    //把命令数据送到数据线上
    __asm("NOP");                    //调用汇编指令延时一个空指令周期，等待稳定
    LCM_EN_1;                        //使能端拉高电平
    __asm("NOP");
    __asm("NOP");                    //调用汇编指令延时两个空指令周期，等待稳定
    LCM_EN_0;                        //拉低使能端，执行写入动作
}
//================   显示光标定位子函数   ================
void LocateXY(char posx,char posy)          //定位位置到地址   x 列 y 行
{
    unsigned char temp;
    temp=posx&0x0f;              //屏蔽高 4位,限定 x 坐标的范围为 0~15
    posy&=0x01;                  //屏蔽高 7位,限定 y 坐标的范围为 0~1
    if(posy)
      temp|=0x40;                //若要显示的是第二行，则地址码    +0x40 ，因为第二行起始地址为 0x40
    temp|=0x80;                  //指令码为地址码 +0x80 ，因为写 DDRAM 时 DB7 恒为 1（即 0x80 ）
    LcdWriteCommand(temp,1);         //把 temp 写入 LCD 中，检测忙信号
}
//===========  显示指定座标的一个字符子函数    ============
void DisplayOneChar(unsigned char x,unsigned char y,unsigned char Wdata)          //在 x 列 y 行处显示变量 Wdata 中的一个字符
{
    LocateXY(x,y);               //定位要显示的位置
    LcdWriteData(Wdata);         //将要显示的数据 Wdata 写入 LCD
}
//==========  显示指定座标的一串字符子函数    ===========
void ePutstr(unsigned char x,unsigned char y,unsigned char const *ptr)          //在 x 列 y 行处显示 ptr 指向的字符串
{
    unsigned char i,j=0;
    while(ptr[j]>31)
      j++;                       //ptr[j]>31 时为 ASCII 码，j 累加， 计算出字符串长度
    for(i=0;i<j;i++)
    {
      DisplayOneChar(x++,y,ptr[i]);          //显示单个字符，同时   x 坐标递增
```

```
        if(x==16)
          {
            x=0;
            y^=1;                    //当每行显示超过   16个字符时换行继续显示
          }
      }
}
//==================LCD      初始化子函数  ==================
void InitLcd(void)
{
      LcdWriteCommand(0x38,0);              //8位数据方式，双行显示，      5X7 字形，不检测忙信
号
      delay_nms(5);                                    //延时 5ms
      LcdWriteCommand(0x38,0);
      delay_nms(5);
      LcdWriteCommand(0x38,0);
      delay_nms(5);                                    //重复三次
      LcdWriteCommand(0x38,1);              //8位数据方式，双行显示，      5X7 字形，检测忙信号
      LcdWriteCommand(0x08,1);              //关闭显示，检测忙信号
      LcdWriteCommand(0x01,1);              //清屏，检测忙信号
      LcdWriteCommand(0x06,1);              //显示光标右移设置，检测忙信号
      LcdWriteCommand(0x0C,1);              //打开显示，光标不显示，不闪烁，检测忙信号
}
//==================    转换子函数  ====================
void conversion(long temp_data)
{
      shiwan=temp_data/100000+0x30 ;
      temp_data=temp_data%100000;          //取余运算
      wan=temp_data/10000+0x30 ;
      temp_data=temp_data%10000;          //取余运算
      qian=temp_data/1000+0x30 ;
      temp_data=temp_data%1000;           //取余运算
      bai=temp_data/100+0x30          ;
      temp_data=temp_data%100;            //取余运算
      shi=temp_data/10+0x30           ;
      temp_data=temp_data%10;             //取余运算
      ge=temp_data+0x30;
```

```
}
//====================BMP085       读温度 ======================
void bmp085ReadTemp(void)
{
   unsigned char t=0x2e;
   I2C_Write(0xf4,&t,1);                                    //向地址 0xf4 写0x2e ， 进行温度转换
   delay_nms(5);                                                 //延时大于 4.5ms
   I2C_Read(0xf6,ReadTemp,2);                          //从地址 0xf6 开始读出温度数据并存到数组
ReadTemp  中，共 2个字节
}
//====================BMP085       读气压 ======================
void bmp085ReadPressure(void)
{
   unsigned char t=0x34;
   I2C_Write(0xf4,&t,1);                                        //向地址 0xf4 写 0x34 ， 进行第一次
气压转换
   delay_nms(5);                                                  //延时大于 4.5ms
   I2C_Read(0xf6,ReadPressure,2);                    //从地址 0xf6 开始读出气压数据并存到数组
ReadPressure   中，共 2 个字节
}

//==================     初始化 BMP085====================
void Init_BMP085(void)
{
   unsigned char temp[2];
   I2C_Read(0xaa,temp,2);
   ac1 = (temp[0]<<8)|temp[1];
   I2C_Read(0xac,temp,2);
   ac2 = (temp[0]<<8)|temp[1];
   I2C_Read(0xae,temp,2);
   ac3 = (temp[0]<<8)|temp[1];
   I2C_Read(0xb0,temp,2);
   ac4 = (temp[0]<<8)|temp[1];
   I2C_Read(0xb2,temp,2);
   ac5 = (temp[0]<<8)|temp[1];
   I2C_Read(0xb4,temp,2);
   ac6 = (temp[0]<<8)|temp[1];
```

```
I2C_Read(0xb6,temp,2);
b1 = (temp[0]<<8)|temp[1];
I2C_Read(0xb8,temp,2);
b2 = (temp[0]<<8)|temp[1];
I2C_Read(0xba,temp,2);
mb = (temp[0]<<8)|temp[1];
I2C_Read(0xbc,temp,2);
  mc = (temp[0]<<8)|temp[1];
  I2C_Read(0xbe,temp,2);
  md = (temp[0]<<8)|temp[1];                    //连续读取  EEPROM  中的校准数据,  并存放到相应
的变量中，以供后面补偿使用
}
//====================     转换子函数  ======================
void bmp085Convert()
{
  long ut,up,temperature,pressure;             //定义长整型变量
  long x1, x2, b5, b6, x3, b3, p;
  unsigned long b4, b7;                        //定义无符号长整型变量
  bmp085ReadT  emp();                          //读取温度
  ut=ReadT  emp[0]<<8|ReadTemp[1];        //合成温度数据
  x1=((long)ut-ac6)*ac5>>15;                   //以下根据  EEPROM  中的值对获取的温度数据的进行
补偿换算
  x2=((long)mc<<11)/(x1+md);
  b5=x1+x2;
  temperature=(b5+8)>>4;
  conversion(temperature);              //调用温度显示转换函数
  DisplayOneChar(5,0,bai);              //显示温度十位
  DisplayOneChar(6,0,shi);              //显示温度个位
  DisplayOneChar(8,0,ge);               //显示温度小数后一位

  bmp085ReadPressure();                        //读取气压
  up=ReadPressure[0]<<8|ReadPressure[1];            //合成气压数据
  up&=0x0000FFFF;
  b6=b5-4000;                                  //以下根据  EEPROM  中的值对获取的气压数据的进
行补偿换算
  x1=(b2*(b6*b6>>12))>>11;
  x2=ac2*b6>>11;
```

```
x3=x1+x2;
b3=(((long)ac1*4+x3)+2)/4;
x1=ac3*b6>>13;
x2=(b1*(b6*b6>>12))>>16;
x3=((x1+x2)+2)>>2;
b4=(ac4*(unsigned long)(x3+32768))>>15;
b7=((unsigned long)up-b3)*(50000>>0);
if(b7<0x80000000)
    p=(b7*2)/b4;
else
    p=(b7/b4)*2;
x1=(p>>8)*(p>>8);
x1=(x1*3038)>>16;
x2=(-7357*p)>>16;
pressure=p+((x1+x2+3791)>>4);
conversion(pressure);                    //调用气压显示转换函数
DisplayOneChar(4,1,shiwan);              //显示气压的百位
DisplayOneChar(5,1,wan);                 //显示气压的十位
   DisplayOneChar(6,1,qian);             //显示气压的个位
   DisplayOneChar(8,1,bai);              //显示气压小数后一位
   DisplayOneChar(9,1,shi);              //显示气压小数后二位
}
//==================     主函数 ======================
void main(void)
{
   delay_nms(400);                              //延时 400ms   等待电源稳定
   DDRA=0xff;PORTA=0x00;
   DDRB=0xff;PORTB=0x00;
   DDRC=0xff;PORTC=0xff;
   DDRD=0xff;PORTD=0xff;                        //初始化   I/O  口
   InitLcd();                                                //LCD   初始化
   Init_BMP085();                        //BMP085  初始化
   ePutstr(0,0,str0);                   //显示温度
   delay_nms(10);
   DisplayOneChar(10,0,0xdf);           //显示特殊符号
   delay_nms(10);
   ePutstr(0,1,str1);                   //显示气压
```

```
  while(1)
  {
      bmp085Convert();                  //调用转换
      delay_nms(1000);
  }
}
```

上述程序中，是运用延时来等待数据转换完成的，并没使用 EOC 脚来检查转化是否完毕。同时，程序利用 EEPROM 中的值对获取的数据进行补偿换算的方法是直接来自 BMP085 的数据手册， 若对换算代码有问题可参看其手册。 气压数据采用了常规的 16位数据， 并没有扩展到 19位。另外，由于在程序中大量使用了长整型数据格式，所以在 IAR 开发环境中编译时仍然要把编译优化选项改成 Low 或 Medium （具体参见第一章） ，系统才能正常运行。若选择不优化（ None）时，可能在液晶屏得不到任何显示。究其原因可能与 IAR 的编译环境有关，这一点要非常注意！ ！

把程序下载到单片机中， 按要求接好连线， 给系统上电， 就可以在液晶屏上看到实时的温度和气压数据了