18B20 温度传感器应用解析

温度传感器的种类众多，在应用与高精度、高可靠性的场合时 DALLAS（达拉斯）公司生产的 DS18B20 温度传感器当仁不让。超小的体积，超低的硬件开消，抗干扰能力强，精度高，附加功能强，使得 DS18B20 更受欢迎。对于我们普通的电子爱好者来说，DS18B20 的优势更是我们学习单片机技术和开发温度相关的小产品的不二选择。了解其工作原理和应用可以拓宽您对单片机开发的思路。

DS18B20 的主要特征：

?? 全数字温度转换及输出。

?? 先进的单总线数据通信。

?? 最高 12 位分辨率，精度可达土 0.5 摄氏度。

?? 12 位分辨率时的最大工作周期为 750 毫秒。

?? 可选择寄生工作方式。

?? 检测温度范围为 - 55° C ~+125° C （ - 67° F ~+257° F)

?? 内置 EEPROM，限温报警功能。

?? 64 位光刻 ROM，内置产品序列号，方便多机挂接。

?? 多样封装形式，适应不同硬件系统。

DS18B20 芯片封装结构：

DS18B20 引脚功能：

• GND 电压地 • DQ 单数据总线 • VDD 电源电压 • NC 空引脚

DS18B20 工作原理及应用：

DS18B20 的温度检测与数字数据输出全集成于一个芯片之上，从而抗干扰力更强。其一个工作周期可分为两个部分，即温度检测和数据处理。在讲解其工作流程之前我们有必要了解 18B20 的内部存储器资源。18B20 共有三种形态的存储器资源，它们分别是：

ROM 只读存储器，用于存放 DS18B20ID 编码，其前 8 位是单线系列编码（DS18B20 的编码是 19H），后面 48 位是芯片唯一的序列号，最后 8 位是以上 56 的位的 CRC 码（冗余校验）。数据在出产时设置不由用户更改。DS18B20 共 64 位 ROM。

RAM 数据暂存器，用于内部计算和数据存取，数据在掉电后丢失，DS18B20 共 9 个字节 RAM，每个字节为 8 位。第 1、2 个字节是温度转换后的数据值信息，第 3、4 个字节是用户 EEPROM（常用于温度报警值储存）的镜像。在上电复位时其值将被刷新。第 5 个字节则是用户第 3 个 EEPROM 的镜像。第 6、7、8 个字节为计数寄存器，是为了让用户得到更高的温度分辨率而设计的，同样也是内部温度转换、计算的暂存单元。第 9 个字节为前 8 个字节的 CRC 码。EEPROM 非易失性记忆体，用于存放长期需要保存的数据，上下限温度报警值和校验数据，DS18B20 共 3 位 EEPROM，并在 RAM 都存在镜像，以方便用户操作。

RAM 及 EEPROM 结构图：

图 2

我们在每一次读温度之前都必须进行复杂的且精准时序的处理，因为 DS18B20 的硬件简单结果就会导致软件的巨大开消，也是尽力减少有

形资产转化为无形资产的投入，是一种较好的节约之道。

控制器对 18B20 操作流程：

1， 复位：首先我们必须对 DS18B20 芯片进行复位，复位就是由控制器（单片机）给 DS18B20 单总线至少 480uS 的低电平信号。当 18B20 接到此复位信号后则会在 15~60uS 后回发一个芯片的存在脉冲。

2， 存在脉冲：在复位电平结束之后，控制器应该将数据单总线拉高，以便于在 15~60uS 后接收存在脉冲，存在脉冲为一个 60~240uS 的低电平信号。至此，通信双方已经达成了基本的协议，接下来将会是控制器与 18B20 间的数据通信。如果复位低电平的时间不足或是单总线的电路断路都不会接到存在脉冲，在设计时要注意意外情况的处理。

3， 控制器发送 ROM 指令：双方打完了招呼之后最要将进行交流了，ROM 指令共有 5 条，每一个工作周期只能发一条，ROM 指令分别是读 ROM 数据、指定匹配芯片、跳跃 ROM、芯片搜索、报警芯片搜索。ROM 指令为 8 位长度，功能是对片内的 64 位光刻 ROM 进行操作。其主要目的是为了分辨一条总线上挂接的多个器件并作处理。诚然，单总线上可以同时挂接多个器件，并通过每个器件上所独有的 ID 号来区别，一般只挂接单个 18B20 芯片时可以跳过 ROM 指令（注意：此处指的跳过 ROM 指令并非不发送 ROM 指令，而是用特有的一条“跳过指令”）。ROM 指令在下文有详细的介绍。

4， 控制器发送存储器操作指令：在 ROM 指令发送给 18B20 之后，紧接着（不间断）就是发送存储器操作指令了。操作指令同样为 8 位，共 6 条，存储器操作指令分别是写 RAM 数据、读 RAM 数据、将 RAM 数

据复制到 EEPROM、温度转换、将 EEPROM 中的报警值复制到 RAM、工作方式切换。存储器操作指令的功能是命令 18B20 作什么样的工作，是芯片控制的关键。

5， 执行或数据读写：一个存储器操作指令结束后则将进行指令执行或数据的读写，这个操作要视存储器操作指令而定。如执行温度转换指令则控制器（单片机）必须等待 18B20 执行其指令，一般转换时间为 500uS。如执行数据读写指令则需要严格遵循 18B20 的读写时序来操作。数据的读写方法将有下文有详细介绍。

若要读出当前的温度数据我们需要执行两次工作周期，第一个周期为复位、跳过 ROM 指令、执行温度转换存储器操作指令、等待 500uS 温度转换时间。紧接着执行第二个周期为复位、跳过 ROM 指令、执行读 RAM 的存储器操作指令、读数据（最多为 9 个字节，中途可停止，只读简单温度值则读前 2 个字节即可）。其它的操作流程也大同小异，在此不多介绍。

DS18B20 芯片与单片机的接口：

图 3

图 4

如图所示，DS18B20 只需要接到控制器（单片机）的一个 I/O 口上，由于单总线为开漏所以需要外接一个 4.7K 的上拉电阻。如要采用寄生工作方式，只要将 VDD 电源引脚与单总线并联即可。但在程序设计中，寄生工作方式将会对总线的状态有一些特殊的要求。

图 5

DS28B20 芯片 ROM 指令表：

Read ROM（读 ROM）[33H] （方括号中的为 16 进制的命令字）

这个命令允许总线控制器读到 DS18B20 的 64 位 ROM。只有当总线上只存在一个 DS18B20 的时候才可以使用此指令，如果挂接不只一个，当通信时将会发生数据冲突。

Match ROM（指定匹配芯片）[55H]

这个指令后面紧跟着由控制器发出了 64 位序列号，当总线上有多只 DS18B20 时，只有与控制发出的序列号相同的芯片才可以做出反应，其它芯片将等待下一次复位。这条指令适应单芯片和多芯片挂接。

Skip ROM（跳跃 ROM 指令）[CCH]

这条指令使芯片不对 ROM 编码做出反应，在单总线的情况之下，为了节省时间则可以选用此指令。如果在多芯片挂接时使用此指令将会出现数据冲突，导致错误出现。

Search ROM（搜索芯片）[F0H]

在芯片初始化后，搜索指令允许总线上挂接多芯片时用排除法识别所有器件的 64 位 ROM。

Alarm Search（报警芯片搜索）[ECH]

在多芯片挂接的情况下，报警芯片搜索指令只对附合温度高于 TH 或小于 TL 报警条件的芯片做出反应。只要芯片不掉电，报警状态将被保持，直到再一次测得温度什达不到报警条件为止。

DS28B20 芯片存储器操作指令表：

Write Scratchpad （向 RAM 中写数据）[4EH]

这是向 RAM 中写入数据的指令，随后写入的两个字节的数据将会被存到地址 2（报警 RAM 之 TH）和地址 3（报警 RAM 之 TL）。写入过程中可以用复位信号中止写入。

Read Scratchpad （从 RAM 中读数据）[BEH]

此指令将从 RAM 中读数据，读地址从地址 0 开始，一直可以读到地址 9，完成整个 RAM 数据的读出。芯片允许在读过程中用复位信号中止读取，即可以不读后面不需要的字节以减少读取时间。

Copy Scratchpad （将 RAM 数据复制到 EEPROM 中）[48H]

此指令将 RAM 中的数据存入 EEPROM 中，以使数据掉电不丢失。此后由于芯片忙于 EEPROM 储存处理，当控制器发一个读时间隙时，总线上输出“0”，当储存工作完成时，总线将输出“1”。在寄生工作方式时必须在发出此指令后立刻超用强上拉并至少保持 10MS，来维持芯片工作。

Convert T（温度转换）[44H]

收到此指令后芯片将进行一次温度转换，将转换的温度值放入 RAM 的第 1、2 地址。此后由于芯片忙于温度转换处理，当控制器发一个读时间隙时，总线上输出“0”，当储存工作完成时，总线将输出“1”。在寄生工作方式时必须在发出此指令后立刻超用强上拉并至少保持 500MS，来维持芯片工作。

Recall EEPROM（将 EEPROM 中的报警值复制到 RAM）[B8H]

此指令将 EEPROM 中的报警值复制到 RAM 中的第 3、4 个字节里。由于芯片忙于复制处理，当控制器发一个读时间隙时，总线上输出“0”，

当储存工作完成时，总线将输出“1”。另外，此指令将在芯片上电复位时将被自动执行。这样 RAM 中的两个报警字节位将始终为 EEPROM 中数据的镜像。

Read Power Supply（工作方式切换）[B4H]

此指令发出后发出读时间隙，芯片会返回它的电源状态字，“0”为寄生电源状态，“1”为外部电源状态。

DS18B20 复位及应答关系示意图：

图 6

每一次通信之前必须进行复位，复位的时间、等待时间、回应时间应严格按时序编程。

DS18B20 读写时间隙：

DS18B20 的数据读写是通过时间隙处理位和命令字来确认信息交换的。

写时间隙：

图 7

写时间隙分为写“0”和写“1”，时序如图 7。在写数据时间隙的前 15uS 总线需要是被控制器拉置低电平，而后则将是芯片对总线数据的采样时间，采样时间在 15~60uS，采样时间内如果控制器将总线拉高则表示写“1”，如果控制器将总线拉低则表示写“0”。每一位的发送都应该有一个至少 15uS 的低电平起始位，随后的数据“0”或“1”应该在 45uS 内完成。整个位的发送时间应该保持在 60~120uS，否则不能保证通信的正常。

读时间隙：

图 8

读时间隙时控制时的采样时间应该更加的精确才行，读时间隙时也是必须先由主机产生至少 1uS 的低电平，表示读时间的起始。随后在总线被释放后的 15uS 中 DS18B20 会发送内部数据位，这时控制如果发现总线为高电平表示读出“1”，如果总线为低电平则表示读出数据“0”。每一位的读取之前都由控制器加一个起始信号。注意：如图 8 所示，必须在读间隙开始的 15uS 内读取数据位才可以保证通信的正确。

在通信时是以 8 位“0”或“1”为一个字节，字节的读或写是从高位开始的，即 A7 到 A0.字节的读写顺序也是如图 2 自上而下的。

```
//实验目的：熟悉 DS18B20 的使用
//六位数码管显示温度结果，其中整数部分 2 位，小数部分 4 位
//每次按下 RB0 键后进行一次温度转换。
//硬件要求：把 DS18B20 插在 18B20 插座上
//          拨码开关 S10 第 1 位置 ON，其他位置 OFF
//          拨码开关 S5、S6 全部置 ON，其他拨码开关全部置 OFF

#i nclude<pic.h>
//__CONFIG(0x1832);
//芯片配置字，看门狗关，上电延时开，掉电检测关，低压编程关，加密，4M
晶体 HS 振荡

#define uch unsigned char                          //给 unsigned char 起别
名 uch
# define DQ RA0                                     //定义 18B20 数据端口
# define DQ_DIR TRISA0                              //定义 18B20D 口方向寄
存器
# define DQ_HIGH() DQ_DIR =1                        //设置数据口为输入
# define DQ_LOW() DQ = 0; DQ_DIR = 0                //设置数据口为输出
unsigned char TLV=0 ;                              //采集到的温度高 8 位
unsigned char THV=0;                               //采集到的温度低 8 位
unsigned char TZ=0;                                //转换后的温度值整数部
分
unsigned char TX=0;                                //转换后的温度值小数部
分
unsigned int wd;                                   //转换后的温度值 BCD 码
形式

unsigned char shi;                                 //整数十位
unsigned char ge;                                  //整数个位
unsigned char shifen;                              //十分位
unsigned char baifen;                              //百分位
unsigned char qianfen;                             //千分位
unsigned char wanfen;                              //万分位
unsigned char table[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0
x90};
//0-9 的显示代码

//-------------------------------------------
//延时函数
void delay(char x,char y)
{
char z;
```

```
do{
       z=y;
       do{;}while(--z);
     }while(--x);
}
//其指令时间为：7+（3*（Y-1）+7）*（X-1）如果再加上函数调用的 call 指令、页面设定、传递参数花掉的 7 个指令。
//则是：14+（3*（Y-1）+7）*（X-1）。

//-------------------------------------------------
//结果显示函数
void display()
{
   TRISA=0X00;
delay(5,100);                                  //设置 A 口全为输出
PORTD=table[shi];                              //显示整数十位
PORTA=0x3e;
delay(10,100);
PORTD=table[ge]&0X7F;                          //显示整数个位，并点亮小数点
PORTA=0x3d;
delay(10,100);
PORTD=table[shifen];                           //显示小数十分位
PORTA=0x3b;
delay(10,100);
PORTD=table[baifen];                           //显示小数百分位
PORTA=0x37;
delay(10,100);
PORTD=table[qianfen];                          //显示小数千分位
PORTA=0x2f;
delay(10,100);
PORTD=table[wanfen];                           //显示小数万分位
PORTA=0x1f;
delay(10,100);
}

//-------------------------------------------------
//系统初始化函数
void init()
{
ADCON1=0X07;                                       //设置 A 口为普通数字口
TRISA=0X00;                                        //设置 A 口方向为输出
TRISD=0X00;                                        //设置 D 口方向为输出
}
```

```
//-----------------------------------------------
//复位 DS18B20 函数
reset(void)
{
char presence=1;
while(presence)
{
    DQ_LOW() ;                                  //主机拉至低电平
    delay(2,70);                                //延时 503us
    DQ_HIGH();                                  //释放总线等电阻拉高总
线,并保持 15~60us
    delay(2,8);                                 //延时 70us
    if(DQ==1) presence=1;                       //没有接收到应答信号,继
续复位
    else presence=0;                            //接收到应答信号
    delay(2,60);                                //延时 430us
   }
}

//-----------------------------------------------
//写 18b20 写字节函数
void write_byte(uch val)
{
uch i;
uch temp;
for(i=8;i>0;i--)
{
   temp=val&0x01;                               //最低位移出
   DQ_LOW();
   NOP();
   NOP();
   NOP();
   NOP();
   NOP();                                       //从高拉至低电平,产生写
时间隙
   if(temp==1) DQ_HIGH();                       //如果写 1,拉高电平
   delay(2,7);                                  //延时 63us
   DQ_HIGH();
   NOP();
   NOP();
   val=val>>1;                                  //右移一位
}
}
```

```
//-----------------------------------------------
//18b20 读字节函数
uch read_byte(void)
{
uch i;
uch value=0;                                    //读出温度
static bit j;
for(i=8;i>0;i--)
{
   value>>=1;
   DQ_LOW();
   NOP();
   NOP();
   NOP();
   NOP();
   NOP();
   NOP();                                       //6us
   DQ_HIGH();                                   //拉至高电平
   NOP();
   NOP();
   NOP();
   NOP();
   NOP();                                       //4us
   j=DQ;
   if(j) value|=0x80;
   delay(2,7);                                  //63us
}
return(value);
}

//-----------------------------------------------
//启动温度转换函数
void get_temp()
{
int i;
DQ_HIGH();
reset();                                        //复位等待从机应答
write_byte(0XCC);                               //忽略 ROM 匹配
write_byte(0X44);                               //发送温度转化命令
for(i=50;i>0;i--)
    {

        display();                              //调用多次显示函数，确保温度
转换完成所需要的时间
```

```
    }
reset();                                    //再次复位，等待从机应答
write_byte(0XCC);                           //忽略 ROM 匹配
write_byte(0XBE);                           //发送读温度命令
TLV=read_byte();                            //读出温度低 8
THV=read_byte();                            //读出温度高 8 位
DQ_HIGH();                                  //释放总线
TZ=(TLV>>4)|(THV<<4)&0X3f;                  //温度整数部分
TX=TLV<<4;                                  //温度小数部分
if(TZ>100) TZ/100;                          //不显示百位
ge=TZ%10;//个位                             //整数部分个位
shi=TZ/10;//十位                            //整数十位
wd=0;
if (TX & 0x80) wd=wd+5000;
if (TX & 0x40) wd=wd+2500;
if (TX & 0x20) wd=wd+1250;
if (TX & 0x10) wd=wd+625;                   //以上 4 条指令把小数部分转换
为 BCD 码形式
shifen=wd/1000;                             //十分位
baifen=(wd%1000)/100;                       //百分位
qianfen=(wd%100)/10;                        //千分位
wanfen=wd%10;                               //万分位
NOP();
NOP();
}

//-----------------------------------------
//主函数
void main()
{
   init();                                  //调用系统初始化函数
   while(1)
     {
       get_temp();                          //调用温度转换函数
//   delay(5,50);

       display();                           //调用结果显示函数
     }
}
```