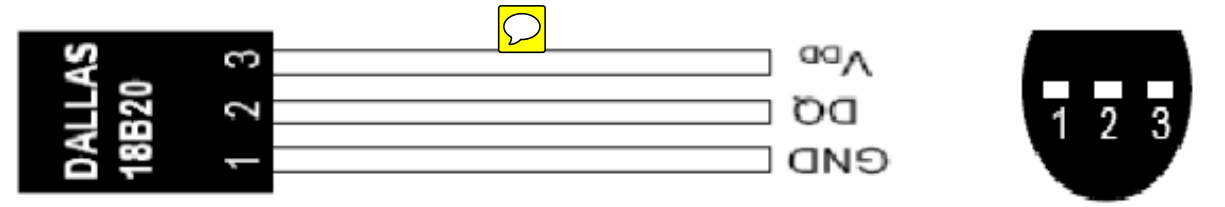
AVR 单片机学习笔记--基于慧净 HJ-2G AVR 开发板

第 14 章 DS18B20

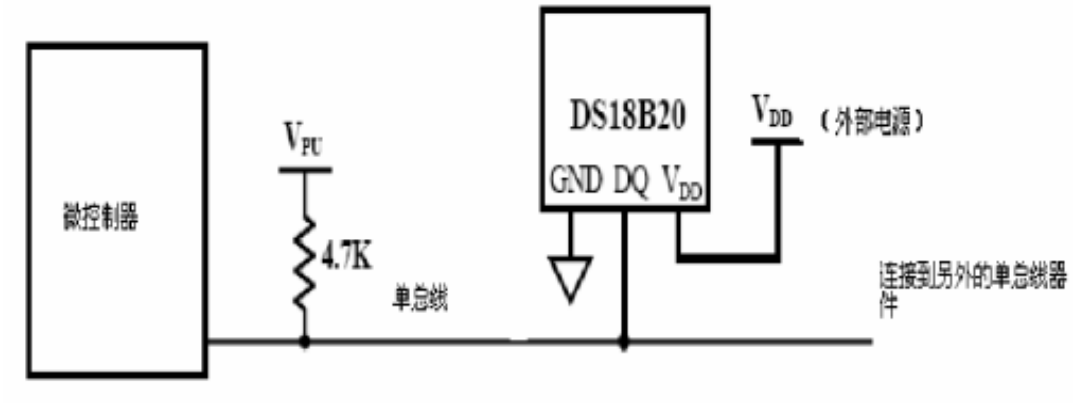
14.1 概念.

这一章是关于 DS18B20 实时温度传感器。相信有学过 c51 单片机的朋友都对他不陌生吧。我恰恰也学习过，不过当初并没有掌握好。学习板搭配的 DS18B20，一般上给人的感觉有点像三极管，其实 DS18B20 的内部结构与原理也挺猥琐的，但是我们使用也是为了实现温度传感的功能而已，基本上不会介绍过度深入。

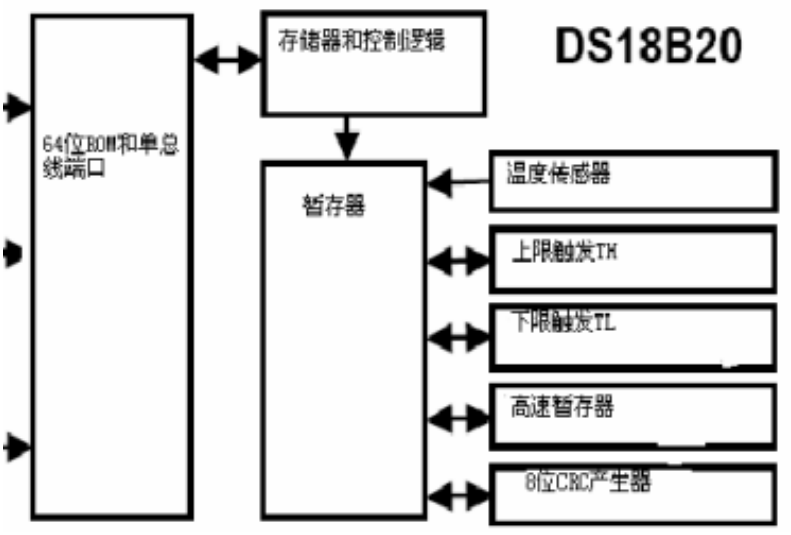
14.2 DS18B20 介绍



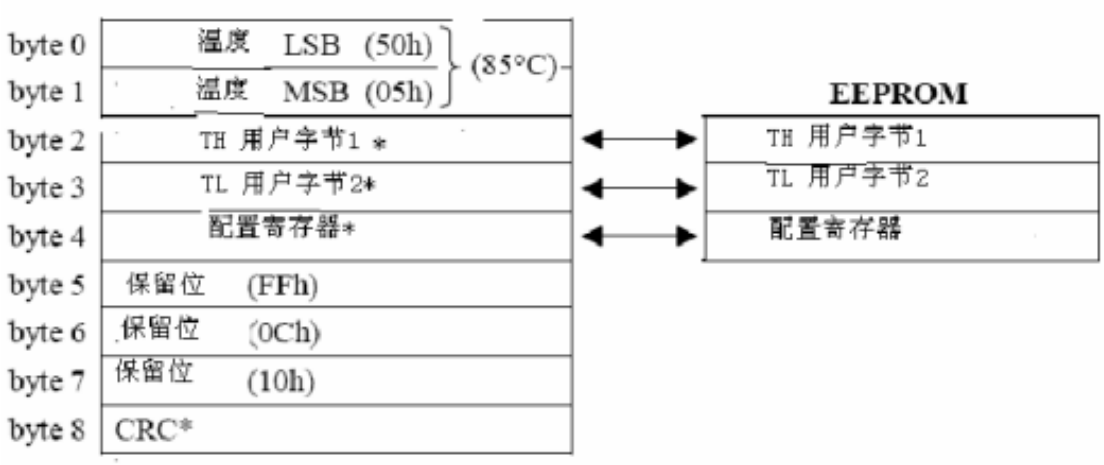
DS18B20 有三只引脚，VCC，DQ，和 VDD。



而 HJ-2G 板子上，采用了外部供电的链接方式，而总线必须链接上拉电阻。这一目的告诉我们，一线总线在空置状态时，都是一直处于高电平。



DS18B20 的内部有 64 位的 ROM 单元，和 9 字节的暂存器单元。64 位 ROM 包含了，DS18B20 唯一的序列号（唯一的名字）。



以上是内部 9 个字节的暂存单元（包括 EEPROM）。
字节 0~1 是转换好的温度。
字节 2~3 是用户用来设置最高报警和最低报警值。这个可以用软件来实现。
字节 4 是用来配置转换精度，9~12 位。
字节 5~8 就不用看了。

14.3 字节 0~1：转换好的温度

温度寄存器格式 图 2

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
LS Byte	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
MS Byte	S	S	S	S	S	2^6	2^5	2^4

DS18B20 的温度操作是使用 16 位，也就是说分辨率是 0.0625。BIT15~BIT11 是符号位，为了就是表示转换的值是正数还是负数。看看数据手册给出的例子吧。

温度/数据关系 表 2

温度 ℃	数据输出（二进制）	数据输出（十六进制）
+125	0000 0111 1101 0000	07D0h
+85	0000 0101 0101 0000	0550h
+25. 0625	0000 0001 1001 0001	0191h
+10. 125	0000 0000 1010 0010	00A2h
+0. 5	0000 0000 0000 1000	0008h
0	0000 0000 0000 0000	0000h
-0. 5	1111 1111 1111 1000	FFF8h
-10. 125	1111 1111 0101 1110	FF5Eh
-25. 0625	1111 1110 0110 1111	FE6Fh
-55	1111 1100 1001 0000	FC90h

要求出正数的十进制值，必须将读取到的 LSB 字节，MSB 字节进行整合处理，然后乘以 0.0625 即可。

Eg：假设从，字节 0 读取到 0xD0 赋值于 Temp1，而字节 1 读取到 0x07 赋值于 Temp2，然后求出十进制值。

```
unsigned int Temp1,Temp2,Temperature;

Temp1=0xD0;          //低八位
Temp2=0x07;          //高八位

Temperature = ((Temp2<<8 ) | Temp1 ) * 0.0625;
//又或者
Temperature = (Temp1 + Temp2 *256) * 0.0625;          //Temperature=125
```

在这里我们遇见了一个问题，就是如何求出负数的值呢？很遗憾的，单片机不像人脑那样会心算，我们必须判断 BIT11~15 是否是 1，然后人为置一负数标志。

Eg. 假设从，字节 0 读取到 0x90 赋值于 Temp1，而字节 1 读取到 0xFC 赋值于 Temp2，然后求出该值是不是负数，和转换成十进制值。

```
unsigned int Temp1,Temp2,Temperature;
unsigned char Minus_Flag=0;

Temp1=0x90;          //低八位
Temp2=0xFC;          //高八位

//Temperature = (Temp1 + Temp2 *256) * 0.0625;        //Temperature=64656
                                                      //很明显不是我们想要的答案
if(Temp2&0xFC)      //判断符号位是否为 1
{
    Minus_Flag=1;                                          //负数标志置一
    Temperature = ((Temp2<<8 ) | Temp1 )                       //高八位第八位进行整合
    Temperature= ((~Temperature)+1);                       //求反，补一
    Temperature*= 0.0625;                                  //求出十进制
}                                                          //Temperature=55;
else
{
    Minus_Flag=0;
    Temperature = ((Temp2<<8 ) | Temp1 ) * 0.0625;
}
```

那个人为的负数标志，是真的很有用处的。这个要看你你自己的想象力了,如何去利用它。

继续继续，以上我们是求出没有小数点的正数。如果我要求出小数点的值的话，那么我应该这样做。

Eg：假设从，字节 0 读取到 0xA2 赋值于 Temp1，而字节 1 读取到 0x00 赋值于 Temp2，然后求出十进制值，要求连同小数点也求出。

```
unsigned int Temp1,Temp2,Temperature;

Temp1=0x90;          //低八位
Temp2=0xFC;          //高八位
                                                    //实际值为 10.125
//Temperature = ((Temp2<<8 ) | Temp1 ) * 0.0625;          //10，无小数点
Temperature = ((Temp2<<8 ) | Temp1 ) * (0.0625 * 10) ;    //101 ，一位小数点
//Temperature = ((Temp2<<8 ) | Temp1 ) * (0.0625 * 100) ;  //1012，二位小数点
```

如以上的例题，我们可以先将 0.0625 乘以 10，然后再乘以整合后的 Temperature 变量，就可以求出后面一个小数点的值（求出更多的小数点，方法都是以此类推 ）。得出的结果是 101，然后再利用简单的算法，求出每一位的值。

```
unsinged char Ten,One,Dot1

Ten=Temperature/100;          //1
One=Temperature%100/10;       //0
Dot1=%10;                     //1
```

求出负数的思路也一样，只不过多出人为置一负数标志，求反补一的动作而已。自己发挥想象力吧。

14.4 字节 2~3：TH 和 TL 配置

TH 与 TL 就是所谓的温度最高界限，和温度最低界限的配置。其实这些可以使用软件来试验，所以就无视了。

14.5 字节 4：配置寄存器

配置寄存器 图 8

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	R1	R0	1	1	1	1	1

BIT7 出厂的时候就已经设置为 0，用户不建议去更改。而 R1 与 R0 位组合了四个不同的转换精度，00 为 9 位转换精度而转换时间是 93.75ms，01 为 10 位转换精度而转换时间是 187.5ms，10 为 11 位转换精度而转换时间是 375ms，11 为 12 位转换精度而转换时间是 750ms（默认）。该寄存器还是留默认的好，毕竟转换精度表示了转换的质量。

14.6 字节 5~7，8：保留位，CRC

无视，无视吧。

14.7 单片机访问 DS18B20

DS18B20 一般都是充当从机的角色，而单片机就是主机。单片机通过一线总线访问 DS18B20 的话，需要经过以下几个步骤：

1.DS18B20 复位。
2.执行 ROM 指令。
3.执行 DS18B20 功能指令（RAM 指令）。

补充一下。一般上我们都是使用单点，也就是说单线总线上仅有一个 DS18B20 存在而已。所以我们无需刻意读取 ROM 里边的序列号来，然后匹配 那个 DS18B20？而是更直接的，跳过 ROM 指令，然后直接执行 DS18B20 功能指令。

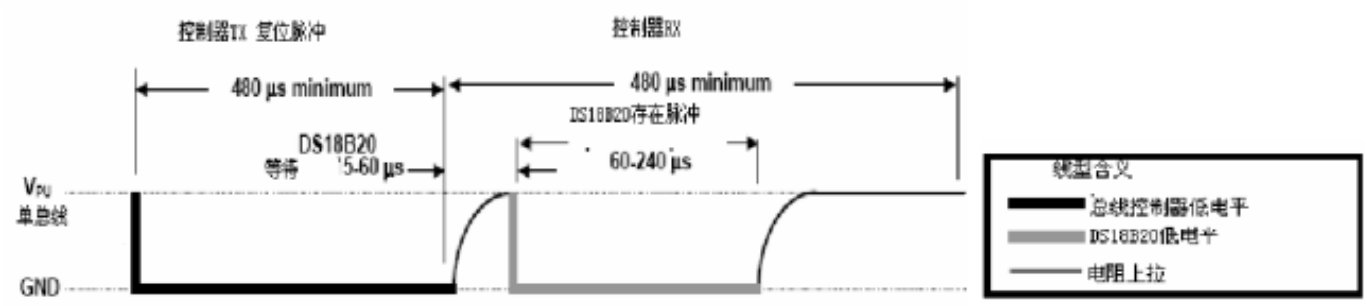
DS18B20 复位，在某种意义上就是一次访问 DS18B20 的开始，或者可说成是开始信号。

ROM 指令，也就是访问，搜索，匹配，DS18B20 个别的 64 位序列号的动作。在单点情况下，可以直接跳过 ROM 指令。而跳过 ROM 指令的字节是 0xCC。

DS18B20 功能指令有很多种，我就不一一的介绍了，数据手册里有更详细的介绍。这里仅列出比较常用的几个 DS18B20 功能指令。

0x44：开始转换温度。转换好的温度会储存到暂存器字节 0 和 1。
0xEE：读暂存指令。读暂存指令，会从暂存器 0 到 9，一个一个字节读取，如果要停止的话，必须写下 DS18B20 复位。

14.8 DS18B20 复位



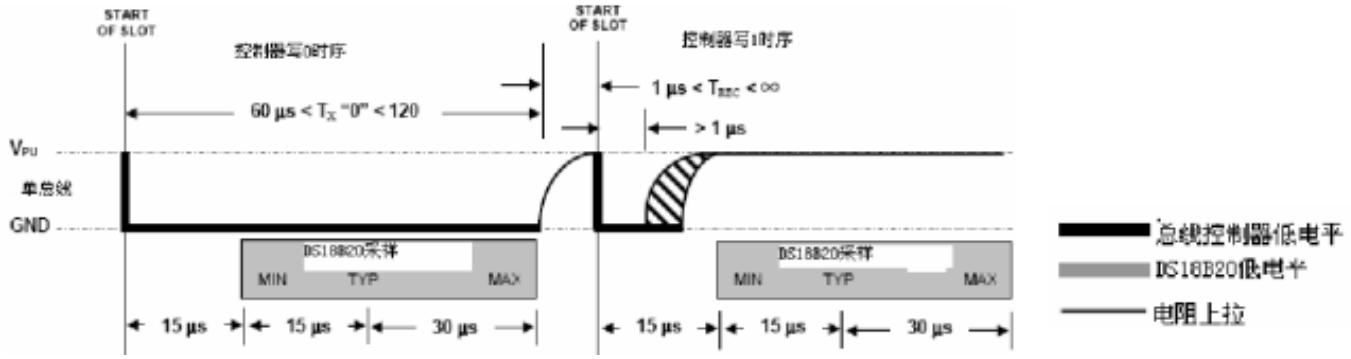
DS18B20 的复位时序如下：

1.单片机拉低总线 480us~950us, 然后释放总线（拉高电平）。
2.这时 DS18B20 会拉低信号，大约 60~240us 表示应答。
3.DS18B20 拉低电平的 60~240us 之间，单片机读取总线的电平，如果是低电平，那么表示复位成功。
4.DS18B20 拉低电平 60~240us 之后，会释放总线。

C 语言代码：

```
//DS1302 复位函数
void DS1302_Reset()
{
    DDRA|=BIT(DQ);          //DQ 为输出状态
    PORTA&=~BIT(DQ);        //输出低电平
    Delay_1us(500);         //延迟 500 微妙
    PORTA|=BIT(DQ);         //示范总线
    Delay_1us(60);          //延迟 60 微妙
    DDRA&=~BIT(DQ);         //DQ 位输出状态
    while(PINA&BIT(DQ));    //等待从机 DS18B20 应答（低电平有效）
    while(!(PINA&BIT(DQ))); //等待从机 DS18B20 释放总线
}
```

14.9 DS18B20 读写逻辑 0 与 1



DS18B20 写逻辑 0 的步骤如下：

1.单片机拉低电平大约 10~15us,。

2.单片机持续拉低电平大约 20~45us 的时间。

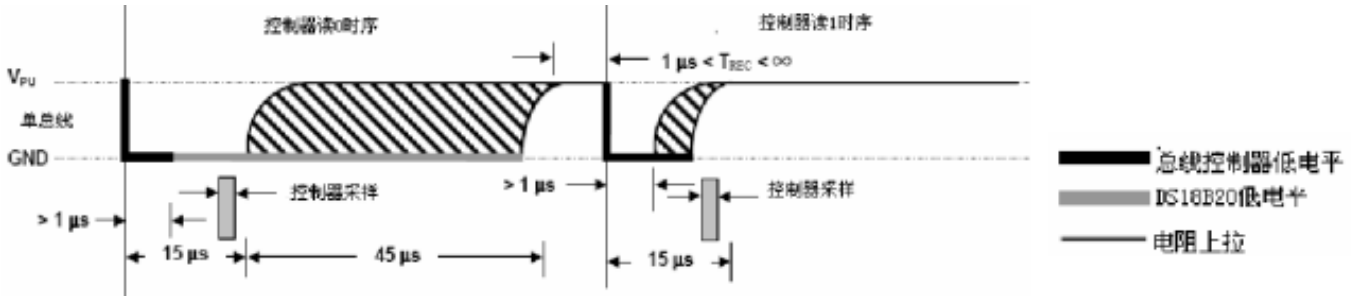
3.释放总线

DS18B20 写逻辑 1 的步骤如下：

1.单片机拉低电平大约 10~15us,。

2.单片机拉高电平大约 20~45us 的时间。

3.释放总线



DS18B20 读逻辑 0 的步骤如下：

1.在读取的时候单片机拉低电平大约 1us

2.单片机释放总线，然后读取总线电平。

3.这时候 DS18B20 会拉低电平。

4.读取电平过后，延迟大约 40~45 微妙

DS18B20 读逻辑 1 的步骤如下：

1.在读取的时候单片机拉低电平大约 1us

2.单片机释放总线，然后读取总线电平。

3.这时候 DS18B20 会拉高电平。

4.读取电平过后，延迟大约 40~45 微妙

如果要读或者写一个字节，就要重复以上的步骤八次。如以下的 C 代码，使用 for 循环，和数据变量的左移和或运算，实现一个字节读与写。

```
//DS18B20 写字节函数
void DS1302_Write(unsigned char Data)
{
    unsigned char i;
    DDRA|=BIT(DQ);          //DQ 为输出
    for(i=0;i<8;i++)
    {
            PORTA&=~BIT(DQ);            //拉低总线
            Delay_1us(10);              //延迟 10 微妙（最大 15 微妙）

            if(Data&0x01) PORTA|=BIT(DQ);
            else PORTA&=~BIT(DQ);
            Delay_1us(40);              //延迟 40 微妙（最大 45 微妙）

            PORTA|=BIT(DQ);             //释放总线
            Delay_1us(1);               //稍微延迟
            Data>>=1;
    }
}
```

```
//DS18B20 读字节函数
unsigned char DS1302_Read()
{
    unsigned char i,Temp;

    for(i=0;i<8;i++)
    {
            Temp>>=1;                       //数据右移

            DDRA|=BIT(DQ);                  //DQ 为输出状态
            PORTA&=~BIT(DQ);                //拉低总线，启动输入
            PORTA|=BIT(DQ);                 //释放总线
            DDRA&=~BIT(DQ);                 //DQ 为输入状态

            if(PINA&BIT(DQ)) Temp|=0x80;
            Delay_1us(45);                  //延迟 45 微妙（最大 45 微妙）
    }
    return Temp;
}
```

就是这么建档而已，不过这里有一个注意点，就是 Delay_1us(); 函数延迟的时间，必须模拟非常准确，因为单线总线对时序的要求敏感点。

14.10 简单归纳

实验开始之前，简单的归纳一些重点。单线总线高电平为闲置状态。单片机访问 DS18B20 必须遵守，DS18B20 复位-->执行 ROM 指令-->执行 DS18B20 功能指令。而在单点上，可以直接跳过 ROM 指令。DS18B20 的转换精度默认为 12 位，而分辨率是 0.0625。

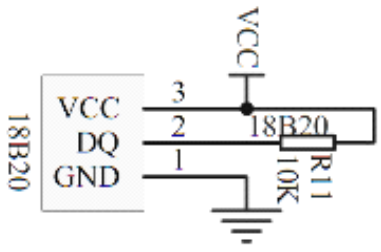
DS18B20 温度读取函数参考步骤：

DS18B20 开始转换：
1.DS18B20 复位。
2.写入跳过 ROM 的字节命令，0xCC。
3.写入开始转换的功能命令，0x44。
4.延迟大约 750~900 毫秒

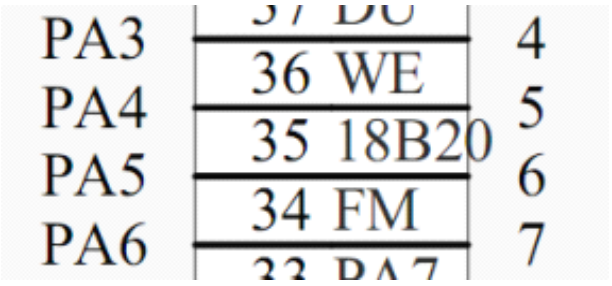
DS18B20 读暂存数据：
1.DS18B20 复位。
2.写入跳过 ROM 的字节命令，0xCC。
3.写入读暂存的功能命令，0xee。
4.读入第 0 个字节 LS Byte，转换结果的低八位。
5.读入第 1 个字节 MS Byte，转换结果的高八位。
6.DS18B20 复位，表示读取暂存结束。

数据求出十进制：
1.整合 LS Byte 和 MS Byte 的数据
2.判断是否为正负数（可选）
3.求得十进制值。正数乘以 0.0625，一位小数点乘以 0.625，二位小数点乘以 6.25。
4.十进制的“个位”求出。

14.11 实验：利用 DS18B20 实现单点温度测量，结果输出在数码管。



DSl8B20 接口



ATMega 16 对应引脚

实验的要求是以 DS18B20 默认的配置，亦即 12 位的转换精度。然而输出的结果为两个小数点 xx.xx。HJ-2G 板子上设计得 DS18B20 的接口和典型，没有什么特别需要注意的。而 DS18B20 DQ 引脚对应的链接是 PA5。

AVR 单片机学习笔记--基于慧净 HJ-2G AVR 开发板

源码：

==========

```
//1400-DS18B20.c
//简单的驱动程式
//akuei2 08-01-10

#include "iom16v.h"
#include "macros.h"
#include "LED7.h"

#define DQ PA5

//微妙级延迟函数
void Delay_1us(unsigned int x)
{
        unsigned int i;
        x=x*5/4;
        for( i=0;i<x;i++);
}

//DS1302 复位函数
void DS1302_Reset()
{
    DDRA|=BIT(DQ);          //DQ 为输出状态
    PORTA&=~BIT(DQ);        //输出低电平
    Delay_1us(500);         //延迟 500 微妙
    PORTA|=BIT(DQ);         //示范总线
    Delay_1us(60);          //延迟 60 微妙
    DDRA&=~BIT(DQ);         //DQ 位输出状态
    while(PINA&BIT(DQ));    //等待从机 DS18B20 应答（低电平有效）
    while(!(PINA&BIT(DQ)));  //等待从机 DS18B20 释放总线
}

//DS1302 写字节函数
void DS1302_Write(unsigned char Data)
{
    unsigned char i;
    DDRA|=BIT(DQ);      //DQ 为输出
    for(i=0;i<8;i++)
    {
            PORTA&=~BIT(DQ);            //拉低总线
            Delay_1us(10);              //延迟 10 微妙（最大 15 微妙）

            if(Data&0x01) PORTA|=BIT(DQ);
            else PORTA&=~BIT(DQ);
            Delay_1us(40);              //延迟 40 微妙（最大 45 微妙）
```

```
            PORTA|=BIT(DQ);                 //释放总线
            Delay_1us(1);                   //稍微延迟
            Data>>=1;
        }
}

//DS1302 读字节函数
unsigned char DS1302_Read()
{
    unsigned char i,Temp;

    for(i=0;i<8;i++)
    {
            Temp>>=1;                       //数据右移

            DDRA|=BIT(DQ);                  //DQ 为输出状态
            PORTA&=~BIT(DQ);                //拉低总线，启动输入
            PORTA|=BIT(DQ);                 //释放总线
            DDRA&=~BIT(DQ);                 //DQ 为输入状态

            if(PINA&BIT(DQ)) Temp|=0x80;
            Delay_1us(45);                  //延迟 45 微妙（最大 45 微妙）
    }

    return Temp;
}

//读温度函数
unsigned int Read_Temperature()
{
    unsigned int Temp1,Temp2;

    DS1302_Reset();             //DS1302 复位
    DS1302_Write(0xCC);         //跳过 ROM
    DS1302_Write(0x44);         //温度转换

    DS1302_Reset();             //DS1302 复位
    DS1302_Write(0xCC);         //跳过 ROM
    DS1302_Write(0xbe);         //读取 RAM

    Temp1=DS1302_Read();        //读低八位，LS Byte, RAM0
    Temp2=DS1302_Read();        //读高八位，MS Byte, RAM1
    DS1302_Reset();             //DS1302 复位，表示读取结束

    return (((Temp2<<8)|Temp1)*6.25); //0.0625=xx, 0.625=xx.x, 6.25=xx.xx
```

```
}

void main()
{
	unsigned int Temp;
	LED7_Init();			//初始化数码管引脚
	while(1)
	{
			Temp=Read_Temperature();		//调用读取温度函数
			Number_Show(Temp);			//显示温度
			Delay_1us(100);				//稍微延迟
	}

}
```

LED7.h 的头文件

==========

```
//LED7.H
//数码管显示

//数组声明并定义在存储数据区 code
//0~9

#pragma data:code
unsigned char const Number[]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x00,};
unsigned char const Number_Dot[]={0xbf,0x86,0xdb,0xcf,0xe6,0xed,0xfd,0x87,0xff,0xef,0x00};

//延迟函数
void Delay(unsigned long x)
{
    while(x--);
}

//数码管显示函数
void Number_Show(unsigned int Num)
{
    unsigned char Ten,One,Dot1,Dot2;
    Ten=Num/1000;		//取十位
    One=Num%1000/100;	//取个位
     Dot1=Num%100/10;	//取点数位
     Dot2=Num%10;		//取点数位

    //显示十位
```

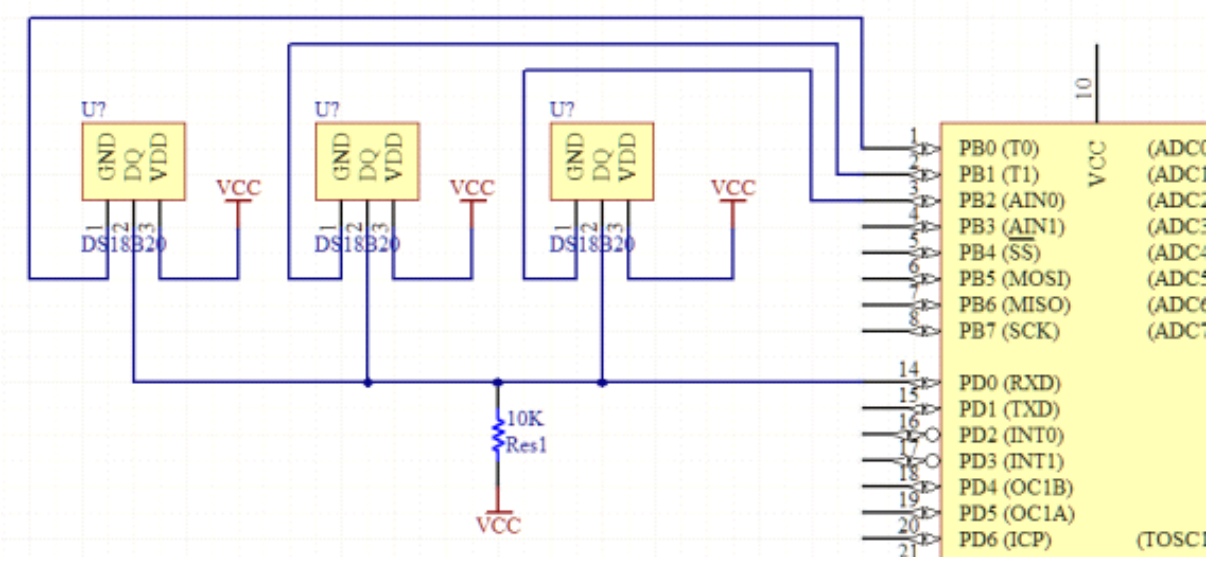
```
PORTB=Number[Ten]; //送模码
PORTA|=BIT(PA3);       //PA3 高电平
PORTA&=~BIT(PA3);      //PA3 低电平

PORTB=~BIT(0);         //送位选
PORTA|=BIT(PA4);       //PA3 高电平
PORTA&=~BIT(PA4);      //PA3 低电平
Delay(600);            //稍微延迟

//显示个位
PORTB=Number_Dot[One];   //送模码
PORTA|=BIT(PA3);       //PA3 高电平
PORTA&=~BIT(PA3);      //PA3 低电平

PORTB=~BIT(1);         //送位选
PORTA|=BIT(PA4);       //PA3 高电平
PORTA&=~BIT(PA4);      //PA3 低电平
Delay(600);            //稍微延迟

//显示点数位 1
PORTB=Number[Dot1];    //送模码
PORTA|=BIT(PA3);       //PA3 高电平
PORTA&=~BIT(PA3);      //PA3 低电平

PORTB=~BIT(2);         //送位选
PORTA|=BIT(PA4);       //PA3 高电平
PORTA&=~BIT(PA4);    //PA3 低电平
Delay(600);            //稍微延迟

 //显示点数位 2
PORTB=Number[Dot2];    //送模码
PORTA|=BIT(PA3);       //PA3 高电平
PORTA&=~BIT(PA3);      //PA3 低电平

PORTB=~BIT(3);         //送位选
PORTA|=BIT(PA4);       //PA3 高电平
PORTA&=~BIT(PA4);      //PA3 低电平
Delay(600);            //稍微延迟

 //显示`
PORTB=0x63;            //送模码
PORTA|=BIT(PA3);       //PA3 高电平
PORTA&=~BIT(PA3);    //PA3 低电平

PORTB=~BIT(4);         //送位选
PORTA|=BIT(PA4);       //PA3 高电平
```

```
    PORTA&=~BIT(PA4);    //PA3 低电平
    Delay(600);          //稍微延迟

    //显示 C
    PORTB=0x39;          //送模码
    PORTA|=BIT(PA3);     //PA3 高电平
    PORTA&=~BIT(PA3);    //PA3 低电平

    PORTB=~BIT(5);       //送位选
    PORTA|=BIT(PA4);     //PA3 高电平
    PORTA&=~BIT(PA4);    //PA3 低电平
    Delay(1000);         //稍微延迟

}

//IO 初始化函数
void LED7_Init()
{
    DDRA|=BIT(PA3);      //PA3 状态为输出
    DDRA|=BIT(PA4);      //PA4 状态为输出
    DDRB|=0xff;          //PB 状态为输出
}
```

==========

以上的程式只有一个注意点就是：DS18B20 的转启动换频率不要超过 750ms。其他的没有什么需要特别注意了。

14.12 一个多点测温的假想



以上是多点测温的一个假想，就是利用 GND 作于片选的角色。该方法有一个好处就是可以省去猥琐

的 ROM 指令，但是最为代价需要牺牲 IO 口，而且还控制好每一个 DS18B20 的执行次序。