温度传感器DS18B20原理，附STM32例程代码

SugarlesS

STM32设计温湿度采集器视频

DS18B20是一款常用的高精度的单总线数字温度测量芯片。具有体积小，硬件开销低，抗干扰能力强，精度高的特点。

DS18B20原理

传感器参数

测温范围为-55℃到+125℃，在-10℃到+85℃范围内误差为±0.4°

返回16位二进制温度数值

主机和从机通信使用单总线，即使用单线进行数据的发送和接收

在使用中不需要任何外围元件，独立芯片即可完成工作

掉电保护功能 DS18B20 内部含有 EEPROM ，通过配置寄存器可以设定数字转换精度和报警温度，在系统掉电以后，它仍可保存分辨率及报警温度的设定值

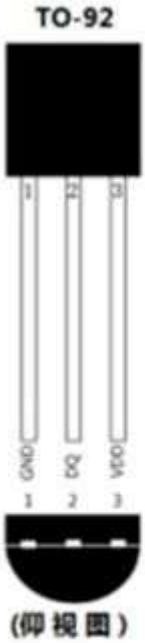
每个DS18B20都有独立唯一的64位-ID，此特性决定了它可以将任意多的DS18b20挂载到一根总线上，通过ROM搜索读取相应DS18B20的温度值

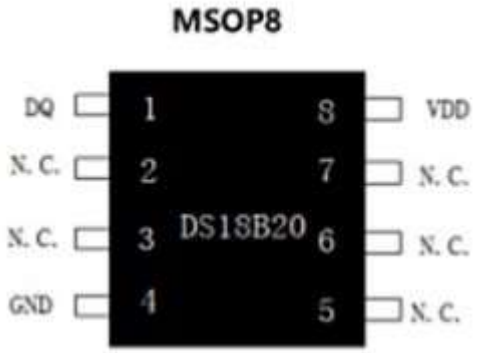
宽电压供电，电压2.5V~5.5V

DS18B20返回的16位二进制数代表此刻探测的温度值，其高五位代表正负。如果高五位全部为1，则代表返回的温度值为负值。如果高五位全部为0，则代表返回的温度值为正值。后面的11位数据代表温度的绝对值，将其转换为十进制数值之后，再乘以0.0625即可获得此时的温度值

传感器引脚及原理图

DS18B20传感器的引脚及封装图如下:





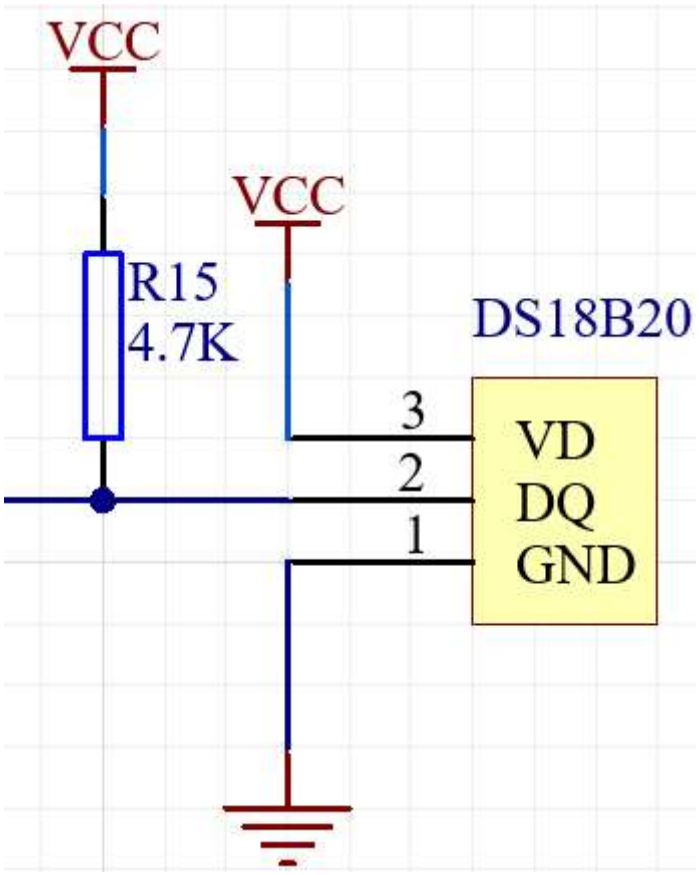
引脚名称	功能
GND	地线
DQ	单总线通信接口，寄生模式时供电端口
VDD	电源线（2 线通信时接地，以保证芯片内正确识别 VDD 状态））

DS18B20一共有三个引脚，分别是：

GND：电源地线

DQ：数字信号输入 / 输出端

VDD：外接供电电源输入端

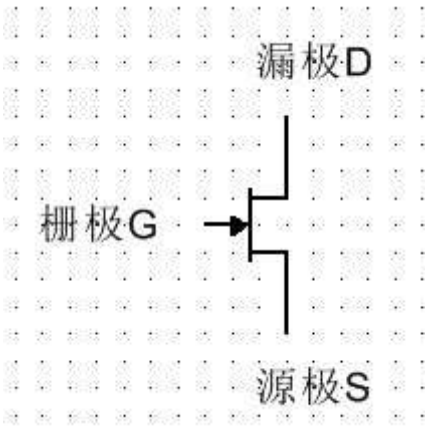


单个DS18B20接线方式：VDD接到电源，DQ接单片机引脚，同时外加上拉电阻，GND接地。

注意这个上拉电阻是必须的，就是DQ引脚必须要一个上拉电阻。

DS18B20上拉电阻

首先来看一下什么是场效应管（MOSFET），如下图。



场效应管是电压控制型元器件，只要对栅极施加一定电压，DS就会导通。

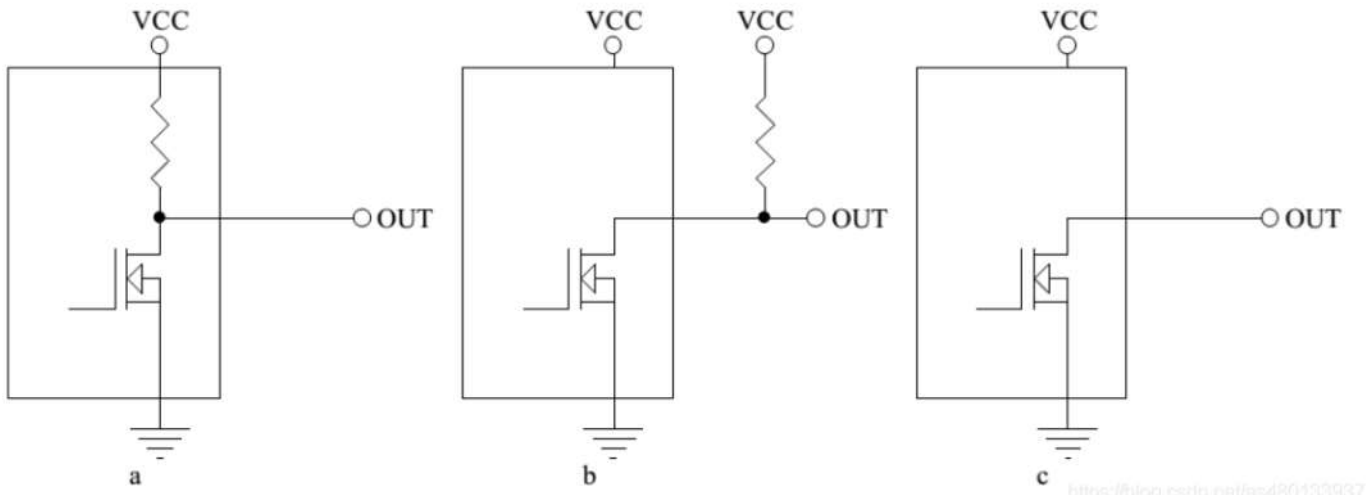
漏极开路：MOS管的栅极G和输入连接，源极S接公共端，漏极D悬空（开路）什么也没有接，直接输出，这时只能输出低电平和高阻态，不能输出高电平。

那么这个时候会出现三种情况：

下图a为正常输出（内有上拉电阻）：场效应管导通时，输出低电位输出低电位，截止时输出高电位

下图b为漏极开路输出，外接上拉电阻：场效应管导通时，驱动电流是从外部的VCC流经电阻通过MOSFET到GND，输出低电位，截止时输出高电位

下图c为漏极开路输出，无外接上拉电阻：场效应管导通时输出低电位，截止呈高阻态(断开)



总结一下：

开漏输出只能输出低电平，不能输出高电平。漏极开路输出高电平时必须在输出端与正电源（VCC）间外接一个上拉电阻。否则只能输出高阻态。

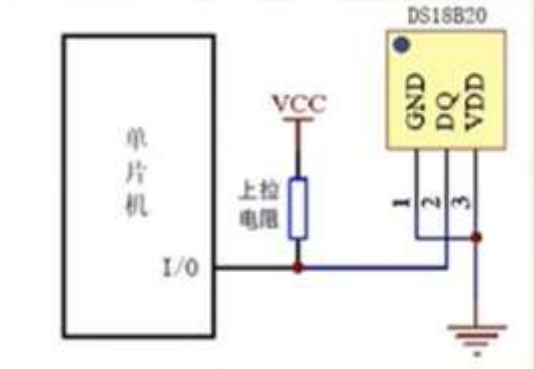
DS18B20 是单线通信，即接收和发送都是这个通信脚进行的。其接收数据时为高电阻输入，其发送数据时是开漏输出，本身不具有输出高电平的能力，即输出0时通过MOS下拉为低电平，而输出1时，则为高阻，需要外接上拉电阻将其拉为高电平。因此，需要外接上拉电阻，否则无法输出1。

外接上拉电阻阻值：

DS18B20的工作电流约为1mA，VCC一般为5V，则电阻R=5V/1mA=5KΩ，所以正常选择4.7K电阻，或者相近的电阻值。

DS18B20寄生电源

DS18B20的另一个特点是不需要再外部供电下即可工作。当总线高电平时能量由单线上拉电阻经过DQ引脚获得。高电平同时充电一个内部电容，当总线低电平时由此电容供应能量。这种供电方法被称为“寄生电源”。另外一种选择是DSI8B20由接在VDD的外部电源供电。



DS18B20内部构成

主要由以下3部分组成：

64 位ROM

高速暂存器

存储器

64位ROM存储独有的序列号，ROM中的64位序列号是出厂前被光刻好的，它可以看作是该DS18B20的地址序列码，每个DS18B20的64位序列号均不相同。这样就可以实现一根总线上挂接多个DS18B20的目的。

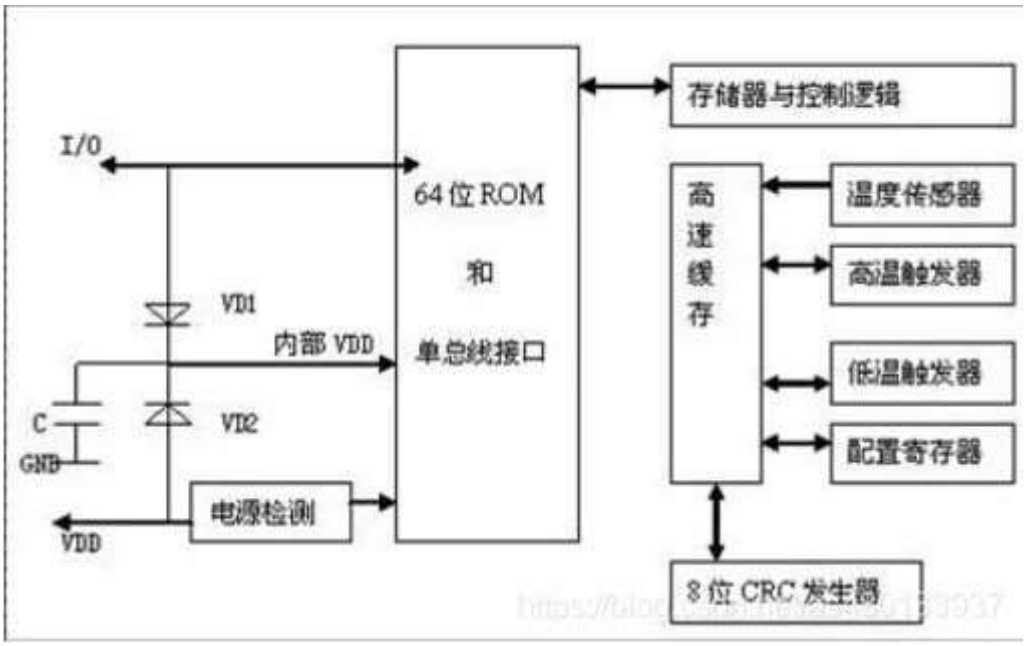
高速暂存器包含：

温度传感器

一个字节的温度上限和温度下限报警触发器(TH和TL)

配置寄存器允许用户设定9位，10位，11位和12位的温度分辨率，分别对应着温度的分辨率为：0.5°C，0.25°C，0.125°C，0.0625°C，默认为12位分辨率

存储器：由一个高速的RAM和一个可擦除的EEPROM组成，EEPROM存储高温和低温触发器(TH和TL)以及配置寄存器的值，(就是存储低温和高温报警值以及温度分辨率)



DS18B20温度读取与计算

DS18B20采用16位补码的形式来存储温度数据，温度是摄氏度。当温度转换命令发布后，经转换所得的温度值以二字节补码形式存放在高速暂存存储器的第0和第1个字节。

高字节的五个S为符号位，温度为正值时S=1,温度为负值时S=0。

剩下的11位为温度数据位，对于12位分辨率，所有位全部有效，对于11位分辨率，位0(bit0)无定义，对于10位分辨率，位0和位1无定义,对于9位分辨率，位0，位1，和位2无定义。

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
低字节	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
高字节	S	S	S	S	S	2^6	2^5	2^4

对应的温度计算：

当五个符号位S=0时，温度为正值，直接将后面的11位二进制转换为十进制，再乘以0.0625(12位分辨率)，就可以得到温度值。

当五个符号位S=1时，温度为负值，先将后面的11位二进制补码变为原码(符号位不变，数值位取反后加1)，再计算十进制值。再乘以0.0625(12位分辨率)，就可以得到温度值。

举两个例子：

数字输出07D0(00000111 11010000)，转换成10进制是2000，对应摄氏度：
0.0625x2000=125°C

数字输出为 FC90，首先取反，然后+1，转换成原码为：11111011 01101111，数值位转换成10进制是870，对应摄氏度：-0.0625x870=-55°C

温度对应表如下：

温度/℃	二进制表示	十六进制表示
+125	00000111 11010000	07D0H
+25.0625	00000001 10010001	0191H
+10.125	00000000 10100010	00A2H
+0.5	00000000 00001000	0008H
0	00000000 00000000	0000H
-0.5	11111111 11111000	FFF8H
-10.125	11111111 01011110	FF5EH
-25.0625	11111110 01101111	FE6FH
-55	11111100 10010000	FC90H

上述例子，用C语言来实现的代码，如下：

```
unsigned int Temp1,Temp2,Temperature; //Temp1低八位，Temp2高八位

unsigned char Minus Flag=0; //负温度标志位

if（Temp2&0xFC）//判断符号位是否为1

{

Minus Flag=l; //负温度标志位置1

Temperature=((Temp2<<8)|Temp1); //高八位第八位进行整合

Temperature=((Temperature)+1); //讲补码转换为原码，求反，补1

Temperature*=0.0625;//求出十进制

}

else //温度为正值

{

Minus Flag=0; //负温度标志位置0

Temperature =((Temp2<<8) |Temp1)*0.0625;

}
```

DS18B20工作步骤

DS18B20的工作步骤可以分为三步：

初始化DS18B20

执行ROM指令

执行DS18B20功能指令

其中第二步执行ROM指令，也就是访问每个DS18B20，搜索64位序列号，读取匹配的序列号值，然后匹配对应的DS18B20，如果我们仅仅使用单个DS18B20，可以直接跳过ROM指令。而跳过ROM指令的字节是0xCC。

初始化DS18B20

任何器件想要使用，首先就是需要初始化，对于DS18B20单总线设备，首先初始化单总线为高电平，然后总线开始也需要检测这条总线上是否存在DS18B20这个器件。如果这条总线上存在DS18B20，总线会根据时序要求返回一个低电平脉冲，如果不存在的话，也就不会返回脉冲，即总线保持为高电平。

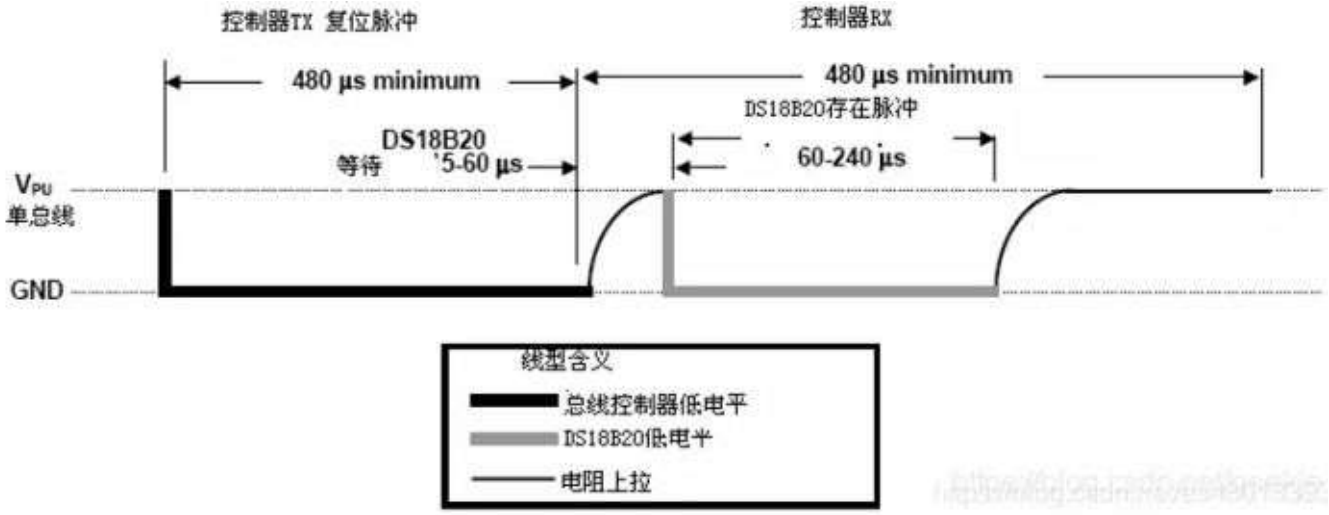
初始化具体时序步骤如下:

单片机拉低总线至少480us，产生复位脉冲，然后释放总线（拉高电平）

这时DS8B20检测到请求之后，会拉低信号，大约60~240us表示应答

DS8B20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，那么表示初始化成功

DS18B20拉低电平60~240us之后，会释放总线



DS18B20的初始化代码如下:

```
/*****初始化DS18B20*****/

unsigned int Init_DS18B20(void)

{

unsigned int x=0;

DQ = 1; //DQ复位

delay(4); //稍做延时

DQ = 0; //单片机将DQ拉低

delay(60); //精确延时，大于480us

DQ = 1; //拉高总线

delay(8);

x = DQ; //稍做延时后，如果x=0则初始化成功，x=1则初始化失败

delay(4);

return x;
```

}

写时序

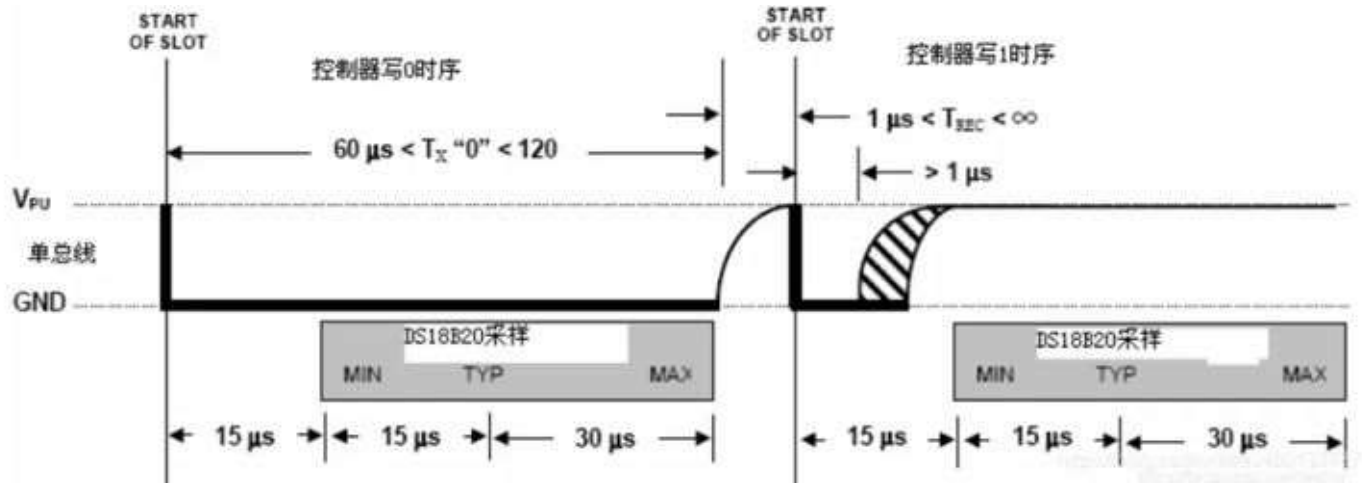
总线控制器通过控制单总线高低电平持续时间从而把逻辑1或0写DS18B20中。每次只传输1位数据。

单片机想要给DS18B20写入一个0时，需要将单片机引脚拉低，保持低电平时间要在60~120us之间，然后释放总线。

单片机想要给DS18B20写入一个1时，需要将单片机引脚拉低，拉低时间需要大于1us，然后在15us内拉高总线。

在写时序起始后15μs到60μs期间，DS18B20处于采样单总线电平状态。如果在此期间总线为高电平，则向DS18B20写入1;如果总线为低电平，则向DSI8B20写入0。

注意：2次写周期之间至少间隔1us。



DS18B20写时序的代码如下:

*/*****写一个字节*****/*

void WriteOneChar(unsigned char dat)

{

```
unsigned char i=0;

for (i=8; i>0; i--)

{

DQ = 0;

DQ = dat&0x01; //与1按位与运算，dat最低位为1时DQ总线为1，dat最低位为0时DQ总线为0

delay(4);

DQ = 1;

dat>>=1;

}

delay(4);

}
```

采用多个DS18B20时，需要写ROM指令来控制总线上的某个DS18B20。如果是单个DS18B20，直接写跳过ROM指令0xCC即可。DS18B20写入ROM功能指令如下表：

指令名称	指令代码	指令功能
读 ROM	33H	读 DS18B20ROM 中的编码（即读 64 位地址）
ROM 匹配（符合 ROM）	55H	发出此命令之后，接着发出 64 位 ROM 编码，访问单总线上与编码相对应 DS18B20 使之作出响应，为下一步对该 DS18B20 的读写作准备
搜索 ROM	0F0H	用于确定挂接在同一总线上 DS18B20 的个数和识别 64 位 ROM 地址，为操作各器件作好准备
跳过 ROM	0CCH	忽略 64 位 ROM 地址，直接向 DS18B20 发温度变换命令，适用于单片机工作
警报搜索	0ECH	该指令执行后，只有温度超过设定值上限或下限的片子才做出响应

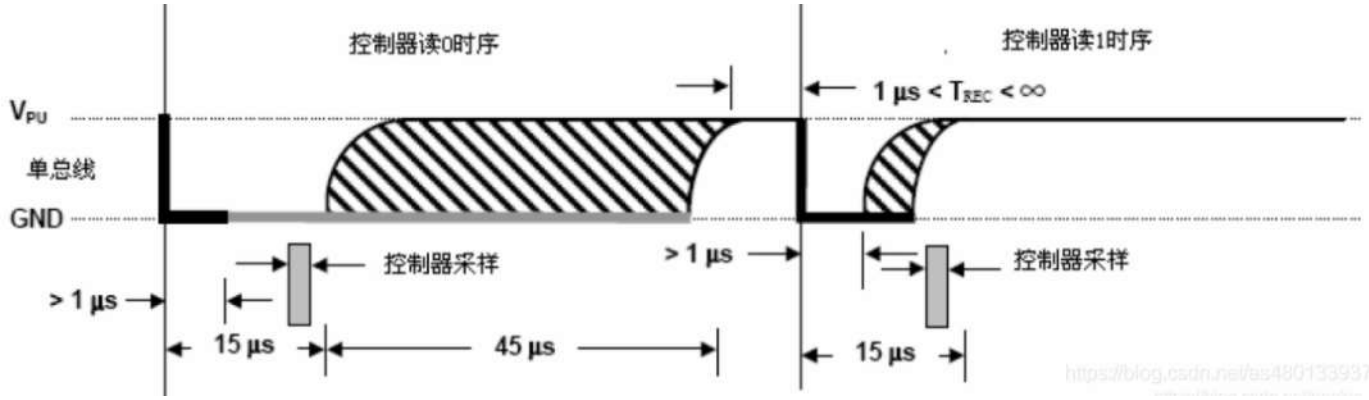
DS18B20的一些RAM功能指令如下表。其中常用的是温度转换指令，开启温度读取转换，读取好的温度会存储在高速暂存器的第0个和第一个字节中。另一个常用的是读取温度指令，读取高速暂存器存储的数据。

指令名称	指令代码	指令功能
温度变换	44H	启动 DS18B20 进行温度转换，转换时间最长为 500ms（典型为 200ms），结果存入内部 9 字节 RAM 中
读暂存器	0BEH	读内部 RAM 中 9 字节的内容
写暂存器	4EH	发出向内部 RAM 的第 3，4 字节写上，下限温度数据命令，紧跟该命令之后，是传送两字节的数据
复制暂存器	48H	将 RAM 中第 3，4 字节的内容复制到 EEPROM 中
重调 EEPROM	0B8H	EEPROM 中的内容恢复到 RAM 中的第 3，4 字节
读供电方式	0B4H	读 DS18B20 的供电模式，寄生供电时 DS18B20 发送“0”，外接电源供电 DS18B20 发送“1”

读时序

读时隙由主机拉低总线电平至少1μs然后再释放总线，读取DS18B20发送过来的1或者0。

DS18B20在检测到总线被拉低1微秒后，便开始送出数据，若是要送出0就把总线拉为低电平直到读周期结束。若要送出1则释放总线为高电平。



注意：所有读时隙必须至少需要60us，且在两次独立的时隙之间至少需要1ps的恢复时间。

同时注意：主机只有在发送读暂存器命令(0xBE)或读电源类型命令(0xB4)后，立即生成读时隙指令，DS18B20才能向主机传送数据。也就是先发读取指令，再发送读时隙。

最后一点：写时序注意是先写命令的低字节，比如写入跳过ROM指令0xCC(11001100),写的顺序是“零、零、壹、壹、零、零、壹、壹”。

读时序时是先读低字节，在读高字节，也就是先读取高速暂存器的第0个字节(温度的低8位)，在读取高速暂存器的第1个字节(温度的高8位) 我们正常使用DS18B20读取温度读取两个温度字节即可。

STM32例程

DS18B20.c代码：

```
#include "ds18b20.h"

#include "delay.h"


//复位DS18B20

void DS18B20_Rst(void)

{

DS18B20_IO_OUT(); //SET PG11 OUTPUT

DS18B20_DQ_OUT=0; //拉低DQ

delay_us(750); //拉低750us

DS18B20_DQ_OUT=1; //DQ=1

delay_us(15); //15US

}

//等待DS18B20的回应

//返回1:未检测到DS18B20的存在

//返回0:存在

u8 DS18B20_Check(void)
```

```
{
u8 retry=0;
DS18B20_IO_IN(); //SET PG11 INPUT
while (DS18B20_DQ_IN&&retry<200)
{
retry++;
delay_us(1);
};
if(retry>=200)return 1;
else retry=0;
while (!DS18B20_DQ_IN&&retry<240)
{
retry++;
delay_us(1);
};
if(retry>=240)return 1;
return 0;
}
//从DS18B20读取一个位
//返回值：1/0
u8 DS18B20_Read_Bit(void)
{
```

```
u8 data;
DS18B20_IO_OUT(); //SET PG11 OUTPUT
DS18B20_DQ_OUT=0;
delay_us(2);
DS18B20_DQ_OUT=1;
DS18B20_IO_IN(); //SET PG11 INPUT
delay_us(12);
if(DS18B20_DQ_IN)data=1;
else data=0;
delay_us(50);
return data;
}
//从DS18B20读取一个字节
//返回值：读到的数据
u8 DS18B20_Read_Byte(void)
{
u8 i,j,dat;
dat=0;
for (i=1;i<=8;i++)
{
j=DS18B20_Read_Bit();
dat=(j<<7)|(dat>>1);
```

```
}

return dat;

}

//写一个字节到DS18B20

//dat：要写入的字节

void DS18B20_Write_Byte(u8 dat)

{

u8 j;

u8 testb;

DS18B20_IO_OUT(); //SET PG11 OUTPUT;

for (j=1;j<=8;j++)

{

testb=dat&0x01;

dat=dat>>1;

if (testb)

{

DS18B20_DQ_OUT=0; // Write 1

delay_us(2);

DS18B20_DQ_OUT=1;

delay_us(60);

}

else
```

```
{
DS18B20_DQ_OUT=0; // Write 0
delay_us(60);
DS18B20_DQ_OUT=1;
delay_us(2);
}
}
}
//开始温度转换
void DS18B20_Start(void)
{
DS18B20_Rst();
DS18B20_Check();
DS18B20_Write_Byte(0xcc); // skip rom
DS18B20_Write_Byte(0x44); // convert
}

//初始化DS18B20的IO口 DQ 同时检测DS的存在
//返回1:不存在
//返回0:存在
u8 DS18B20_Init(void)
```

```
{

GPIO_InitTypeDef GPIO_InitStructure;


RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOG, ENABLE); //使能PORTG口时钟


GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11; //PORTG.11 推挽输出

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;

GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;

GPIO_Init(GPIOG, &GPIO_InitStructure);



GPIO_SetBits(GPIOG,GPIO_Pin_11); //输出1



DS18B20_Rst();



return DS18B20_Check();

}

//从ds18b20得到温度值

//精度：0.1C

//返回值：温度值 （-550~1250）

short DS18B20_Get_Temp(void)

{
```

```
u8 temp;

u8 TL,TH;

short tem;

DS18B20_Start (); // ds1820 start convert

DS18B20_Rst();

DS18B20_Check();

DS18B20_Write_Byte(0xcc); // skip rom

DS18B20_Write_Byte(0xbe); // convert

TL=DS18B20_Read_Byte(); // LSB

TH=DS18B20_Read_Byte(); // MSB


if(TH>7)

{

TH=~TH;

TL=~TL;

temp=0; //温度为负

}else temp=1; //温度为正

tem=TH; //获得高八位

tem<<=8;

tem+=TL; //获得底八位

tem=(float)tem*0.625; //转换

if(temp)return tem; //返回温度值
```

```
else return -tem;

}
```

DS18B20.h代码：

```
#ifndef __DS18B20_H

#define __DS18B20_H

#include "sys.h"


//IO方向设置

#define DS18B20_IO_IN() {GPIOG->CRH&=0XFFFF0FFF;GPIOG->CRH|=8<<12;}

#define DS18B20_IO_OUT() {GPIOG->CRH&=0XFFFF0FFF;GPIOG->CRH|=3<<12;}

//IO操作函数

#define DS18B20_DQ_OUT PGout(11) //数据端口 PA0

#define DS18B20_DQ_IN PGin(11) //数据端口 PA0


u8 DS18B20_Init(void);//初始化DS18B20

short DS18B20_Get_Temp(void);//获取温度

void DS18B20_Start(void);//开始温度转换

void DS18B20_Write_Byte(u8 dat);//写入一个字节

u8 DS18B20_Read_Byte(void);//读出一个字节

u8 DS18B20_Read_Bit(void);//读出一个位
```

u8 DS18B20_Check(void);//检测是否存在DS18B20

void DS18B20_Rst(void);//复位DS18B20

#endif

发布于 2021-07-02 16:55