MY18E20温度传感芯片应用例程

V201903

1. **概述**

本文介绍了敏源MY18E20温度传感芯片的简单软件解决方案，用于微控制器与任意数量的MY18E20之间的基本单总线通信控制。

MY18E20的详细时序和操作信息可在数据手册中找到。

1. **硬件配置**

图1中的框图说明了使用多个单总线温度传感器时的硬件配置。单总线为所有设备提供通信访问通道。总线电源通过4.7kΩ上拉电阻从5V外接电源提供。理想情况下（不考虑线路干扰和损耗），几乎无限数量的单总线器件可以连接到总线，因为每个器件都有唯一的64位ROM代码标识符。



图1 主机微控制器接口

1. **接口时序**

主机微控制器通过使用“时隙”实现与MY18E20的通信，“时隙”允许数据通过单总线传输。每个通信周期都以微控制器的复位脉冲开始，接着是MY18E20的存在脉冲，如图2所示。

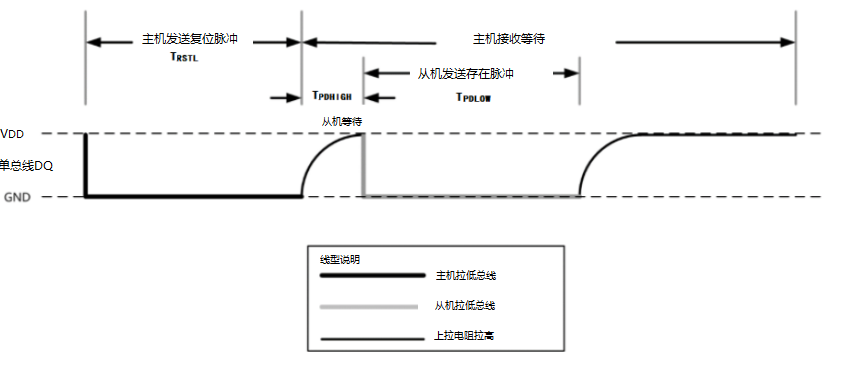


图2 复位脉冲和存在脉冲

当主机微控制器将单总线从逻辑高电平（无效）拉至逻辑低电平时，将启动写入时隙。所有写时隙的持续时间必须为60μs至130μs，周期之间的最短恢复时间为1μs。写入“0”并写入“1”时隙如图3所示。在写入“0”时隙期间，主机微控制器在时隙的持续时间内将线路拉低。但是，在写入“1”时隙期间，微控制器将线路拉低，然后在时隙开始后15μs内释放线路。

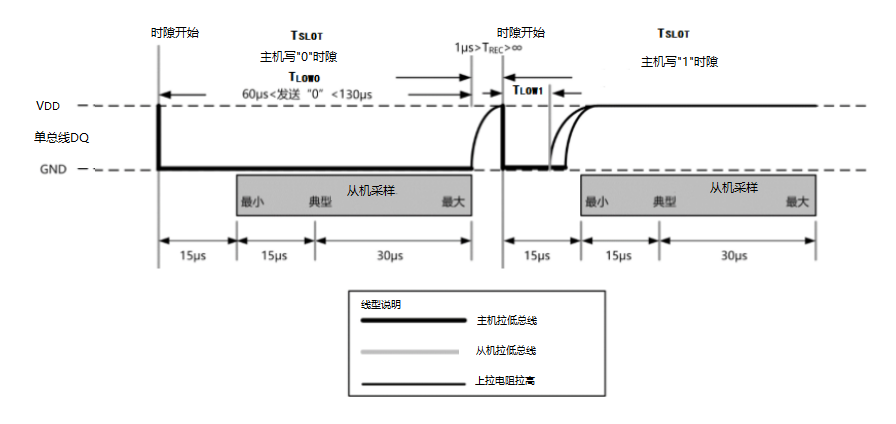


图3 写时隙

当主机微控制器将总线拉低1μs然后将其释放时，将启动读时隙，以便MY18E20可以控制线路并显示有效数据（高或低）。所有读取时隙的持续时间必须为60μs至130μs，并且周期之间的恢复时间至少为1μs（参见图4）。

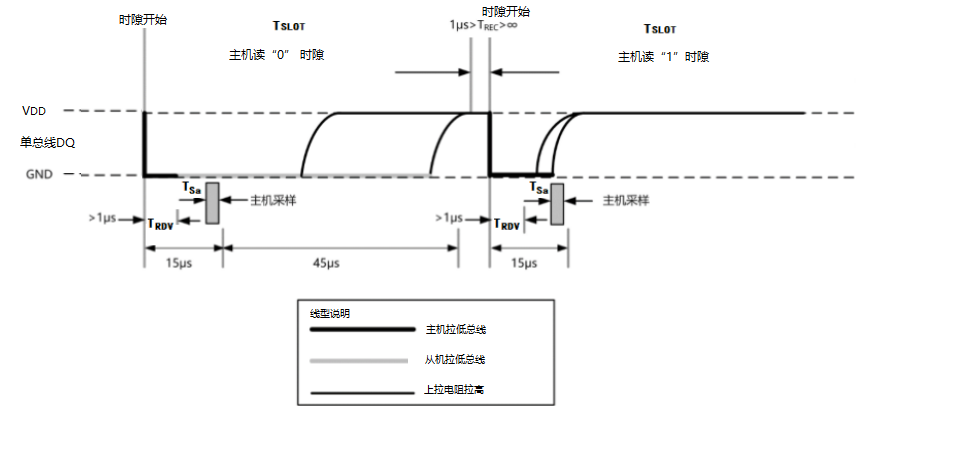


图4 读时隙

1. **软件控制**

为了精确控制单总线接口的特殊时序要求，必须首先建立某些关键功能。以下为测温流程相关功能伪代码，其中延时函数需根据主机微控制器配置自行编写。

4.1 复位例程

////////////////////////////////////////////////////////////////////

// 主机发送复位脉冲（大于480us低电平），并检查存在脉冲

unsigned char ow\_reset(void) {

unsigned char presence;

DQ = 0; //主机拉低DQ总线

Delay\_us(480); // 拉低DQ总线持续480us

DQ = 1; // 主机释放总线，由上拉电阻拉高

Delay\_us(80); // 等待80us后检查presence

presence = DQ; // 检查存在脉冲

Delay\_us(15); // 时序末尾等待

return(presence);

} // 0 = MY18E20存在, 1 = MY18E20不存在

4.2 主机读1-bit例程

///////////////////////////////////////////////////////////////////////// READ\_BIT – 从MY18E20读取一位数

unsigned char read\_bit(void) {

unsigned char res;

DQ = 0; //主机拉低DQ总线

Delay\_us(5); //拉低DQ总线持续5us

DQ = 1; //主机释放DQ总线，由上拉电阻拉高

Delay\_us(2); // 等待2us确保总线上数据稳定

res = DQ;

Delay\_us(50); //时序末尾等待

return(res); // 返回读取数值

}

4.3 主机写1-bit例程

///////////////////////////////////////////////////////////////////////// WRITE\_BIT – 主机写一位数据到MY18E20

void write\_bit(char bitval) {

DQ = 0; //主机拉低DQ总线

Delay\_us(5); //拉低DQ总线持续5us

if(bitval==1) DQ =1; // 如果写1，此时拉高DQ总线

Delay\_us (55); // 如果写0，则持续拉低DQ总线60us

DQ = 1; //释放DQ总线

Delay\_us (5); //（写0）恢复时间1us

}

4.4 主机读1-byte例程

///////////////////////////////////////////////////////////////////////// READ\_BYTE - 从MY18E20读取一字节数据

unsigned char read\_byte(void) {

unsigned char i, j;

unsigned char value = 0;

for (i=0;i<8;i++) {

j = read\_bit();

value = (value) | (j<<i); //每次读进1 位(低位先行), 然后左移

}

return(value);

}

4.5 主机写1-byte例程

///////////////////////////////////////////////////////////////////////// WRITE\_BYTE - 主机写一字节数据到MY18E20

void write\_byte(char val) {

unsigned char i;

unsigned char temp;

for (i=0; i<8; i++) { // 写字节,每次一位

temp = val>>i;

temp &= 0x01; // 低位先行

write\_bit(temp);

}

}

4.6 主机测温及读取例程（单点，12-bit精度）

///////////////////////////////////////////////////////////////////////

void Single-point\_Read\_Temperature\_12-bit(void) {

unsigned char get[10];

unsigned char tpmsb, tplsb, i;

short s\_tem;

float f\_tem;

ow\_reset();

write\_byte(0xCC); //Skip ROM，或使用匹配ROM序列号（见后文）

write\_byte(0x44); // 转换温度

Delay\_us (Measure\_Period);

// 温度转换时间，其中快速模式为15ms，慢速模式为500ms

ow\_reset();

write\_byte(0xCC); // Skip ROM，或使用匹配ROM序列号（见后文）

write\_byte(0xBE); // 读取Scratch Pad中数值

for (i=0;i<9;i++){

get[i]=read\_byte();

}

tpmsb = get[1]; // 温度高字节

tplsb = get[0]; // 温度低字节

s\_tem = tpmsb<<8;

s\_tem = s\_tem | tplsb;

s\_tem = s\_tem & 4095;

if( (s\_tem & 2048) == 2048 ) {

s\_tem = (s\_tem^4095) & 4095;

f\_tem = -1\*(s\_tem+1) \* 0.0625; //温度为负值

}

else f\_tem = s\_tem \* 0.0625;

printf( "\nTempF= %f degrees C\n", f\_tem); //打印摄氏温度

}

4.7 主机测温及读取例程（单点，14-bit精度）

///////////////////////////////////////////////////////////////////////

void Single-point\_Read\_Temperature\_14-bit(void) {

unsigned char get[10];

unsigned char tpmsb, tplsb, i;

short s\_tem, temp;

float f\_tem;

unsigned short lsb\_tem;

ow\_reset();

write\_byte(0xCC); //Skip ROM，或使用匹配ROM序列号（见后文）

write\_byte(0x44); // 转换温度

Delay\_us (Measure\_Period);

// 温度转换时间，其中快速模式为15ms，慢速模式为500ms

ow\_reset();

write\_byte(0xCC); // Skip ROM，或使用匹配ROM序列号（见后文）

write\_byte(0xBE); // 读取Scratch Pad中数值

for (i=0;i<9;i++){

get[i]=read\_byte();

}

tpmsb = get[1]; // 温度高字节

tplsb = get[0]; // 温度低字节

s\_tem = tpmsb<<8;

s\_tem = s\_tem | tplsb;

lsb\_tem = s\_tem & 49152;

lsb\_tem = lsb\_tem >> 14;

s\_tem = s\_tem & 4095;

temp = s\_tem << 2;

s\_tem = temp | lsb\_tem;

if( (s\_tem & 8192) == 8192 ) {

s\_tem = (s\_tem^ 16383) & 16383;

f\_tem = -1\*(s\_tem+1) \* 0.015625; //温度为负值

}

else f\_tem = s\_tem \* 0.015625;

printf( "\nTempF= %f degrees C\n", f\_tem); //打印摄氏温度

}

4.8主机读取ROM序列号例程

///////////////////////////////////////////////////////////////////////

void Read\_ROMCode(void) {

int n;

unsigned char rom[9];

ow\_reset();

write\_byte(0x33); //读取ROM指令

for (n=0;n<8;n++){

rom[n]=read\_byte();

}

printf("\n ROM Code = %X%X%X%X\n",rom[7],rom[6],rom[5],rom[4],rom[3],rom[2],rom[1],rom[0]);

}

4.9 主机匹配ROM序列号例程

///////////////////////////////////////////////////////////////////////

// 执行匹配ROM序列号，后续衔接相应的功能指令

//

unsigned char Send\_MatchRom(unsigned char (\*FoundROM)[8]) {

unsigned char i;

if(ow\_reset()) //检查总线上从机是否存在

write\_byte(0x55); // 匹配ROM指令

for(i=0;i<8;i++) {

write\_byte(FoundROM[numROMs][i]); //发送要匹配的ROM序列号

}

return true;

}

4.10 主机搜索ROM序列号例程

// 搜索算法

unsigned char Search\_ROM(uint8\_t (\*romID)[8],uint8\_t Num) {

unsigned char k,l = 0,ConflictBit,m,n;

unsigned char BUFFER[MAXNUM\_SWITCH\_18E20\_OUT] = {0};

unsigned char ss[64];

unsigned char s = 0;

int num = 0;

do {

ow\_reset();

write\_byte(0xF0); //搜索ROM序列号指令

for(m=0; m<8; m++) {

for(n=0; n<8; n++) {

k = read\_2Bit(); //主机从DQ总线上读两位数据,函数见后文

k = k&0x03;

s = s>>1;

if(k == 0x02) { //此时读到总线上当前数据位为0

write\_bit(0); //主机写0, 使总线上数据位为0的设备响应

ss[(m\*8+n)]=0;

}

else if(k == 0x01) { //此时读到总线上当前数据位为1

s = s|0x80;

write\_bit(1); //主机写1, 使总线上数据位为1的设备响应

ss[(m\*8+n)] = 1;

}

else if(k == 0x00) { //如果读到00, 则此位数据有冲突，需选择

ConflictBit = m\*8+n+1;

if(ConflictBit > BUFFER[l]){ //凡遇到新的差异位，选择0

write\_bit (0);

ss[(m\*8+n)] = 0; BUFFER[++l] = ConflictBit; }

else if(ConflictBit < BUFFER[l]) {

//凡上次遍历时最后一个走0的差异位之前的差异位仍按上次遍历的老路走

s = s|((ss[(m\*8+n)]&0x01)<<7);

write\_bit(ss[(m\*8+n)]);

}

else if(ConflictBit == BUFFER[l]) {

//凡上次遍历时最后一个走0的差异位本次应走1

s = s|0x80;

write\_bit(1);

ss[(m\*8+n)] = 1;

l = l-1;

}

}

else { //如果读到11，说明总线上不存在设备

return num; //搜索完成

}

}

romID[num][m] = s;

s = 0;

}

num = num+1;

}

while(BUFFER[l] != 0&&(num < MAXNUM\_SWITCH\_18E20\_OUT));

return num; //返回搜索到的MY18E20个数

}

unsigned char read\_2bit(void) {

unsigned char i, j;

unsigned char value = 0;

for (i=0;i<2;i++) {

j = read\_bit();

value = (value) | (j<<i);

// 每次读进1 bit(低位先行), 然后左移到对应位置

}

return(value);

}

4.11单总线多节点测温及读取例程（12-bit精度）

/////////////////////////////////////////////////////////////////////

// Search\_ROM()与Send\_MatchRom()函数功能请参考上文

//

void Multi-points\_Read\_Temperature\_12-bit(void) {

unsigned char tpmsb, tplsb, j, get[9];

short s\_tem, temp;

float f\_tem;

unsigned short lsb\_tem;

int slave\_num, i;

unsigned char ROM\_Buff[MAXNUM\_18E20][8];

// MAXNUM\_18E20为总线上节点设备个数，用户按需自行定义

ow\_reset();

write\_byte(0xCC); //Skip ROM

write\_byte(0x44); // 总线上所有节点设备开始转换温度

Delay\_us (Measure\_Period);

// 温度转换时间，其中快速模式为15ms，慢速模式为500ms

slave\_num = Search\_ROM(ROM\_Buff, MAXNUM\_18E20);

//收集总线上所有节点设备ROM号

for(i=0; i<slave\_num; i++) {

Send\_MatchRom(ROM\_Buff[i]); //根据搜索到的ROM号进行匹配

write\_byte(0xBE); // 读取Scratch Pad中数值

for (j=0;j<9;j++){get[j]=read\_byte();}

tpmsb = get[1]; // 温度高字节

tplsb = get[0]; // 温度低字节

s\_tem = tpmsb<<8;

s\_tem = s\_tem | tplsb;

s\_tem = s\_tem & 4095;

if( (s\_tem & 2048) == 2048 ) {

s\_tem = (s\_tem^4095) & 4095;

f\_tem = -1\*(s\_tem+1) \* 0.0625; //温度为负值

}

else f\_tem = s\_tem \* 0.0625;

printf( "\nTempF= %f degrees C\n", f\_tem); //输出温度

}

}

4.12 单总线多节点测温及读取例程（14-bit精度）

/////////////////////////////////////////////////////////////////////

// Search\_ROM()与Send\_MatchRom()函数功能请参考上文

void Multi-points\_Read\_Temperature\_14-bit(void) {

unsigned char tpmsb, tplsb, j, get[9];

short s\_tem, temp;

float f\_tem;

unsigned short lsb\_tem;

int slave\_num, i;

unsigned char ROM\_Buff[MAXNUM\_18E20][8];

// MAXNUM\_18E20为总线上节点设备个数，用户按需自行定义

ow\_reset();

write\_byte(0xCC); //Skip ROM

write\_byte(0x44); // 总线上所有节点设备开始转换温度

Delay\_us (Measure\_Period);

// 温度转换时间，其中快速模式为15ms，慢速模式为500ms

slave\_num = Search\_ROM(ROM\_Buff, MAXNUM\_18E20);

//收集总线上所有节点设备ROM号

for(i=0; i<slave\_num; i++) {

Send\_MatchRom(ROM\_Buff[i]); //根据搜索到的ROM号进行匹配

write\_byte(0xBE); // 读取Scratch Pad中数值

for (j=0;j<9;j++){get[j]=read\_byte();}

tpmsb = get[1]; // 温度高字节

tplsb = get[0]; // 温度低字节

s\_tem = tpmsb<<8;

s\_tem = s\_tem | tplsb;

lsb\_tem = s\_tem & 49152;

lsb\_tem = lsb\_tem >> 14;

s\_tem = s\_tem & 4095;

temp = s\_tem << 2;

s\_tem = temp | lsb\_tem;

if( (s\_tem & 8192) == 8192 ) {

s\_tem = (s\_tem^ 16383) & 16383;

f\_tem = -1\*(s\_tem+1) \* 0.015625; //温度为负值

}

else f\_tem = s\_tem \* 0.015625;

printf( "\nTempF= %f degrees C\n", f\_tem); //输出摄氏温度

}

}