DS18B20

目录

一、概述

1.DS18B20 概述

传统的温度检测大多以热敏电阻为传感器，采用热敏电阻，可满足 40℃至 90℃测量范围，但热敏电阻可靠性差，测量温度准确率低，对于小于 1℃的温度信号是不适用的，还得经过专门的接口电路转换成数字信号才能由微处理器进行处理。

目前常用的微机与外设之间进行的数据通信的串行总线主要有 i2c 总线，spi 总线等。其中 i2c 总线以同步串行 2 线方式进行通信（一条时钟线，一条数据线），spi 总线则以同步串行 3 线方式进行通信（一条时钟线，一条数据输入线，一条数据输出线）。这些总线至少需要两条或两条以上的信号线。而单总线（ 1-wire bus ），采用单根信号线，既可传输数据，而且数据传输是双向的， cpu 只需一根端口线就能与诸多单总线器件通信，占用微处理器的端口较少，可节省大量的引线和逻辑电路。 因而，这种单总线技术具有线路简单，硬件开销少，成本低廉，软件设计简单，便于总线扩展和维护。同时，基于单总线技术能较好地解决传统识别器普遍存在的携带不便，易损坏，易受腐馈，易受电磁干扰等不足，因此，单总线具有广阔的应用前景，是值得关注的一个发展领域。

单总线即只有一根数据线，系统中的数据交换，控制都由这根线完成。主机或从机通过一个漏极开路或三态端口连至数据线，以允许设备在不发送数据时能够释放总线，而让其它设备使用总线。**单总线通常要求外接一个约为 4.7k 的上拉电阻，这样，当总线闲置时其状态为高电平。**

ds18b20 数字式温度传感器，与传统的热敏电阻有所不同的是，使用集成芯片，采用单总线技术，其能够有效的减小外界的干扰，提高测量的精度。同时，它可以直接将被测温度转化成串行数字信号供微机处理，接口简单， 使数据传输和处理简单化。 部分功能电路的集成，使总体硬件设计更简洁，能有效

地降低成本，搭建电路和焊接电路时更快，调试也更方便简单化，这也就缩短了开发的周期 。

2.DS18B20 的主要特征：

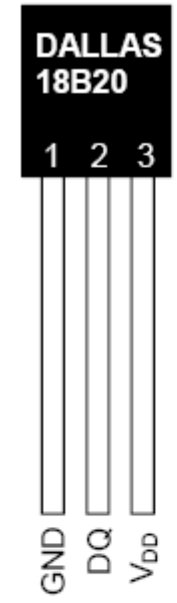
全数字温度转换及输出。
先进的单总线数据通信。
最高12位分辨率，精度可达土0.5摄氏度。
12位分辨率时的最大工作周期为750毫秒。
可选择寄生工作方式。
检测温度范围为－55° C ~+125° C （－67° F ~+257° F)
内置EEPROM，限温报警功能。
64位光刻ROM，内置产品序列号，方便多机挂接。
多样封装形式，适应不同硬件系统。

3.DS18B20 测温原理

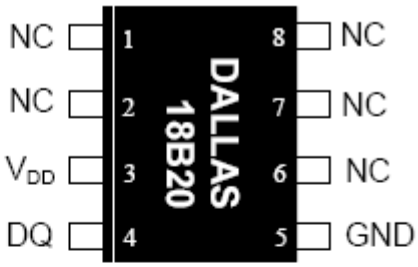
低温度系数晶振的振荡频率受温度的影响很小，用于产生固定频率的脉冲信号送给减法计数器 1，为计数器提供一频率稳定的计数脉冲。高温度系数晶振随温度变化其震荡频率明显改变，很敏感的振荡器，所产生的信号作为减法计数器 2 的脉冲输入，为计数器 2 提供一个频率随温度变化的计数脉冲。图中还隐含着计数门，当计数门打开时，ds18b20 就对低温度系数振荡器产生的时钟脉冲进行计数，进而完成温度测量。计数门的开启时间由高温度系数振荡器来决定，每次测量前，首先将-55℃ 所对应的基数分别置入减法计数器 1 和温度寄存器中，减法计数器 1 和温度寄存器被预置在-55℃ 所对应的一个基数值。减法计数器 1 对低温度系数晶振产生的脉冲信号进行减法计数，当减法计数器 1 的预置值减到 0 时温度寄存器的值将加 1，减法计数器 1 的预置将重新被装入，减法计数器 1 重新开始对低温度系数晶振产生的脉冲信号进行计数，如此循环直到减法计数器 2 计数到 0 时，停止温度寄存器值的累加，此时温度寄存器中的数值即为所测温度。斜率累加器用于补偿和修正测温过程中的非线性，其输出用于修正减法计数器的预置值，只要计数门仍未关闭就重复上述过程，直至温度寄存器值达到被测温度值。

二、DS18B20 结构

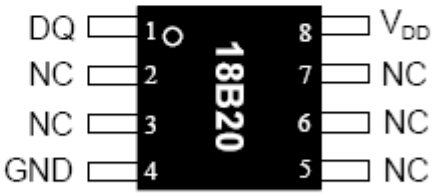
1.DS18B20 芯片封装结构：



TO-92
(DS18B20)



8-Pin 150mil SO
(DS18B20Z)



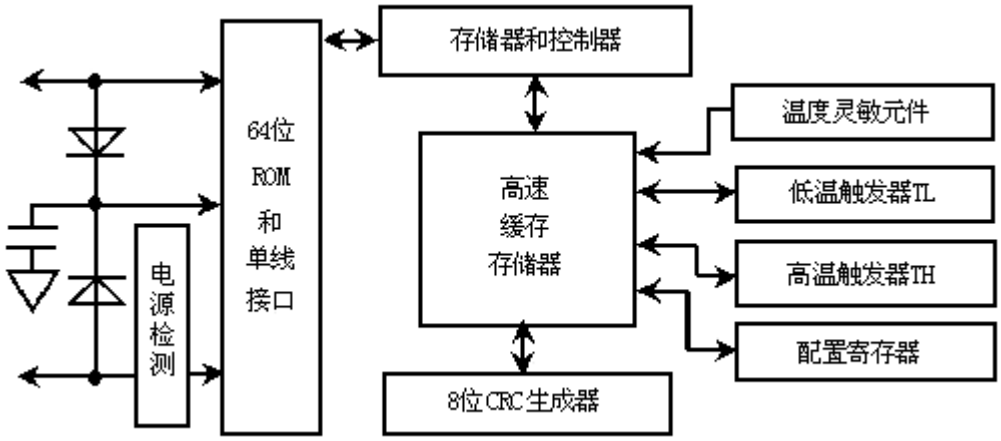
8-Pin μSOP
(DS18B20U)

DS18B20引脚功能：

•GND 电压地 •DQ 单数据总线 •VDD 电源电压（3.0-5.5V） •NC 空引脚

2.DS18B20 内部结构：

DS18B20 内部结构主要由四部分组成：64 位光刻 ROM，温度传感器，温度报警触发器 TH 和 TL,配置寄存器。



64 位光刻 rom：

光刻 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。64 位光刻 rom 的排列是：开始 8 位（地址： 28h ）是产品类型标号，接着的 48 位是该 ds18b20 自身的序列号，并且每个 ds18b20 的序列号都不相同，因此它可以看作是该 ds18b20 的地址序列码；最后 8 位则是前面 56 位的循环冗余校验码（ crc=x8+x5+x4+1 ）。由于每一个 ds18b20 的 rom 数据都各不相同，因此微控制器就可以通过单总线对多个 ds18b20 进行寻址，从而实现一根总线上挂接多个 ds18b20 的目的。

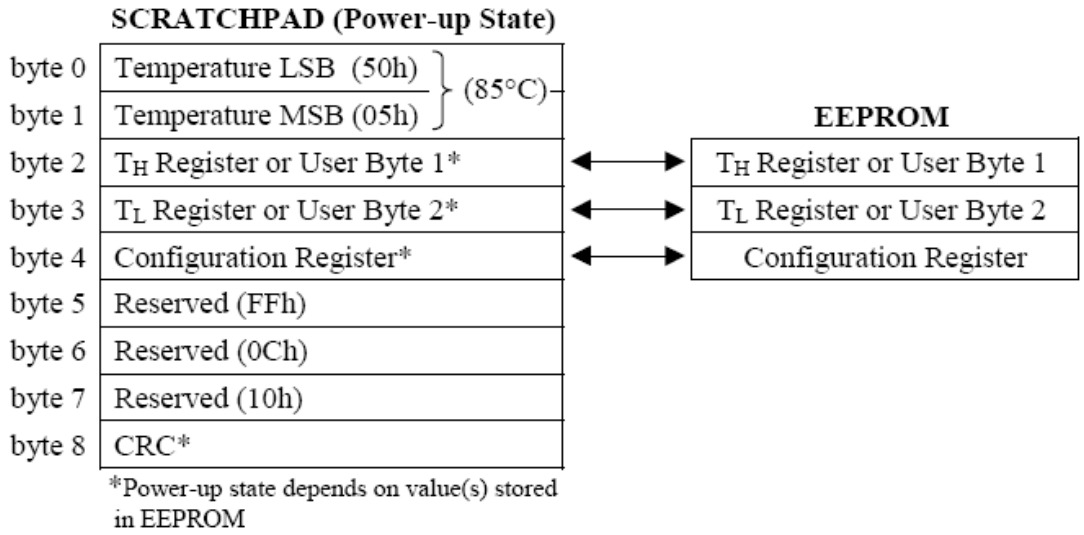
温度传感器

DS18B20 中的温度传感器可完成对温度的测量，以 12 位转化为例：用 16 位符号扩展的二进制补码读数形式提供，以 0.0625℃/LSB 形式表达，其中 S 为符号位。这是 12 位转化后得到的 12 位数据，存储在 18B20 的两个 8 比特的 RAM 中，二进制中的前面 5 位是符号位，**如果测得的温度大于 0， 这 5 位为 0，只要将测到的数值乘于 0.0625 即可得到实际温度；如果温度小于 0，这 5 位为 1，测到的数值需要取反加 1 再乘于 0.0625 即可得到实际 温度。** 例如+125℃的数字输出为 07D0H，+25.0625℃的数字输出为 0191H，-25.0625℃的数字输出为 FE6FH，-55℃的数字输出为 FC90H 。

温度传感器的存储器

ds18b20 温度传感器的内部存储器包括一个高速暂存 ram 和一个非易失性的可电擦除的 e2ram，后者存放高温度和低温度触发器 th 、tl 和结构寄存器。数据先写入 ram，经校验后再传给 e2ram 。

暂存存储器包含了 8 个连续字节，前两个字节是测得的温度信息，第一个字节的内容是温度的低八位 tl，第二个字节是温度的高八位 th。第三个和第四个字节是 th、tl 的易失性拷贝，第五个字节是结构寄存器的易失性拷贝，这三个字节的内容在每一次上电复位时被刷新。第六、七、八个字节用于内部计算。第九个字节是冗余检验字节，可用来保证通信正确。ds18b20 的分布如下：



配置寄存器

配置寄存器是配置不同的位数来确定温度和数字的转化。

TM	R1	R0	1	1	1	1	1

低五位一直都是"1"，TM 是测试模式位，用于设置 DS18B20 在工作模式还是在测试模式。在 DS18B20 出厂时该位被设置为 0，用 户不要去改动。R1 和 R0 用来设置分辨率（**DS18B20 出厂时被设置为 12 位**）

三、DS18B20 工作流程

1.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）[FOH]

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。

2.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。

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”）。

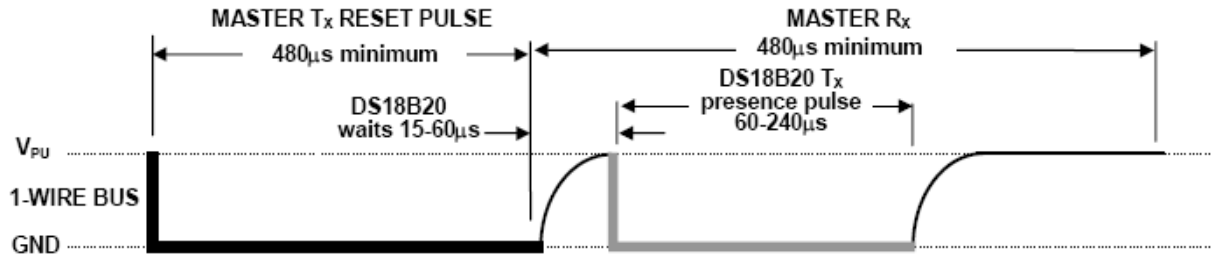
4， 控制器发送存储器操作指令：在ROM指令发送给18B20之后，紧接着（不间断）就是发送存储器操作指令了。操作指令同样为8位，共6条，存储器操作指令分别是写RAM数据、读RAM数据、将RAM数据复制到EEPROM、温度转换、将EEPROM中的报警值复制到RAM、工作方式切换。存储器操作指令的功能是命令18B20作什么样的工作，是芯片控制的关键。

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。

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（0xCC）、执行温度转换存储器操作指令（0x44）、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（0xCC）、执行读RAM的存储器操作指令（0xBE）、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）。

3.DS18B20 复位、读、写时序图

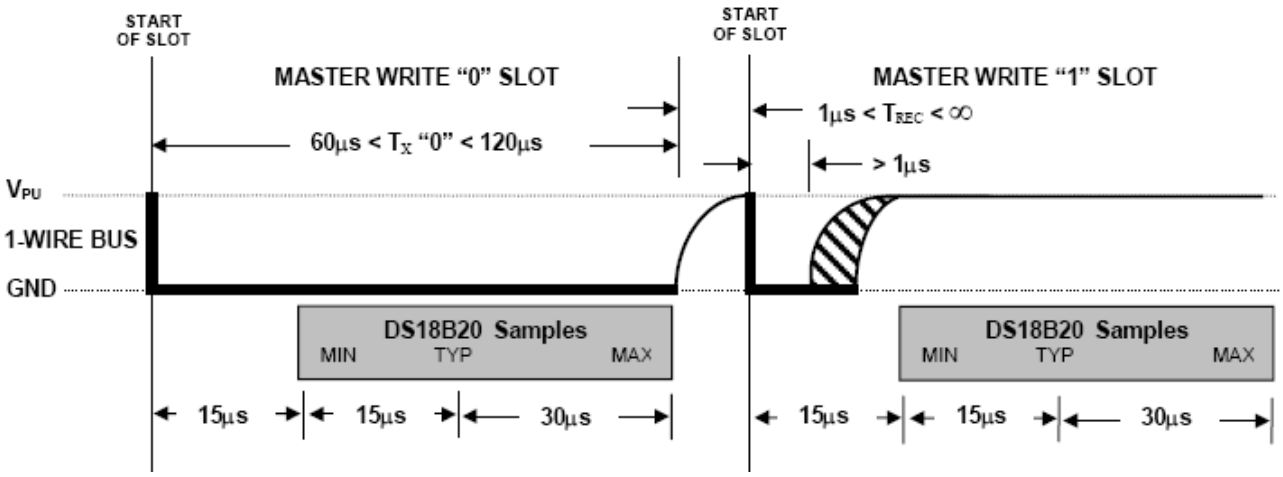
DS18B20 复位及应答关系示意图：



首先释放总线，然后将总线拉低，并延时 480us，将总线拉高，15-60us 后，18b20 会返回一个 60-240us 的低电平信号（存在脉冲），检测这个信号，释放总线，延时至少 240us，至此，通信双方已经达成了基本的协议。

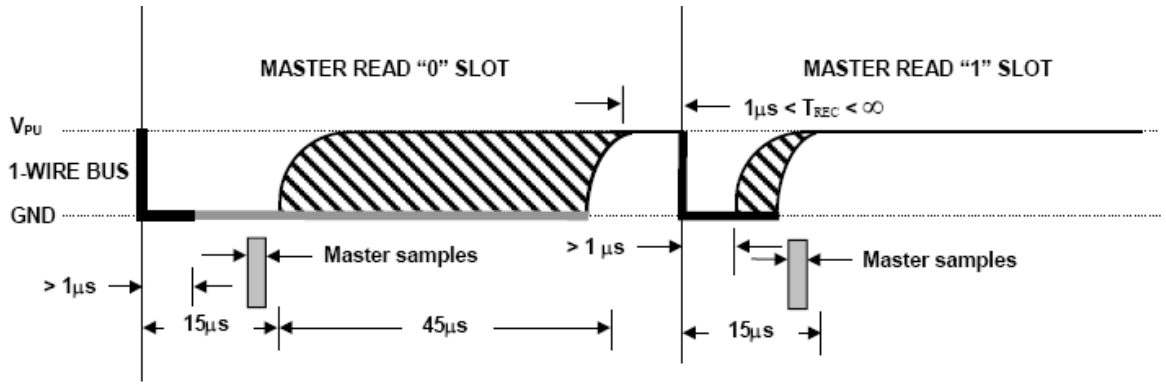
DS18B20 读写时间隙：

写时间隙：



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。具体操作是先将总线拉低至少 15uS，如果写 0，延时 45uS，之后再将总线拉高，并给与一定的延时以保证整个位的发送时间在 60-120uS，如果写 1，先将总线拉高，延时 45uS，在给与一定的延时以保证整个位的发送时间在 60-120uS。

读时间隙：



首先释放总线，然后让总线拉低，延时至少 1uS，以表示读时间的开始，随后释放总线，18B20 会发送内部数据位，我们要在 15uS 内读出这些数据位，读完释放总线，之后必须加一个至少 45uS 的延时，以保证整个位的读取时间大于 60-120uS。也必须保证连续读数直接有一定的时间间隔。

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（寄存器）开始的。

四、ds18b20 使用中注意事项

ds1820 虽然具有测温系统简单、测温精度高、连接方便、占用口线少等优点，但在实际应用中也应注意以下几方面的问题：

（1）每一次读写之前都要对 ds18b20 进行复位，复位成功后发送一条 rom 指令，最后发送 ram 指令，这样才能对 ds18b20 进行预定的操作。复位要求主 cpu 将数据线下拉 500 微秒，然后释放， ds18b20 收到信号后等待 16 ～ 60 微秒左右，后发出 60 ～ 240 微秒的存在低脉冲，主 cpu 收到此信号表示复位成功。（所有的读写时序至少需要 60us ，且每个独立的时序之间至少需要 1us 的恢复时间。在写时序时，主机将在下拉低总线 15us 之内释放总线，并向单总线器件写 1 ；若主机拉低总线后能保持至少 60us 的低电平，则向单总线器件写 0 。单总线仅在主机发出读写时序时才向主机传送数据，所以，当主机向单总线器件发出读数据指令后，必须马上产生读时序，以便单总线器件能传输数据。）

（2）在写数据时，写 0 时单总线至少被拉低 60us， 写 1 时 ，15us 内就得释放总线。

（3）转化后得到的 12 位数据，存储在 18b20 的两个 8 比特的 ram 中，二进制中的前面 5 位是符号位，如果测得的温度大于 0 ，这 5 位为 0 ，只要将测到的数值乘于 0.0625 即可得到实际温度；如果温度小于 0 ，这 5 位为 1 ，测到的数值需要取反加 1 再乘于 0.0625 即可得到实际温度。

（4）较小的硬件开销需要相对复杂的软件进行补偿，由于 ds1820 与微处理器间采用串行数据传送，因此，在对 ds1820 进行读写编程时，必须严格的保证读写时序，否则将无法读取测温结果。在使用 pl/m 、 c 等高级语言进行系统程序设计时，对 ds1820 操作部分最好采用汇编语言实现。

（5） 在 ds1820 的有关资料中均未提及单总线上所挂 ds1820 数量问题，容易使人误认为可以挂任意多个 ds1820 ，在实际应用中并非如此。当单总线上

所挂 ds1820 超过 8 个时，就需要解决微处理器的总线驱动问题，这一点在进行多点测温系统设计时要加以注意。

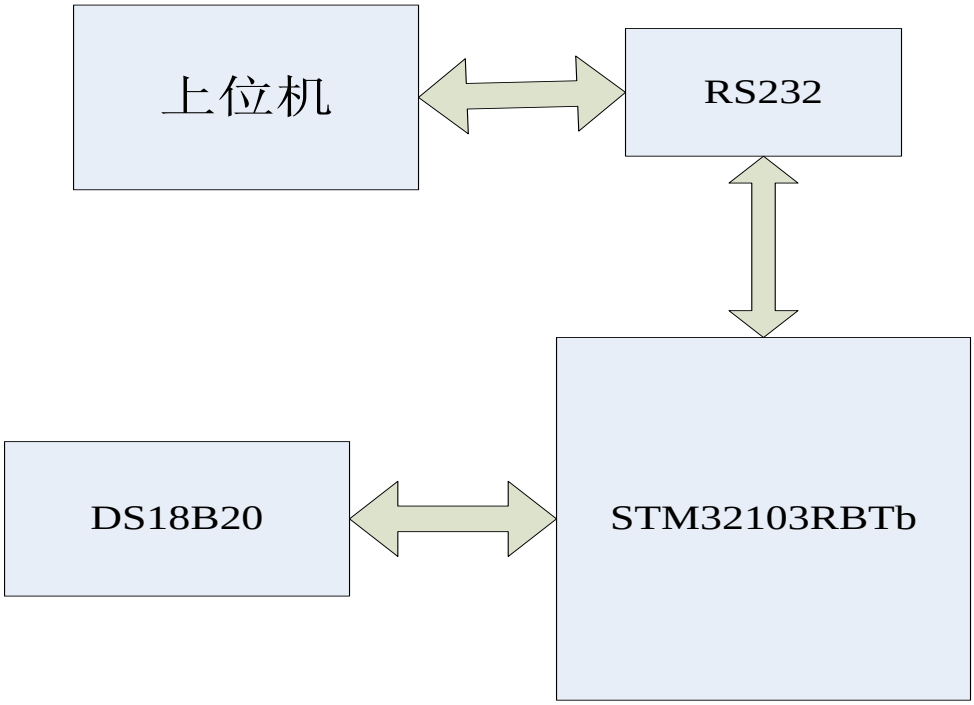
（6） 连接 ds1820 的总线电缆是有长度限制的。试验中，当采用普通信号电缆传输长度超过 50m 时，读取的测温数据将发生错误。当将总线电缆改为双绞线带屏蔽电缆时，正常通讯距离可达 150m ，当采用每米绞合次数更多的双绞线带屏蔽电缆时，正常通讯距离进一步加长。这种情况主要是由总线分布电容使信号波形产生畸变造成的。因此，在用 ds1820 进行长距离测温系统设计时要充分考虑总线分布电容和阻抗匹配问题。测温电缆线建议采用屏蔽 4 芯双绞线，其中一对线接地线与信号线，另一组接 vcc 和地线，屏蔽层在源端单点接地。

（7）在 ds1820 测温程序设计中，向 ds1820 发出温度转换命令后，程序总要等待 ds1820 的返回信号，一旦某个 ds1820 接触不好或断线，当程序读该 ds1820 时，将没有返回信号，程序进入死循环。这一点在进行 ds1820 硬件连接和软件设计时也要给予一定的重视。

（8）ds18b20 出厂时的设置是 12 位，默认温度是 85 度，发出温度转化指令后，要给与至少 500uS 的延时以保证温度转换。

五、stm32 下 DS18B20 的驱动

1. 系统框图

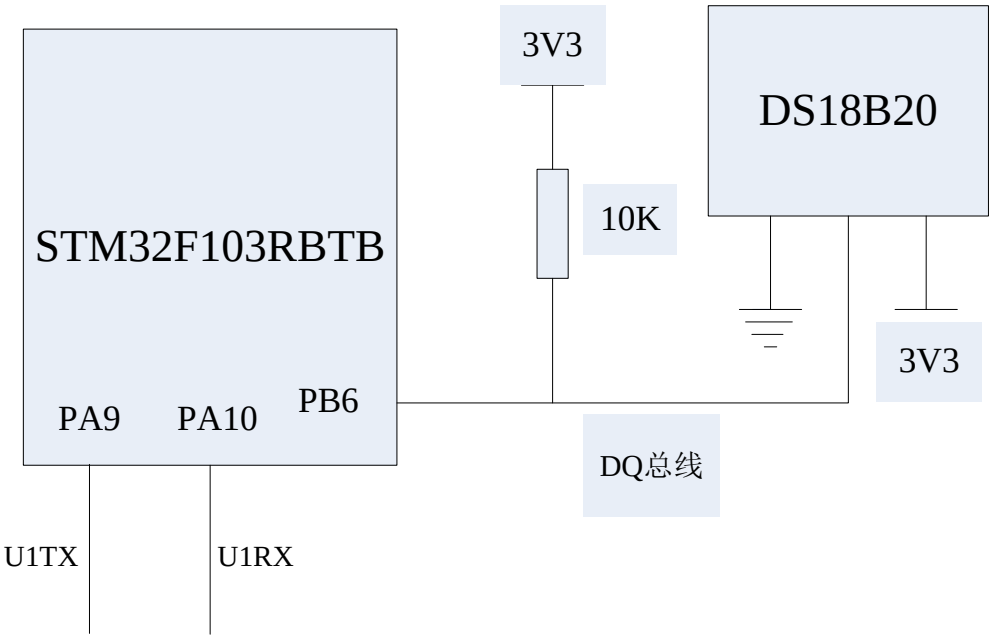


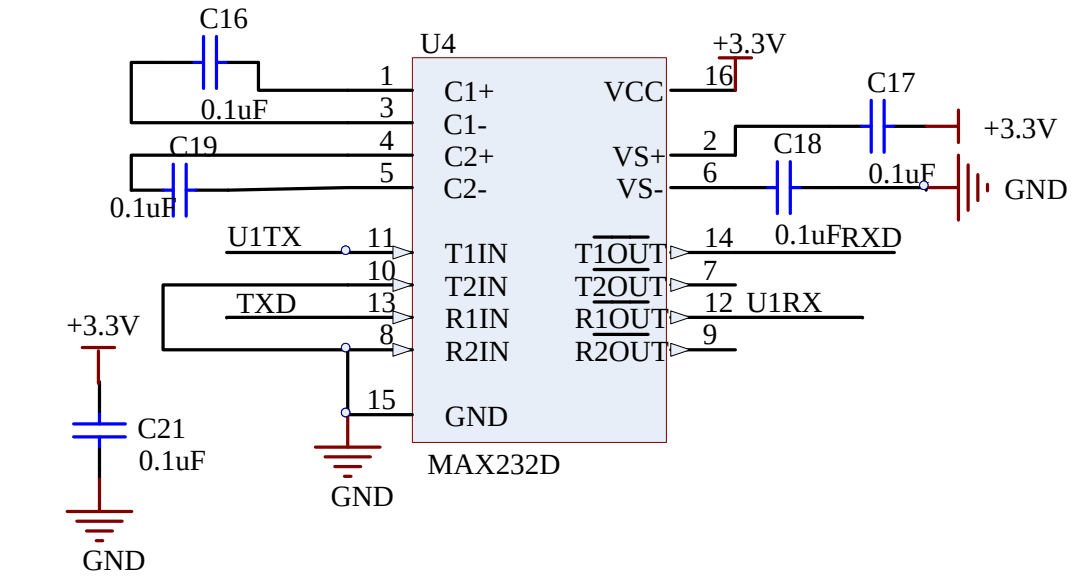
DS18B20 将感应到的温度模拟信号转换为数字电信号后，输入到温度检测模块，由温度检测模块传给微控制器模块，进行数据的处理， 处理好的数据一方面通过RS232 通讯传输给上位机实时监控显示。

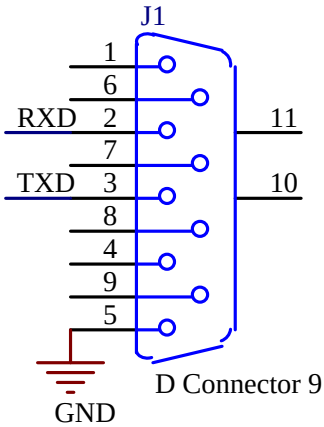
2.硬件电路图

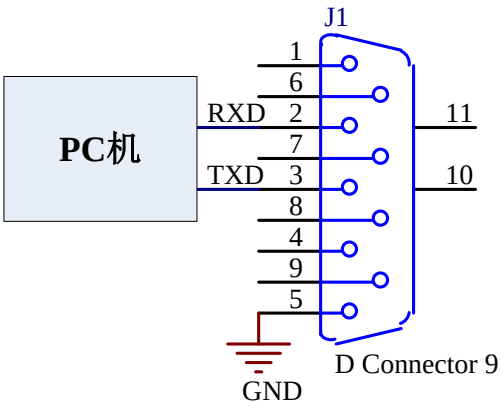
主机与DS18B20 交换数据主要靠CPU 按照1-wire 单总线协议在单总线上产生复位时序和读写时序来实现。其中包含复位脉冲、响应脉冲，写1 写0 读1、读0 时序。只有响应脉冲由DS18B20 发出，其余都由主机(程序)发出。

主机得到数据后与通过MAX232传输到PC机上，并通过串口调试助手显示。

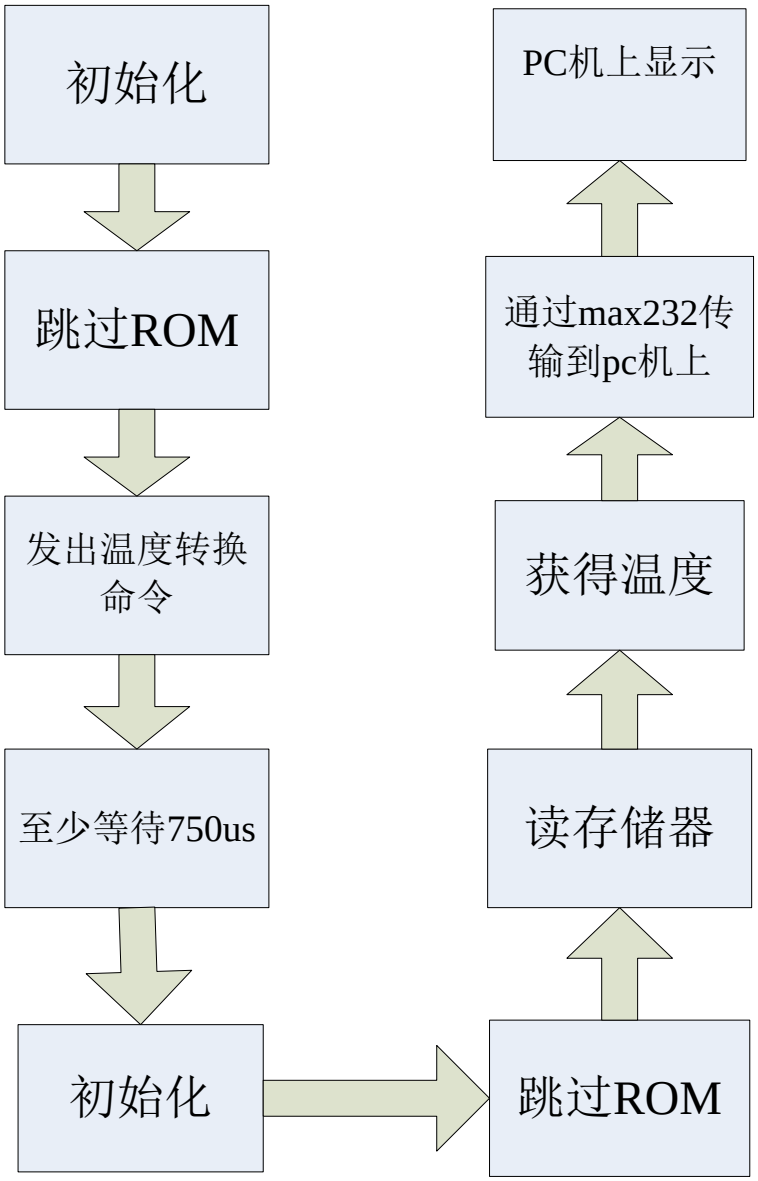

STM32F103RBTB
PA9
PA10
PB6
U1TX
U1RX
3V3
10K
DS18B20
3V3
DQ总线


C16
0.1uF
C19
0.1uF
U4
C1+
C1-
C2+
C2-
VCC
VS+
VS-
T1IN
T2IN
R1IN
R2IN
T1OUT
T2OUT
R1OUT
R2OUT
GND
MAX232D
+3.3V
C17
0.1uF
C18
0.1uF
+3.3V
GND
U1TX
TXD
RXD
U1RX
+3.3V
C21
0.1uF
GND
GND


J1
RXD
TXD
11
10
D Connector 9
GND


PC机
J1
RXD
TXD
11
10
D Connector 9
GND

3.软件部分


初始化
跳过ROM
发出温度转换命令
至少等待750us
初始化
跳过ROM
读存储器
获得温度
通过max232传输到pc机上
PC机上显示

程序如下：

Ds18b20.c：

```
#include "stm32f10x.h"
#include "ds18b20.h"
 void us_delay(u16 us)
{
        while(us--);
}

void ms_delay(u16 count)
{
        u8 i;
        while(count) {
                i = 255;
                while(i --);
                count --;
        }
}
void ds18b20_reset(void)
{
   GPIO_Configuration1();
   GPIO_SetBits(GPIOB,  GPIO_Pin_6);
  // GPIO_WriteBit(GPIOB,  GPIO_Pin_6, Bit_SET);
   us_delay(10);
   GPIO_ResetBits( GPIOB,  GPIO_Pin_6);
   us_delay(4000); //300us
   GPIO_SetBits(GPIOB,  GPIO_Pin_6);
```

```
   //GPIO_WriteBit(GPIOB,  GPIO_Pin_6, Bit_SET);
   GPIO_Configuration2();
   while(GPIO_ReadInputDataBit( GPIOB,  GPIO_Pin_6));
  // while(GPIO_ReadInputDataBit( GPIOB,  GPIO_Pin_6)==0)

   us_delay(4000);
   GPIO_Configuration1();
}

 u8   ds18b20_read_byte(void)
 {
   //u8 value=0;
   u8 i;
   u8 data=0;
   //GPIO_Configuration1();
   //GPIO_SetBits(GPIOB,  GPIO_Pin_6);
  // us_delay(15);
   for(i=0;i<8;i++)
   {
       data>>=1;
       GPIO_Configuration1();
       GPIO_ResetBits( GPIOB,  GPIO_Pin_6);
     us_delay(15);
       //GPIO_SetBits(GPIOB,  GPIO_Pin_6);

       GPIO_Configuration2();
       us_delay(50);
```

```
        //GPIO_WriteBit(GPIOB,  GPIO_Pin_6, Bit_SET);

        if(GPIO_ReadInputDataBit( GPIOB,  GPIO_Pin_6))
           data|=0x80;
        else
           data&=0x7f;
        //value=(data<<7)|(value>>1);
        us_delay(600);

        //GPIO_SetBits(GPIOB,  GPIO_Pin_6);
    }
       return data;

}

  void ds18b20_write_byte(u8 dat)
{
     u8 i;
     u8 one_bit;

     for(i = 0; i < 8; i ++)
       {
        one_bit = dat & 0x01;
        dat = dat >> 1;
          GPIO_Configuration1();
          GPIO_ResetBits( GPIOB,  GPIO_Pin_6);
            us_delay(125);//15us
```

```
          if(one_bit)
          {
              GPIO_SetBits(GPIOB,  GPIO_Pin_6);
            us_delay(300); //45us
          }
          else
          {
            us_delay(300) ; //45us
              GPIO_SetBits(GPIOB,  GPIO_Pin_6);
          }

       //GPIO_Configuration2();
     us_delay(80) ;
    }
}

 void temperature_convert(void)
 {
      ds18b20_reset();
      ds18b20_write_byte(0xCC);
      ds18b20_write_byte(0x44);

 }

u16 get_temperature(void)
 {
       u8 a, b;
```

```
    u16 temp;

    ds18b20_reset();
    ds18b20_write_byte(0xCC);
    ds18b20_write_byte(0xBE);

    a =ds18b20_read_byte();
    b =ds18b20_read_byte();

    temp = (u16)b << 8 | a;

    return temp;
 }

void initialize_ds18b20(void)
{
        u16 temp;

        do {
                temperature_convert();
                ms_delay(20000);
                temp = get_temperature();
        } while (temp == 85);
        temperature_convert();
}

void GPIO_Configuration1(void)
```

```
 {

     GPIO_InitTypeDef ds18b20_in ;
     ds18b20_in.GPIO_Pin = GPIO_Pin_6;
       ds18b20_in.GPIO_Speed = GPIO_Speed_50MHz;
       ds18b20_in.GPIO_Mode = GPIO_Mode_Out_PP;
       GPIO_Init(GPIOB, &ds18b20_in);
}

void GPIO_Configuration2(void)
{
     GPIO_InitTypeDef ds18b20_out ;
     ds18b20_out.GPIO_Pin = GPIO_Pin_6;
     ds18b20_out.GPIO_Speed = GPIO_Speed_50MHz;
     ds18b20_out.GPIO_Mode = GPIO_Mode_IN_FLOATING;
       GPIO_Init(GPIOB, &ds18b20_out);

}
```

Ds18b20.h：

```
#ifndef _DS18B20_H
#define _DS18B20_H
extern u8 flag;

void us_delay(u16 us);
```

```
void ms_delay(u16 count);                          //延时子函数
void ds18b20_reset(void);
u8 ds18b20_read_byte(void);                        //读一字节
void ds18b20_write_byte(u8 dat);        //写一个字节
void temperature_convert(void);
u16 get_temperature(void);
void initialize_ds18b20(void);
void GPIO_Configuration1(void);
void GPIO_Configuration2(void);
#endif
```

Main.c：

```
#include "stm32f10x.h"
#include "ds18b20.h"
void RCC_Configuration(void);
void GPIO_Configuration(void);
void USART_Configuration(void);
void TIM_Configuration(void);
u16 showdata=0;
//u8 showdata1=0;
//u8 showdata2=0;
u8 flag=0;

u8 i;
u8 data;
float dat;
extern      void ms_delay(u16 count) ;
int main(void)
```

```
{
 RCC_Configuration();
 GPIO_Configuration();
 USART_Configuration();
 initialize_ds18b20() ;
 TIM_Configuration() ;

 while(1)
 {
  if(TIM_GetFlagStatus(TIM1, TIM_IT_Update)!=RESET)
    {
     u16 j;
     TIM_ClearFlag(TIM1, TIM_IT_Update);

       temperature_convert();
       ms_delay(20000);
     showdata=get_temperature();
       j=showdata&0xEFFF;
       j>>=15;

       //        发送高八位
     //showdata=showdata>>8;
     //data=showdata&0xff;

       if(j==0)
       {
         dat=showdata;
```

```
dat*=0.0625;
  //发送整数部分
data=(u8)dat;
USART_SendData(USART1, data);

  //发送小数部分
  dat*=100;
  us_delay(4000); //300us
  data=(u8)dat%100;
  USART_SendData(USART1, data);
 }
 if(j==0x0001)
 {
   showdata=~showdata;
   showdata+=1;
   data=(u8)dat;
USART_SendData(USART1, data);

  //发送小数部分
  dat*=100;
  us_delay(4000); //300us
  data=(u8)dat%100;
  USART_SendData(USART1, data);
 }

//us_delay(4000); //300us
  //   发送低八位
```

```
    //data=(u8) showdata&0xff;
    //USART_SendData(USART1, data);

    }
   }
 }

 void RCC_Configuration(void)
 {        ErrorStatus HSEStartUpStatus ;

        RCC_DeInit();                // RCC system reset(for debug
purpose)
        RCC_HSEConfig(RCC_HSE_ON);                // Enable HSE
        HSEStartUpStatus = RCC_WaitForHSEStartUp();                //
Wait till HSE is ready

        if(HSEStartUpStatus == SUCCESS) {
                FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
// Enable Prefetch Buffer
                FLASH_SetLatency(FLASH_Latency_2);                //
Flash 2 wait state

                RCC_HCLKConfig(RCC_SYSCLK_Div1);                //
HCLK = SYSCLK
                RCC_PCLK2Config(RCC_HCLK_Div1);                //
PCLK2 = HCLK
                RCC_PCLK1Config(RCC_HCLK_Div2);                //
PCLK1 = HCLK/2
                RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
// configure PLLCLK
```

```
                RCC_PLLCmd(ENABLE);         // Enable PLL

                while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET) {}
// Wait till PLL is ready



                RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
// Select PLL as system clock source

                while(RCC_GetSYSCLKSource() != 0x08) {}         //
Wait till PLL is used as system clock source

        }




RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOA|RCC_
APB2Periph_AFIO , ENABLE);

         RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);

}



 void GPIO_Configuration(void)

 {     GPIO_InitTypeDef GPIO_InitStructure ;


       GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;

       GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;

       GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;

       GPIO_Init(GPIOA, &GPIO_InitStructure);



       GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;

     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
```

```
    GPIO_Init(GPIOA, &GPIO_InitStructure);



    //GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
     // GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
     // GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    //GPIO_Init(GPIOB, &GPIO_InitStructure);

}




void USART_Configuration(void)
{ USART_InitTypeDef USART_InitStructure ;
  USART_InitStructure.USART_BaudRate = 9600;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl =
  USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
  USART_Init(USART1, &USART_InitStructure);
  USART_Cmd(USART1,ENABLE);
```

```
}

 void TIM_Configuration(void)
 {
      TIM_TimeBaseInitTypeDef  TIM_TimeBaseInitStruct;
      RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);

      TIM_DeInit(TIM1);
    TIM_TimeBaseInitStruct.TIM_Period =10000;
    TIM_TimeBaseInitStruct.TIM_Prescaler =0;
    TIM_TimeBaseInitStruct.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInitStruct.TIM_RepetitionCounter = 0x0000;
    TIM_TimeBaseInit(TIM1,&TIM_TimeBaseInitStruct);
    TIM_PrescalerConfig(TIM1, 0x8C9F, TIM_PSCReloadMode_Immediate);
    TIM_ARRPreloadConfig(TIM1, DISABLE);
      TIM_ClearFlag(TIM1, TIM_FLAG_Update) ;
    TIM_ITConfig( TIM1,  TIM_IT_Update, ENABLE);
      TIM_Cmd( TIM1, ENABLE);
 }
```

4．下载与调试

将程序通过 J-LINK 下载到 stm32 开发板中，并调试，结果通过串口调试助手显示出来，如下所示：

串口调试助手每次接收 4 个数字，前面 2 个为整数部分，后面 2 个为小数部分，数字以 16 进制表示，如 15 59，就表示测量温度为 21.89 度。可以通过调试程序，让调试助手每隔一段时间（自己设定）收一次数据。

六、结束语

通过认识了解 DS18B20 的原理，工作方式等，编写程序，实现 18b20 在 stm32 开发板上的驱动，并将 18b20 测量的温度通过串口传到上位机，最后通过串口调试助手显示出来。

在这个系统的设计过程中有一些不足的地方，如温度转化失败了，应该如何处理，或者是串口通信失败了，如果处理，这些都是系统不完善的地方。

希望通过以后的学习能让这个系统更加完善。