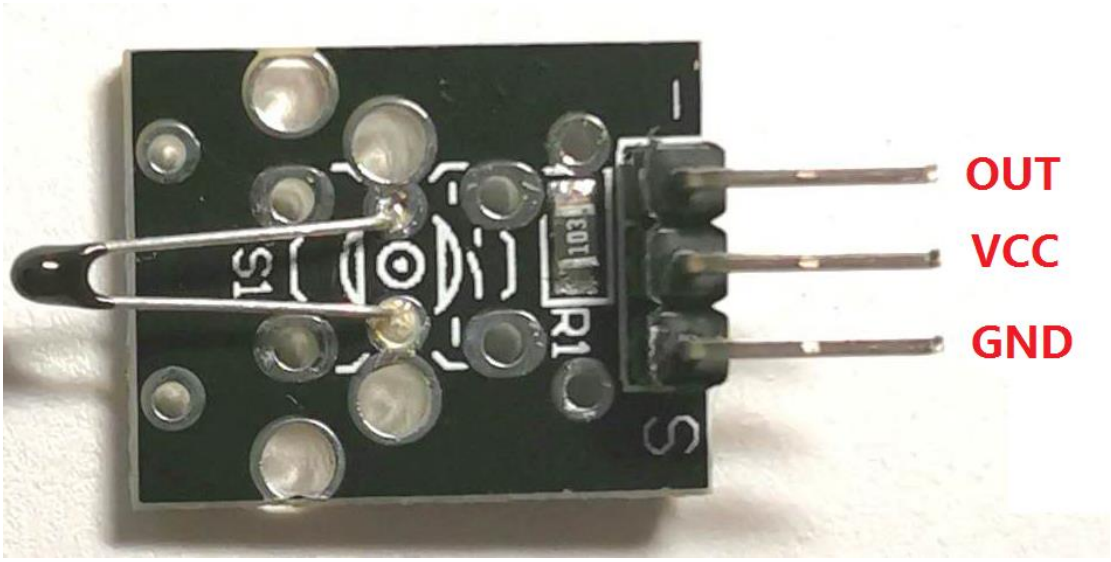
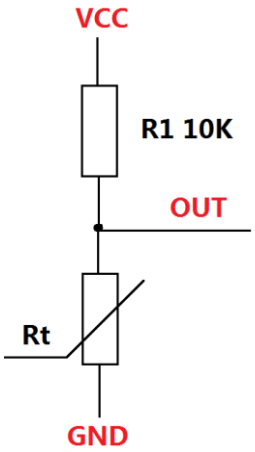
二、温度类传感器

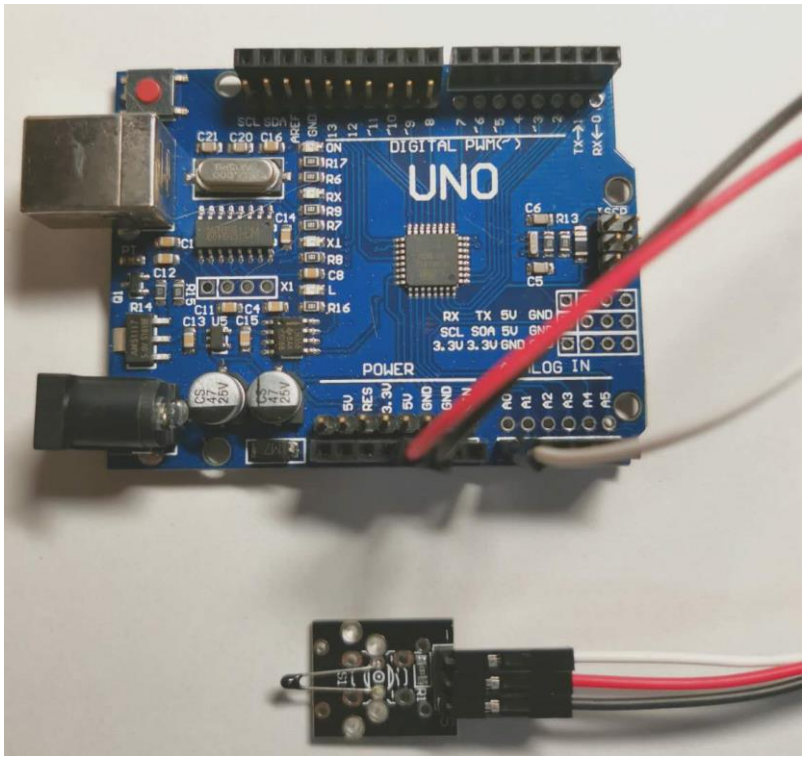
1、 模拟温度传感器模块



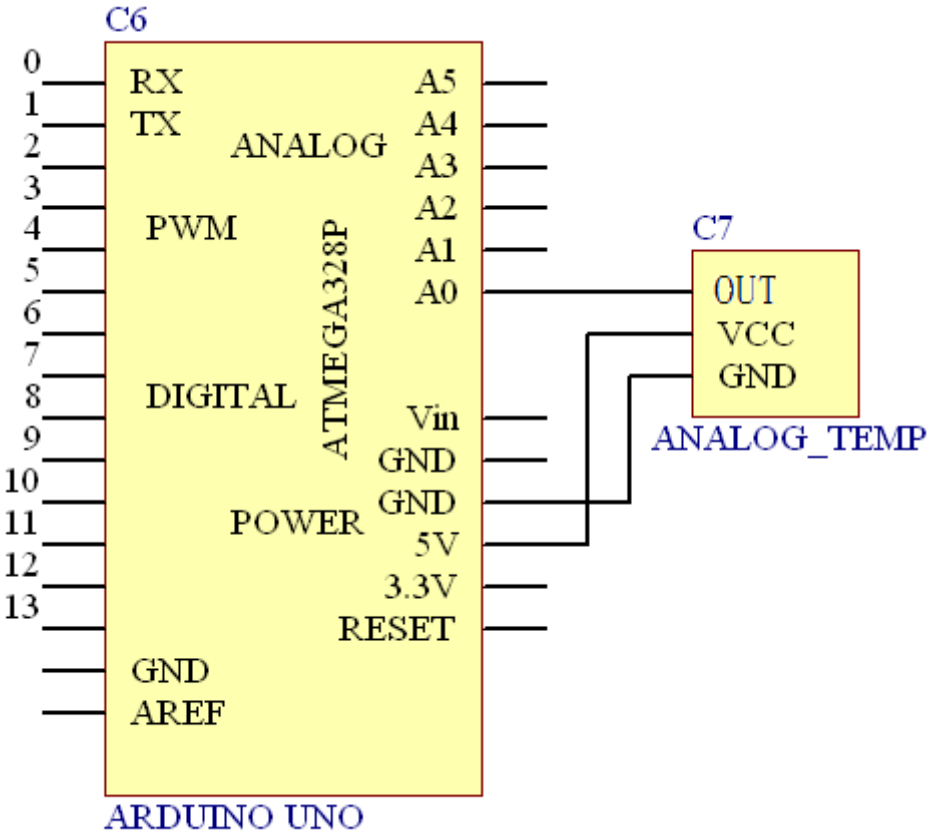
模块实物图



原理图



实物连线图（实际接线电源正负要交换）



电路原理图

（1）工作原理

热敏电阻是可变电阻，可随温度改变其电阻 R，它们按其电阻对温度变化的响应方式进行分类。有负温度系数型（NTC）和正温度系数型（PTC）。在负温度系数（NTC）热敏电阻中，电阻随温度的升高而降低，在正温度系数（PTC）热敏电阻中，电阻随温度的升高而增加。

NTC 热敏电阻是最常见的，NTC 热敏电阻由半导体材料（例如金属氧化物或陶瓷）制成，其被加热和压缩以形成温度敏感的导电材料。

常规 NTC 热敏电阻参数：温度为(R25℃)时，阻值为 10kΩ，B 值(25/85)为 3435。

B 值是热敏电阻器的材料常数，即热敏电阻器的芯片（一种半导体陶瓷）在经过高温烧结后，形成具有一定电阻率的材料，每种配方和烧结温度下只有一个 B 值，所以种之为材料常数。

B 值可以通过测量在 25 摄氏度和 50 摄氏度（或 85 摄氏度）时的电阻值后进行计算。B 值与产品电阻温度系数正相关，也就是说 B 值越大，其电阻温度系数也就越大。

温度系数就是指温度每升高 1 度，电阻值的变化率。采用以下公式可以将 B 值换算成电阻温度系数：

电阻温度系数 = B 值/T^2 （T 为要换算的点绝对温度值）

NTC 热敏电阻器的 B 值一般在 2000K - 6000K 之间，不能简单地说 B 值是越大越好还是越小越好，要看你用在什么地方。一般来说，作为温度测量、温度补偿以及抑制浪涌电阻用的产品，同样条件下是 B 值大点好。因为随着温度的变化，B 值大的产品其电阻值变化更大，也就是说更灵敏。

热敏电阻是一种具有很高电阻温度系数的非线性电阻。

（2）应用电路

由于热敏电阻是可变电阻，我们需要在计算温度之前测量电阻，但是我们不能直接测量电阻，只能测量电压。

利用分压器电路采集测量热敏电阻和已知电阻之间的电压，分压器的公式是：

$$V_{out} = V_{in} \times \left(\frac{R2}{R1+R2}\right)$$ （式子 1-1）

就热敏电阻电路中的分压器而言，上述等式中的变量为：

V_{out} : $Voltage\ between\ thermistor\ and\ known\ resistor$
V_{in} : $V_{cc},\ i.e.\ 5V$
$R1$: $Known\ resistor\ value$
$R2$: $Resistance\ of\ thermistor$

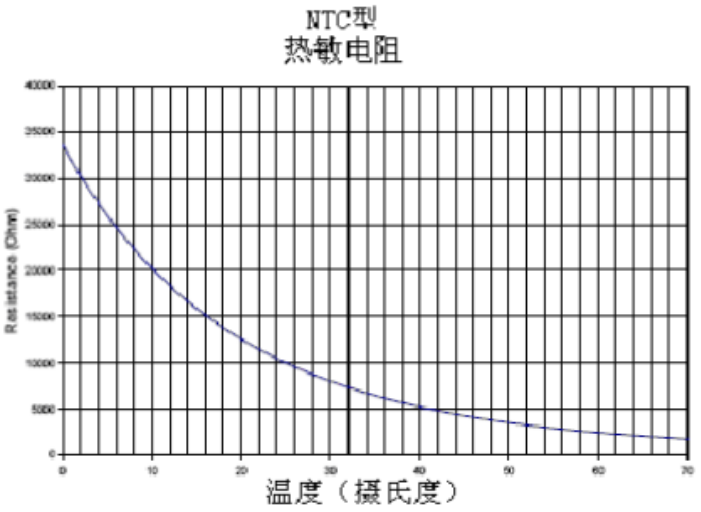
这个等式可以重新排列和简化，以求解 R2，即热敏电阻的电阻：

$$R2 = R1 \times \left(\frac{V_{in}}{V_{out}} - 1\right)$$ （式子 1-2）

热敏电阻的阻值与所测量的温度对应关系如下图所示：

抗性倍数 10k

温度 (℃)	Rt/R25 标称值
0	3.363000
5	2.599000
10	2.024000
15	1.589000
20	1.256000
25	1.000000
30	0.801300
35	0.646100
40	0.524100
45	0.427600
50	0.350700
55	0.289400
60	0.240000
65	0.200100
70	0.167700
75	0.141200
80	0.119400
85	0.101400
90	0.086520
95	0.074090
100	0.063700



Steinhart-Hart 方程是一个经验公式，用来描述 NTC 型热敏电阻的阻值与温度关系的数学表达式。

$$\frac{1}{T} = a + b(\ln R) + c(\ln R)^3$$

其中 T 是开氏温标（273.15K 为摄氏温度零点），R 是阻值，单位欧姆；系数 a，b，c 是常数。正常情况下是通过测量三组温度下的热敏电阻的阻值，然后求解得到 a,b,c 的值。

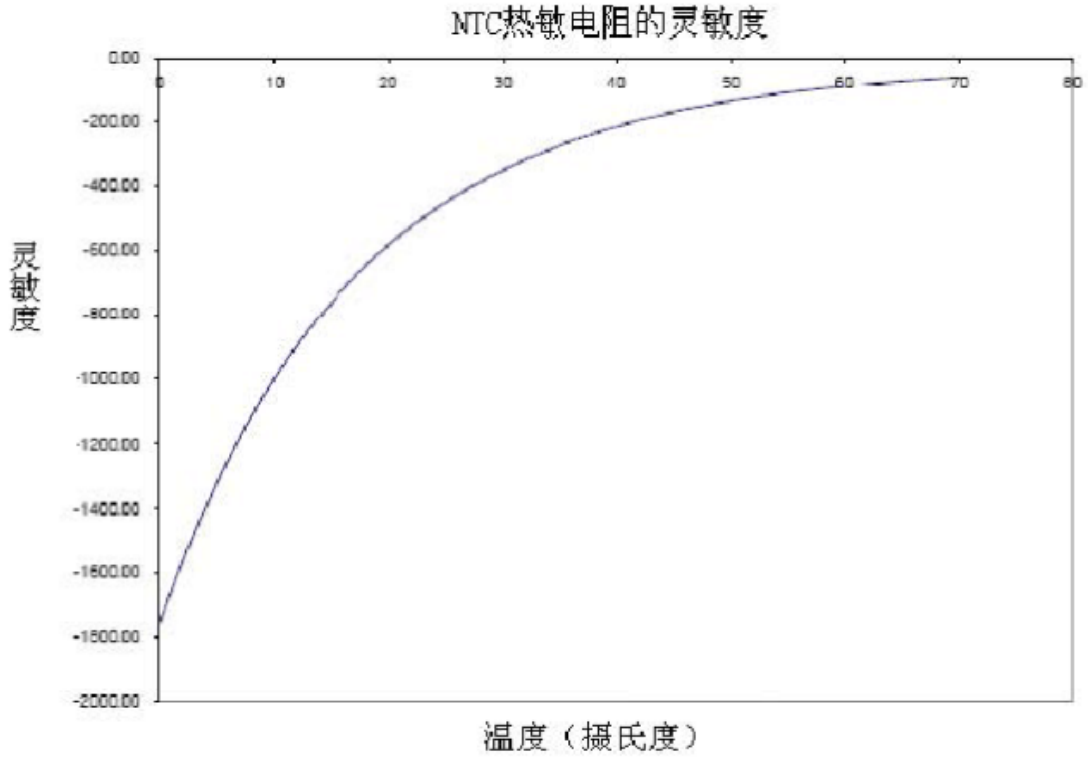
温度范围在 0 到 50 摄氏度之间时，实验室测得的三个系数为：

$$a = 1.1869 \times 10^{-3}$$

$$b = 2.2790 \times 10^{-4}$$

$$c = 8.7000 \times 10^{-8}$$

热敏电阻的主要缺点是它的非线性。另外，热敏电阻的灵敏度随温度的不同而不同。



从中可以发现，灵敏度随温度上升而下降。这就是它测温范围小的原因。

摄氏温度和开氏温度换算：

t℃ = Tk - 273.15

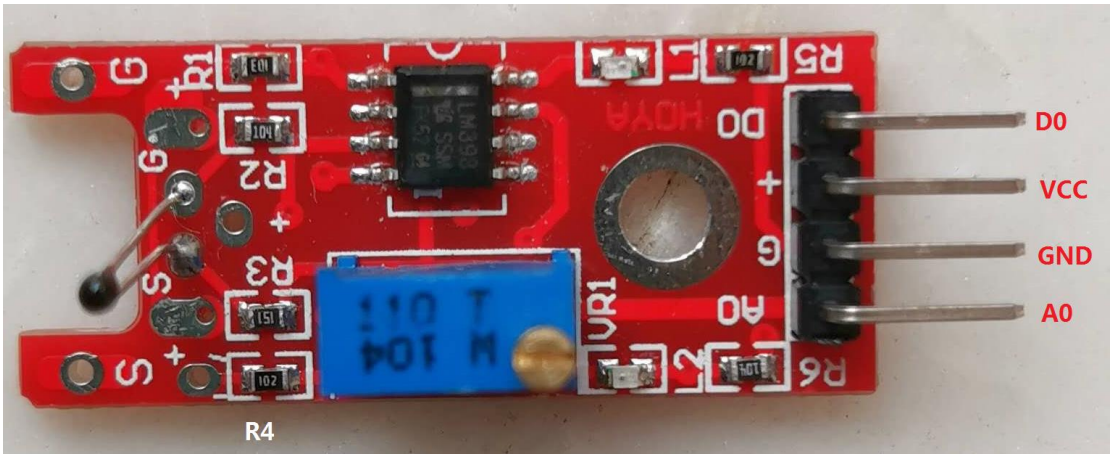
（3）程序：

```
#include <math.h>
double Thermister(int RawADC) {
  double Temp;
  Temp = log((10240000/RawADC) - 10000); //lnR，式子 1-2 求 R
  //c 语言里面只有两个函数 log 和 log10，其中函数 log(x) 表示是以 e 为底的自然对数，即 ln(x)函数。
  //log10(x) 以 10 为底的对数，即 lg(x)。
  Temp = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * Temp * Temp ))* Temp );
  //1/(a+b*lnR+c*lnR*lnR*lnR)
  Temp = Temp - 273.15;            // Convert Kelvin to Celcius
  return Temp;
}
void setup() {
  Serial.begin(9600);
}
void loop() {
  Serial.println(analogRead(0));
  Serial.print(Thermister(analogRead(0)));  //display Temperature ，
```

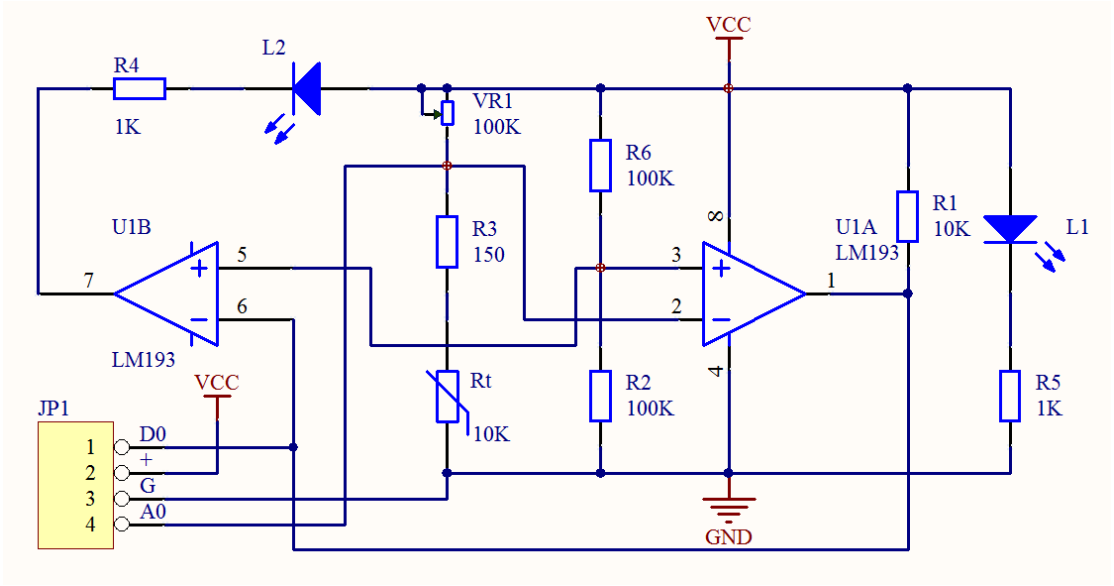
接线不调换的话修改 1024-analogRead(0)

```
  Serial.println("c");
  delay(500);
}
```

2、 数字温度传感器模块（其实也是模拟温度传感器）



实物图



电路原理图

从原理图中可以看出，D0 引脚输出是开关量，当测量温度高于某一个值时，U1A-1 输出高电平，此时 U1B-7 脚输出低电平，L2 指示灯亮；

TDC393 是由两个独立的、高精度电压比较器组成的集成电路，失调电压低,最大为 2.0mV。 它专为获得宽电压范围、单电源供电而设计，也可以以双电源供电；而且无论电源电压大小，电源消耗的电流都很低。它还有一个特性：即使是单电源供电，比较器的共模输入电压范围接近地电平。

LM393 电路的特点如下：

工作温度范围:0°C -- +70°C；

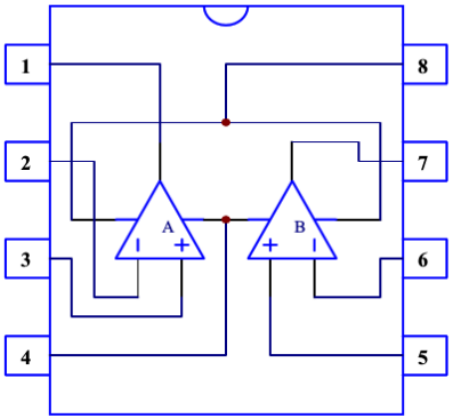
工作电源电压范围宽，单电源、双电源均可工作，单电源：2～ 36V， 双电源：±1～±18V；

消耗电流小， ICC=0.4mA；

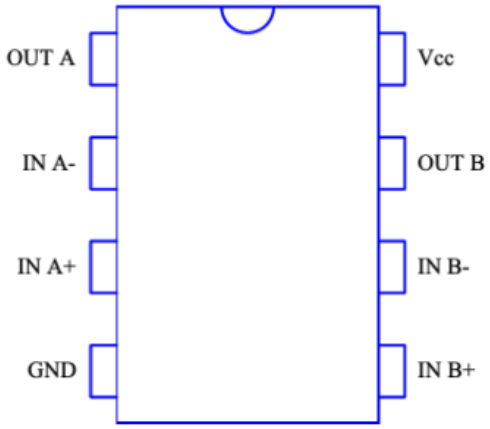
输入失调电压小， VIO=±2mV；

共模输入电压范围宽， VIC=0～VCC-1.5V；

输出与 TTL，DTL，MOS，CMOS 等兼容；



LM393 功能框图



LM393 引脚排列

LM393 引脚功能：

引出端序号	符号	功能
1	OUT A	输出 A
2	IN A-	反相输入 A
3	IN A+	同相输入 A
4	GND	接地端
5	IN B+	同相输入 B
6	IN B-	反相输入 B
7	OUT B	输出 B
8	Vcc	电源电压

程序同第一个模拟温度传感器一样。

3、 温度传感器 18B20

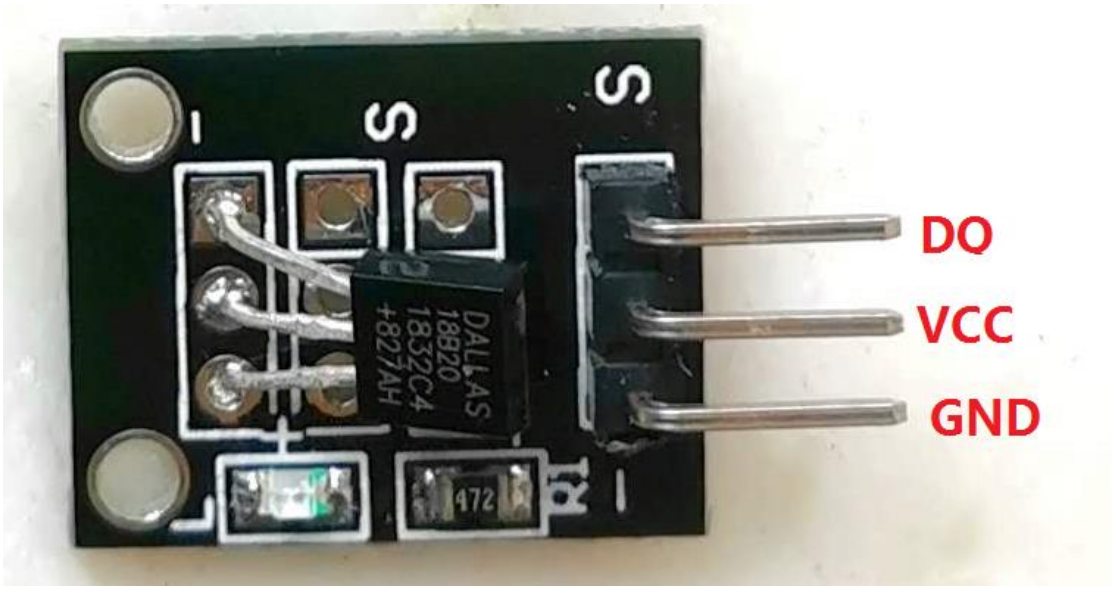


图 3.1 18B20 实物图

DS18B20 是世界第一片支持"单总线"接口的温度传感器，单总线独特而且经济的特点，使用户可轻松组建传感器网络。

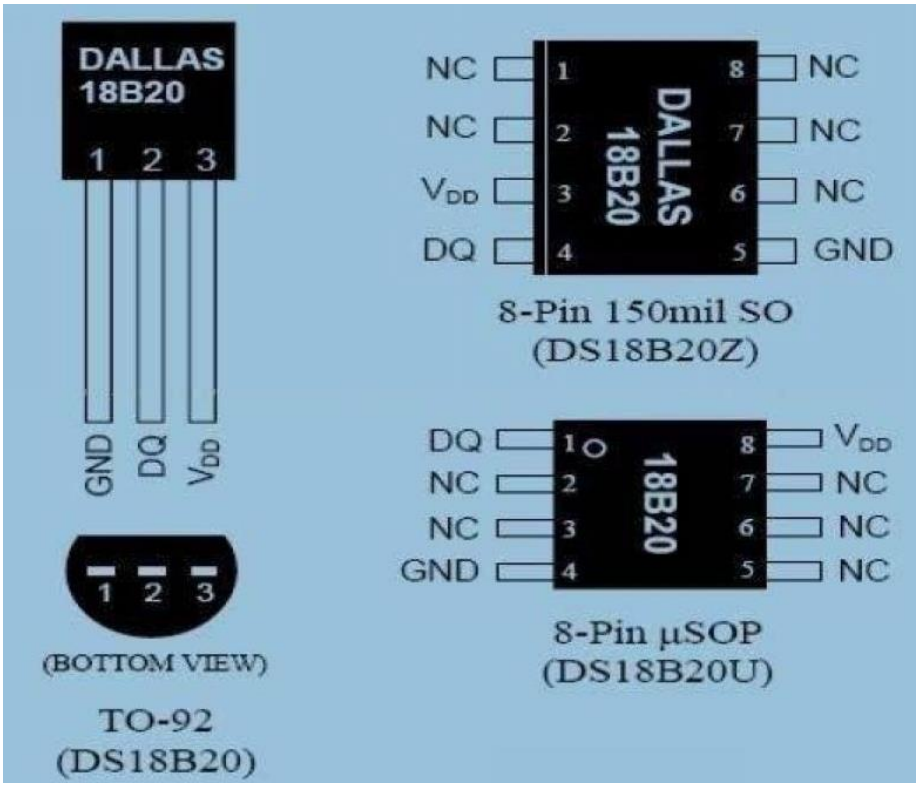


图 3.2 18B20 封装

DS18B20 的主要特性

※ 适应电压范围更宽，电压范围：3.0～5.5V，在寄生电源方式下可由数据线供电；

※ 独特的单线接口方式，DS18B20 在与微处理器连接时仅需要一条口线即可实现微处理器与 DS18B20 的双向通讯；

※ DS18B20 支持多点组网功能，多个 DS18B20 可以并联在唯一的三线上，实现组网多点测温；

※ DS18B20 在使用中不需要任何外围元件，全部传感元件及转换电路集成在形如一只三极管的集成电路内；

※ 温范围 - 55℃～ + 125℃，在-10～+85℃时精度为±0.5℃；

※ 可编程的分辨率为 9～12 位，对应的可分辨温度分别为 0.5℃、0.25℃、0.125℃和 0.0625℃，可实现高精度测温；

※ 在 9 位分辨率时最多在 93.75ms 内把温度转换为数字，12 位分辨率时最多在 750ms 内把温度值转换为数字；

※ 测量结果直接输出数字温度信号，以"一线总线"串行传送给 CPU，同时可传送 CRC 校验码，具有极强的抗干扰纠错能力；

※ 负压特性：电源极性接反时，芯片不会因发热而烧毁，但不能正常工作。

DS18B20 的内部结构

DS18B20 内部结构主要由四部分组成：64 位光刻 ROM、温度灵敏元件、内部存储器和配置寄存器。

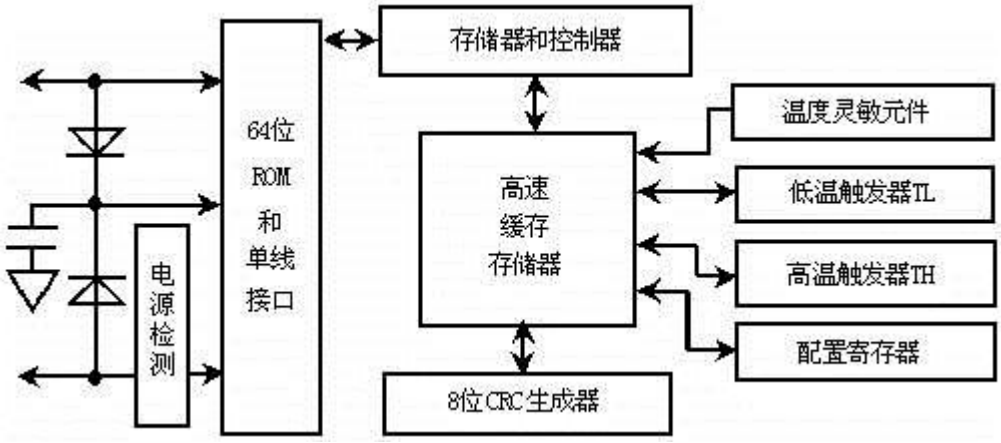


图 3.3 DS18B20 内部结构图

DS18B20 工作原理

DS18B20 测温原理如图 3.4 所示。图中低温度系数振荡器的振荡频率受温度影响很小，用于产生固定频率的脉冲信号送给计数器 1。高温度系数振荡器随温度变化其振荡率明显改变，所产生的信号作为计数器 2 的脉冲输入。计数器 1 和温度寄存器被预置在 - 55℃所对应的一个基数值。计数器 1 对低温度系数振荡器产生的脉冲信号进行减法计数，当计数器 1 的预置值减到 0 时，温度寄存器的值将加 1，计数器 1 的预置将重新被装入，计数器 1 重新开始对低温度系数振荡器产生的脉冲信号进行计数，如此循环直到计数器 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。图 3.4 中的斜率累加器用于补偿和修正测温过程中的非线性，其输出用于修正计数器 1 的预置值。

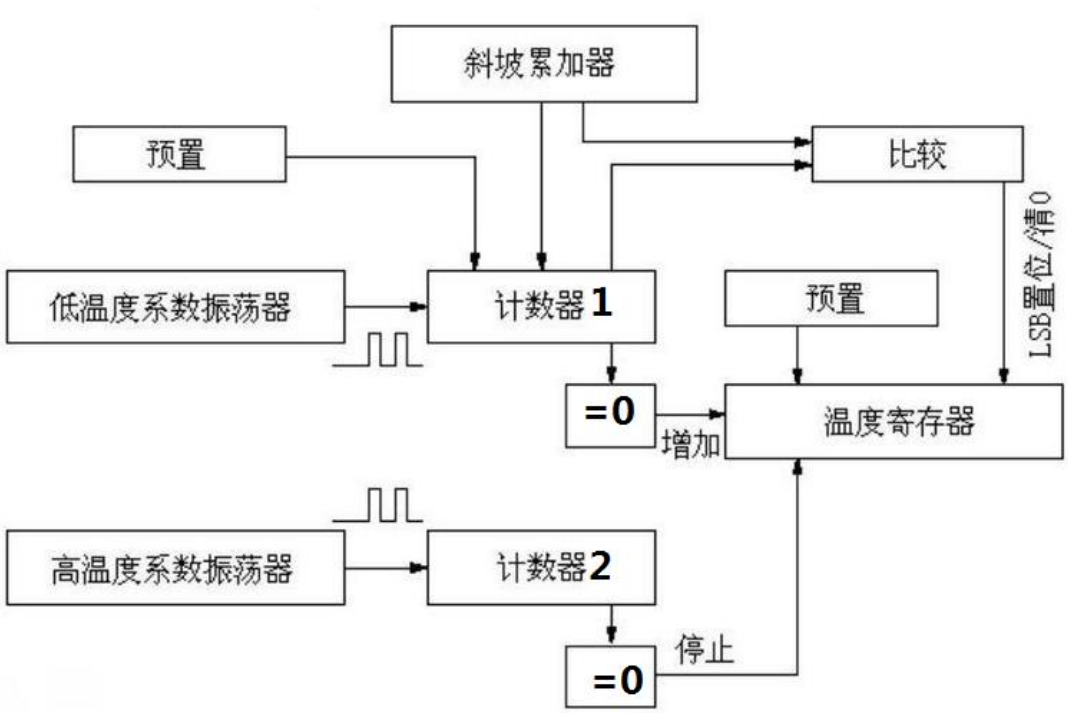


图 3.4 DS18B20 测温原理框图

DS18B20 有 4 个主要的数据部件：

（1）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（28H）是产品类型标号，接着的 48 位是该 DS18B20 自身的序列号，最后 8 位是前面 56 位的循环冗余校验码（CRC=X8+X5+X4+1）。光刻 ROM 的作用是使每一个 DS18B20 都各不相同，这样就可以实现一根总线上挂接多个 DS18B20 的目的。

（2）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。

表 1: DS18B20 温度值格式表

	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

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以 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** 再乘以 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 FF6FH，-55℃的数字输出为 FC90H。

表 2: DS18B20 温度数据表

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C	0000 0111 1101 0000	07D0h
+85°C*	0000 0101 0101 0000	0550h
+25.0625°C	0000 0001 1001 0001	0191h
+10.125°C	0000 0000 1010 0010	00A2h
+0.5°C	0000 0000 0000 1000	0008h
0°C	0000 0000 0000 0000	0000h
-0.5°C	1111 1111 1111 1000	FFF8h
-10.125°C	1111 1111 0101 1110	FF5Eh
-25.0625°C	1111 1110 0110 1111	FE6Fh
-55°C	1111 1100 1001 0000	FC90h

*The power-on reset value of the temperature register is +85°C

（3）DS18B20 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 EEPRAM，后者存放高温度和低温度触发器 TH、TL 和结构寄存器。

（4）配置寄存器 该字节各位的意义如下：

表 3： 配置寄存器结构

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，如下表所示：（DS18B20 出厂时被设置为 12 位）

表 4： 温度分辨率设置表

R1	R0	分辨率	温度最大转换时间
0	0	9 位	93.75ms
0	1	10 位	187.5ms
1	0	11 位	375ms
1	1	12 位	750ms

高速暂存存储器

高速暂存存储器由 9 个字节组成，其分配如表 5 所示。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。单片机可通过单线接口读到该数据，读取时低位在前，高位在后，数据格式如表 1 所示。对应的温度计算：当符号位 S=0 时，直接将二进制位转换为十进制；当 S=1 时，先将补码变为原码，再计算十进制值。表 2 是对应的一部分温度值。第九个字节是冗余检验字节。

表 5： DS18B20 暂存寄存器分布

寄存器内容	**字节地址**
温度值低位（LS Byte）	0
温度值高位（MS Byte）	1
高温限值（TH）	2

低温限值（TL）	3
配置寄存器	4
保留	5
保留	6
保留	7
CRC 校验值	8

根据 DS18B20 的通讯协议，主机（单片机）控制 DS18B20 完成温度转换必须经过三个步骤：每一次读写之前都要对 DS18B20 进行复位操作，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，当 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。

表 6： ROM 指令表

指 令	**约定代码**	**功 能**
读 ROM	33H	读 DS1820 温度传感器 ROM 中的编码（即 64 位地址）
符合 ROM	55H	发出此命令之后，接着发出 64 位 ROM 编码，访问单总线上与该编码相对应的 DS1820 使之作出响应，为下一步对该 DS1820 的读写作准备。
搜索 ROM	0FOH	用于确定挂接在同一总线上 DS1820 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 DS1820 发温度变换命令。适用于单片工作。

告警搜索命令	0ECH	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。

表 6： RAM 指令表

指 令	**约定代码**	**功 能**
温度变换	44H	启动 DS1820 进行温度转换，12 位转换时最长为 750ms（9 位为 93.75ms）。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 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	将 EEPROM 中内容恢复到 RAM 中的第 3 、4 字节。
读供电方式	0B4H	读 DS1820 的供电模式。寄生供电时 DS1820 发送 “ 0 ” ，外接电源供电 DS1820 发送 “ 1 ” 。

DS18B20 的应用电路

DS18B20 测温系统具有测温系统简单、测温精度高、连接方便、占用口线少等优点。

DS18B20 寄生电源供电方式电路图 如下图 3.5 所示，在寄生电源供电方式下，DS18B20 从单线信号线上汲取能量：在信号线 DQ 处于高电平期间把能量储存在内部电容里，在信号线处于低电平期间消耗电容上的电能工作，直到高电平到来再给寄生电源（电

容）充电。

独特的寄生电源方式有三个好处：

1）进行远距离测温时，无需本地电源

2）可以在没有常规电源的条件下读取 ROM

3）电路更加简洁，仅用一根 I/O 口实现测温

要想使 DS18B20 进行精确的温度转换，I/O 线必须保证在温度转换期间提供足够的能量，由于每个 DS18B20 在温度转换期间工作电流达到 1mA，当几个温度传感器挂在同一根 I/O 线上进行多点测温时，只靠 4.7K 上拉电阻就无法提供足够的能量，会造成无法转换温度或温度误差极大。 因此，图 3.5 电路只适应于单一温度传感器测温情况下使用，不适宜采用电池供电系统中。并且工作电源 VCC 必须保证在 5V，当电源电压下降时，寄生电源能够汲取的能量也降低，会使温度误差变大。有人做过实验，当低于 4.5V 时，测出的温度值比实际的温度高，误差较大。当电源电压降为 4V 时，温度误差有 3℃之多，这就应该是因为寄生电源汲取能量不够造成的。

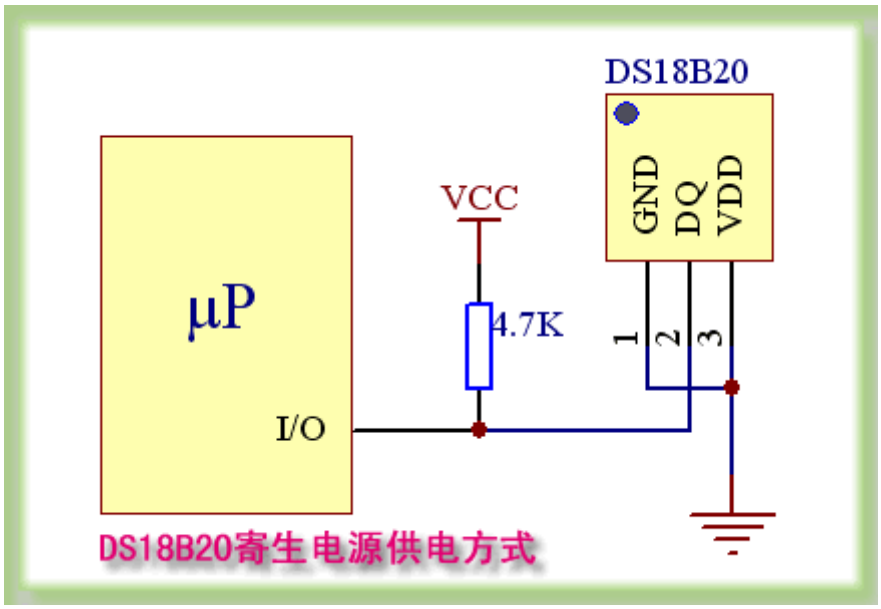


图 3.5 DS18B20 寄生电源供电方式

DS18B20 寄生电源强上拉供电方式电路图　改进的寄生电源供电方式如下面图 3.6 所示，为了使 DS18B20 在动态转换周期中获得足够的电流供应，当进行温度转换或拷贝到 E2 存储器操作时，用 MOSFET 把 I/O 线直接拉到 VCC 就可提供足够的电流，在发出任何涉及到拷贝到 E2 存储器或启动温度转换的指令后，必须在最多 10μS 内把 I/O 线转换到强上拉状态。在强上拉方式下可以解决电流供应不足的问题，因此也适合于多点测温应用，缺点就是要多占用一根 I/O 口线进行强上拉切换。

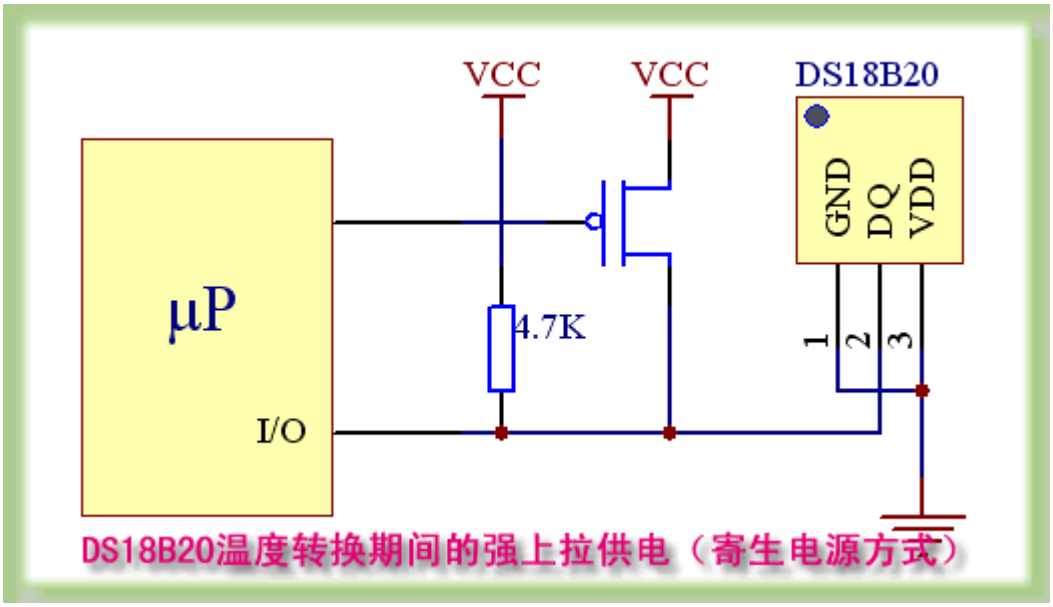


图 3.6 DS18B20 改进寄生电源供电方式

注意：在图 3.5 和图 3.6 寄生电源供电方式中，DS18B20 的 VDD 引脚必须接地

DS18B20 的外部电源供电方式　在外部电源供电方式下，DS18B20 工作电源由 VDD 引脚接入，此时 I/O 线不需要强上拉，不存在电源电流不足的问题，可以保证转换精度，同时在总线上理论可以挂接任意多个 DS18B20 传感器，组成多点测温系统。注意：在外部供电的方式下，DS18B20 的 GND 引脚不能悬空，否则不能

转换温度，读取的温度总是 85℃。

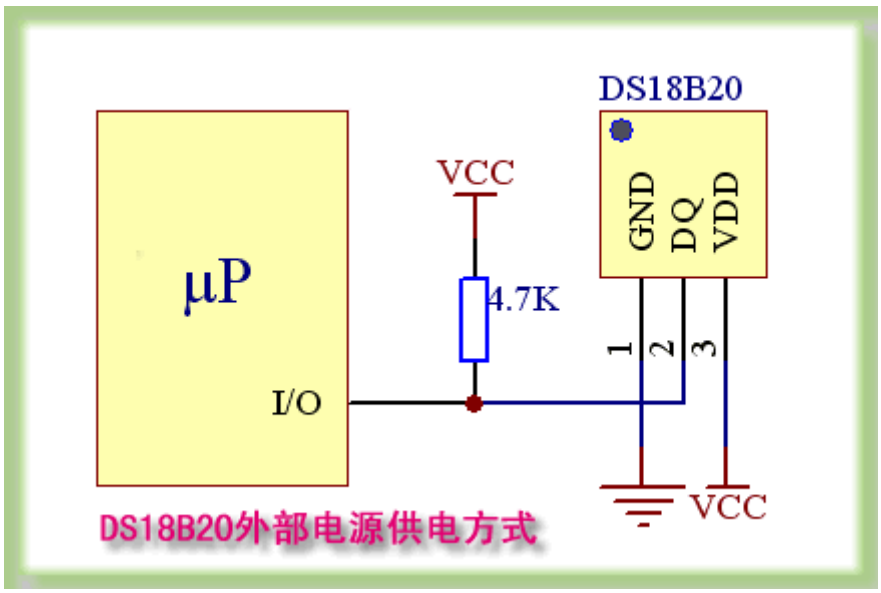


图 3.7：外部供电方式单点测温电路

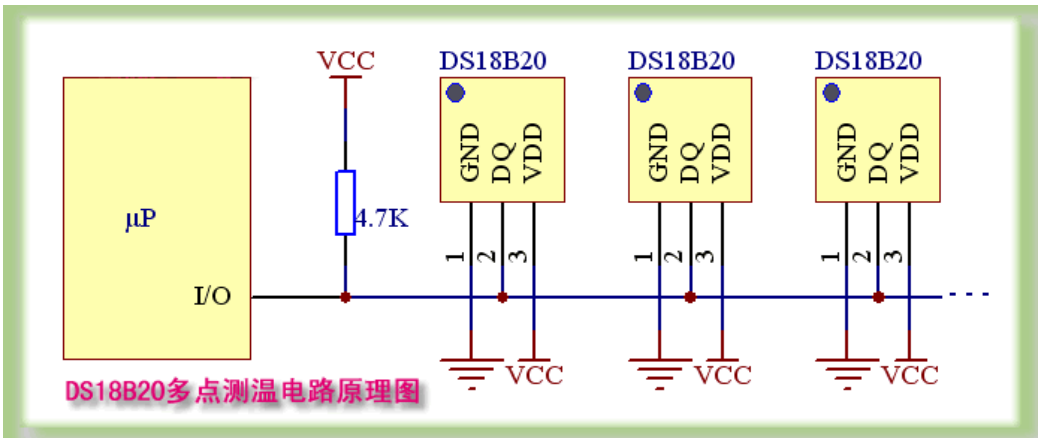


图 3.8：外部供电方式的多点测温电路图

外部电源供电方式是 DS18B20 最佳的工作方式，工作稳定可靠，抗干扰能力强，而且电路也比较简单，可以开发出稳定可靠的多点温度监控系统。推荐大家在开发中使用外部电源供电方式，毕竟比寄生电源方式只多接一根 VCC 引线。在外接电源方式下，可以充分

发挥 DS18B20 宽电源电压范围的优点，即使电源电压 VCC 降到 3V 时，依然能够保证温度量精度。

DS1820 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）、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。

2）、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。

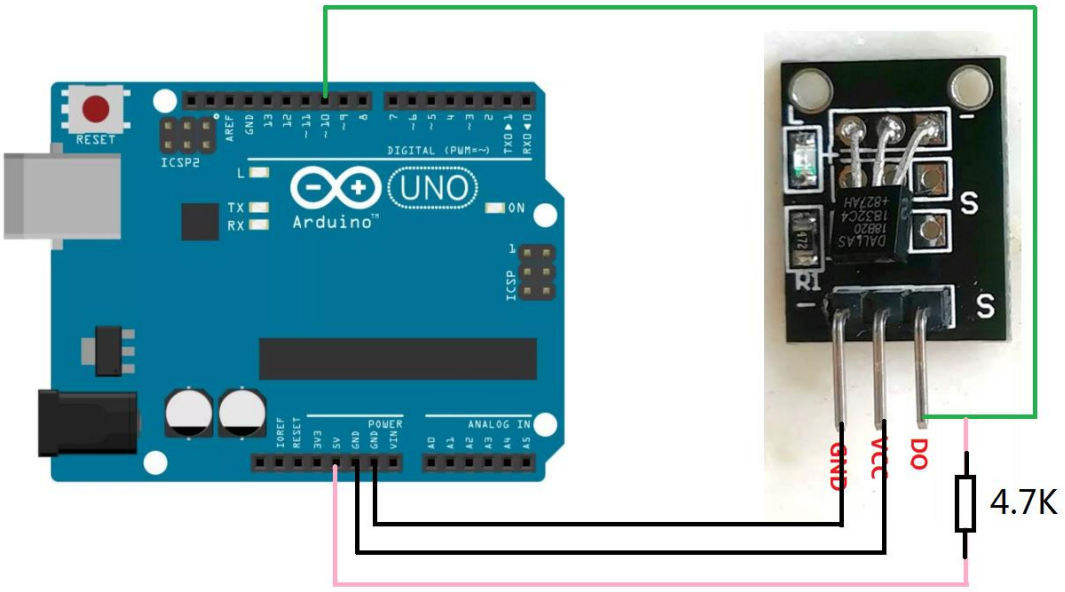
3）、连接 DS1820 的总线电缆是有长度限制的。试验中，当采用普通信号电缆传输长度超过 50m 时，读取的测温数据将发生错误。当将总线电缆改为双绞线带屏蔽电缆时，正常通讯距离可达 150m，当采用每米绞合次数更多的双绞线带屏蔽电缆时，正常通讯距离进一步加长。这种情况主要是由总线分布电容使信号波形产生畸变造成的。因此，在用 DS1820 进行长距离测温系统设计时要充分考虑总线分布电容和阻抗匹配问题。

4）、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。

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。

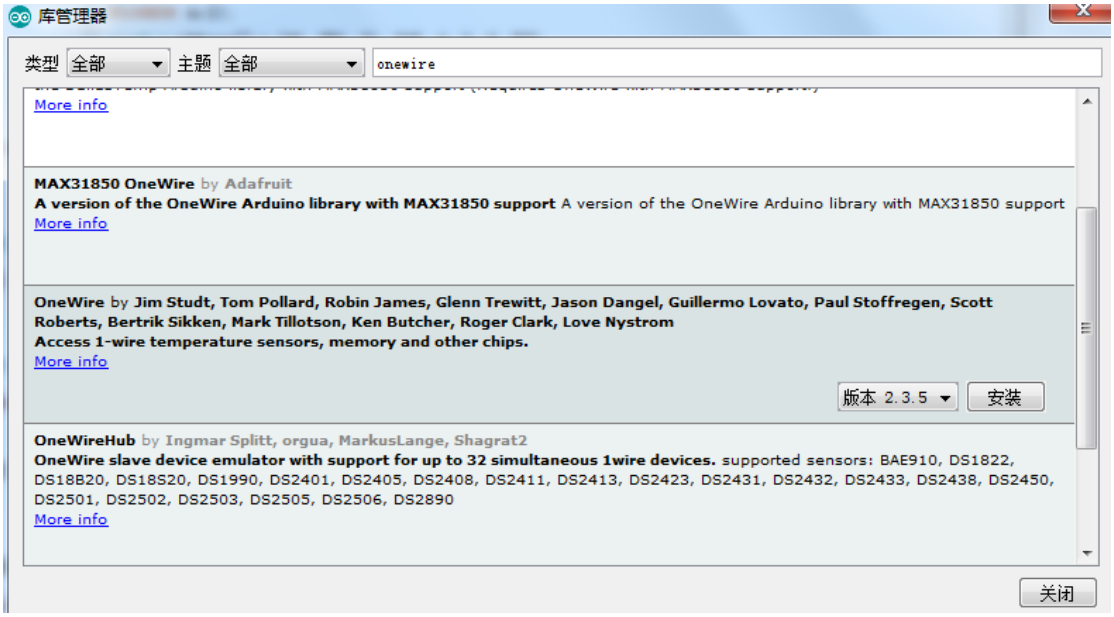
Arduino 控制接线图

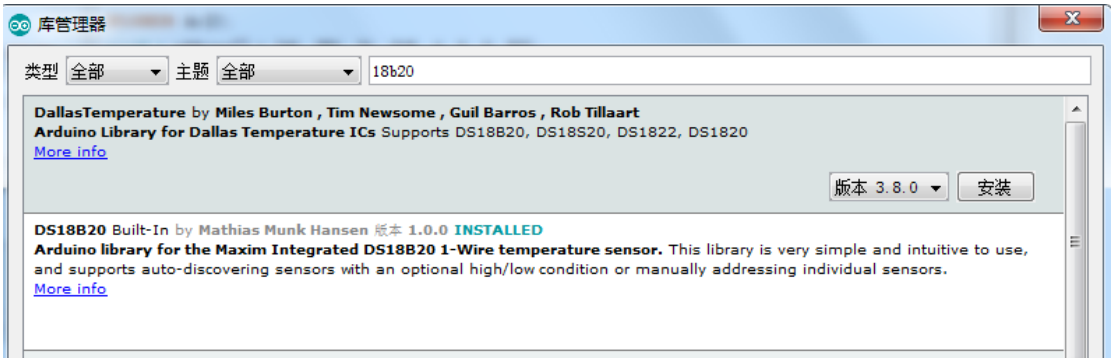


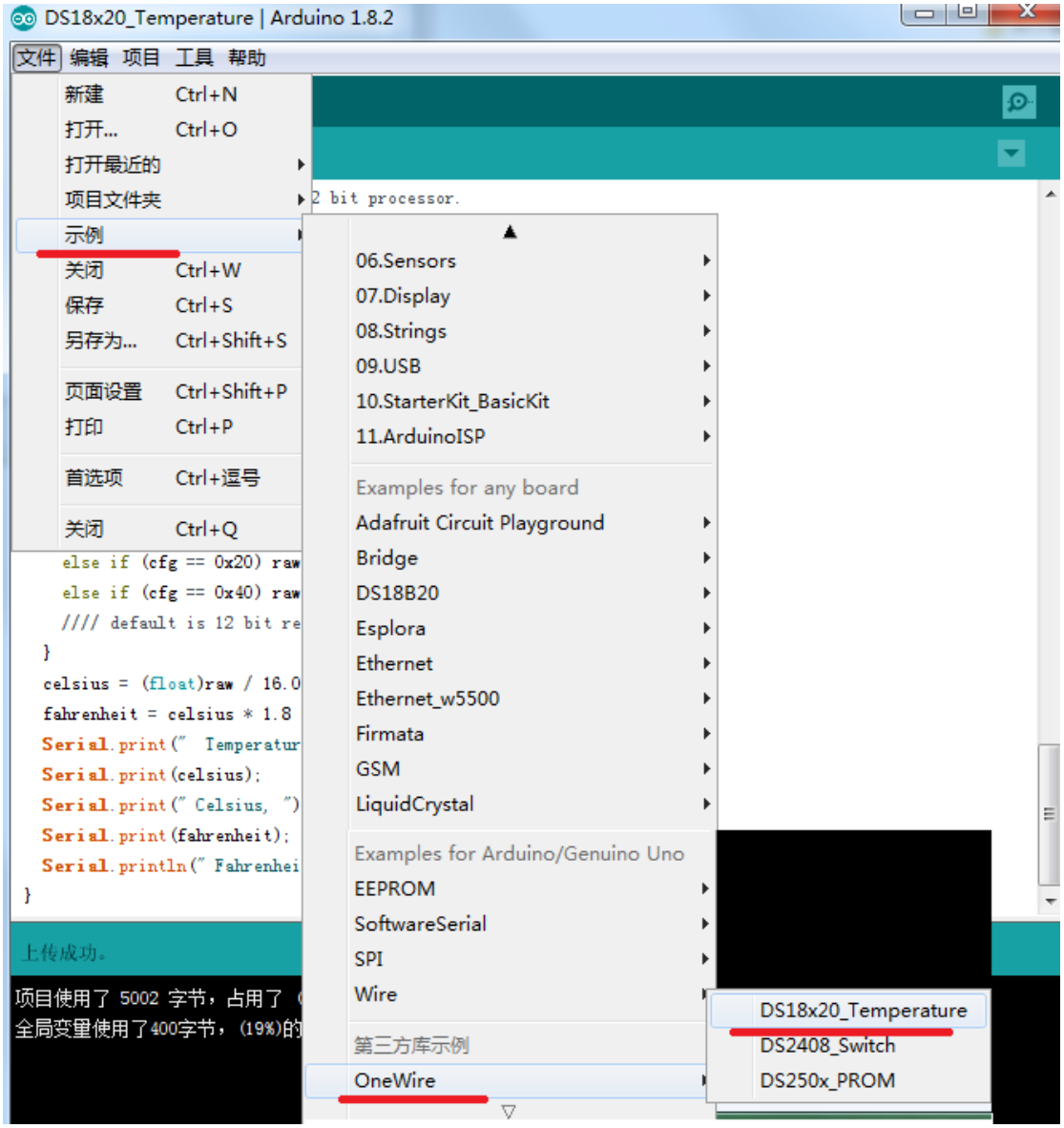
寄生供电接法（非寄生供电也可以）

Arduino 程序实现：

首先要加载库文件


库管理器
类型 全部
主题 全部
onewire
More info
MAX31850 OneWire by Adafruit
A version of the OneWire Arduino library with MAX31850 support A version of the OneWire Arduino library with MAX31850 support
More info
OneWire by Jim Studt, Tom Pollard, Robin James, Glenn Trewitt, Jason Dangel, Guillermo Lovato, Paul Stoffregen, Scott Roberts, Bertrik Sikken, Mark Tillotson, Ken Butcher, Roger Clark, Love Nystrom
Access 1-wire temperature sensors, memory and other chips.
More info
版本 2.3.5
安装
OneWireHub by Ingmar Splitt, orgua, MarkusLange, Shagrat2
OneWire slave device emulator with support for up to 32 simultaneous 1wire devices. supported sensors: BAE910, DS1822, DS18B20, DS18S20, DS1990, DS2401, DS2405, DS2408, DS2411, DS2413, DS2423, DS2431, DS2432, DS2433, DS2438, DS2450, DS2501, DS2502, DS2503, DS2505, DS2506, DS2890
More info
关闭


库管理器
类型 全部
主题 全部
18b20
DallasTemperature by Miles Burton , Tim Newsome , Guil Barros , Rob Tillaart
Arduino Library for Dallas Temperature ICs Supports DS18B20, DS18S20, DS1822, DS1820
More info
版本 3.8.0
安装
DS18B20 Built-In by Mathias Munk Hansen 版本 1.0.0 INSTALLED
Arduino library for the Maxim Integrated DS18B20 1-Wire temperature sensor. This library is very simple and intuitive to use, and supports auto-discovering sensors with an optional high/low condition or manually addressing individual sensors.
More info




```
#include <OneWire.h>

// OneWire DS18S20, DS18B20, DS1822 Temperature Example
// http://www.pjrc.com/teensy/td_libs_OneWire.html
// The DallasTemperature library can do all this work for you!
// https://github.com/milesburton/Arduino-Temperature-Control-Library

OneWire  ds(10);  // on pin 10 (a 4.7K resistor is necessary) 定义对象

void setup(void) {
  Serial.begin(9600); //初始化串口
}

void loop(void) {
  byte i;
  byte present = 0;
```

```
byte type_s;
byte data[12]; //存放 RAM 数据
byte addr[8];  //存放 64 位 ROM
float celsius, fahrenheit;

if ( !ds.search(addr)) {//搜索总线上的设备，找到返回 1，否则返回 0；
                        //找到设备后把设备序列号写入 addr 数组
  Serial.println("No more addresses.");
  Serial.println();
  ds.reset_search();  //复位，为下一次搜索做准备
  delay(250);
  return;
}

Serial.print("ROM =");  //显示光刻 ROM 中的 64 位序列号
for( i = 0; i < 8; i++) {
  Serial.write(' ');
  Serial.print(addr[i], HEX);
}

if (OneWire::crc8(addr, 7) != addr[7]) {//验证校验码
    Serial.println("CRC is not valid!");
    return;
}
Serial.println();

// the first ROM byte indicates which chip 开始 8 位是产品类型标号
switch (addr[0]) {
  case 0x10:
    Serial.println("  Chip = DS18S20");  // or old DS1820
    type_s = 1;//当 S=1 时，先将补码变为原码，再计算十进制值。
    break;
  case 0x28:
    Serial.println("  Chip = DS18B20");
    type_s = 0;//当符号位 S=0 时，直接将二进制位转换为十进制；
    break;
  case 0x22:
    Serial.println("  Chip = DS1822");
    type_s = 0;
    break;
  default:
    Serial.println("Device is not a DS18x20 family device.");
    return;
}

  ds.reset();     //复位，等待最多 250us 看总线是否拉高，否则可能是断线或短路，正
常响应返回 1
```

ds.select(addr); //访问单总线上与该编码相对应的 DS1820 使之作出响应，为下一步对该 DS1820 的读写作准备。

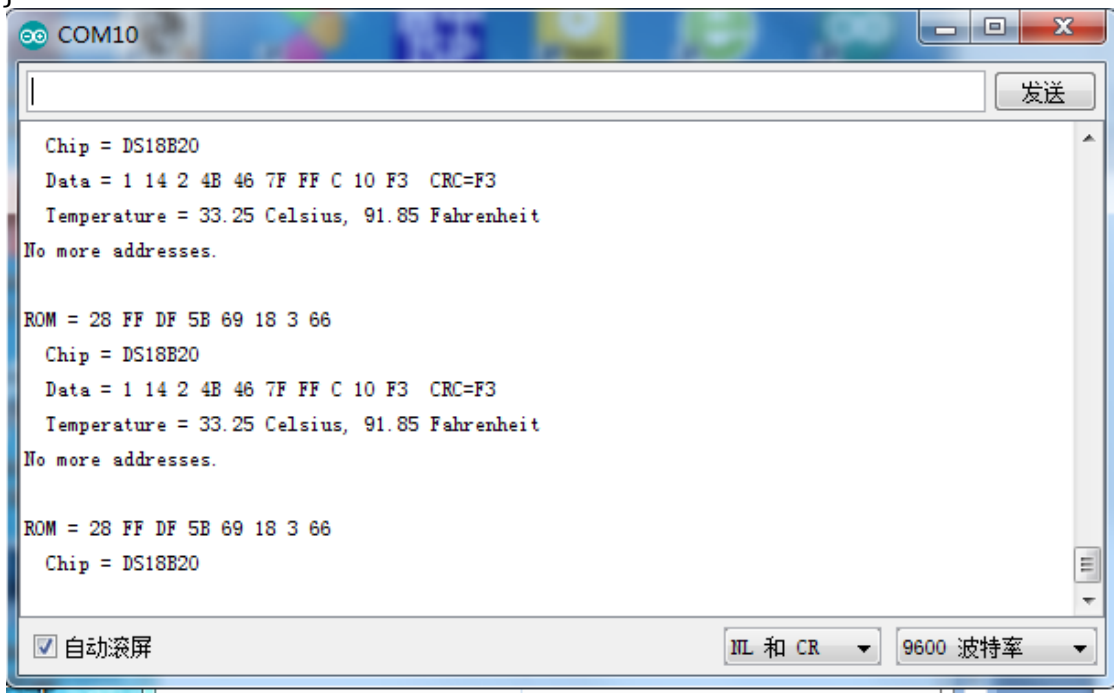
ds.write(0x44, 1);//启动 DS1820 进行温度转换，12 位转换时最长为 750ms（9 位为 93.75ms），结果存入内部 9 字节 RAM 中。采用寄生供电，就是使用总线供电

```
delay(1000);     // maybe 750ms is enough, maybe not
// we might do a ds.depower() here, but the reset will take care of it.

present = ds.reset();//存在设备
ds.select(addr);     //访问设备
ds.write(0xBE);      //读内部 RAM 中 9 字节的内容

Serial.print("  Data = ");
Serial.print(present, HEX);
Serial.print(" ");
for ( i = 0; i < 9; i++) {
  data[i] = ds.read();  //每次读一个字节
  Serial.print(data[i], HEX);
  Serial.print(" ");
}
Serial.print(" CRC=");
Serial.print(OneWire::crc8(data, 8), HEX);
Serial.println();

// Convert the data to actual temperature
// because the result is a 16 bit signed integer, it should
// be stored to an "int16_t" type, which is always 16 bits
// even when compiled on a 32 bit processor.
```

int16_t raw = (data[1] << 8) | data[0]; //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。

```
if (type_s) { //当 S=1 时，先将补码变为原码（取反加一），再计算十进制值。
   //raw = ~raw+1;
   raw = raw << 3; // 9 bit resolution default  18S20
  if (data[7] == 0x10) {
    // "count remain" gives full 12 bit resolution
    raw = (raw & 0xFFF0) + 12 - data[6];
  }
}
else {      //当符号位 S=0 时，直接将二进制位转换为十进制；
  byte cfg = (data[4] & 0x60);//配置寄存器（0 R1 R0 1 1 1 1 1）R1R0 位的组合决定转换分别率
  // at lower res, the low bits are undefined, so let's zero them
  if (cfg == 0x00) raw = raw & ~7;  // 9 bit resolution, 93.75 ms
  else if (cfg == 0x20) raw = raw & ~3; // 10 bit res, 187.5 ms，
  else if (cfg == 0x40) raw = raw & ~1; // 11 bit res, 375 ms，bit 0 is undefined
```

```
    //// default is 12 bit resolution, 750 ms conversion time
  }
  celsius = (float)raw / 16.0;   //原始数据*0.0625=实际温度
  fahrenheit = celsius * 1.8 + 32.0;
  Serial.print("  Temperature = ");
  Serial.print(celsius);
  Serial.print(" Celsius, ");
  Serial.print(fahrenheit);
  Serial.println(" Fahrenheit");
}
```

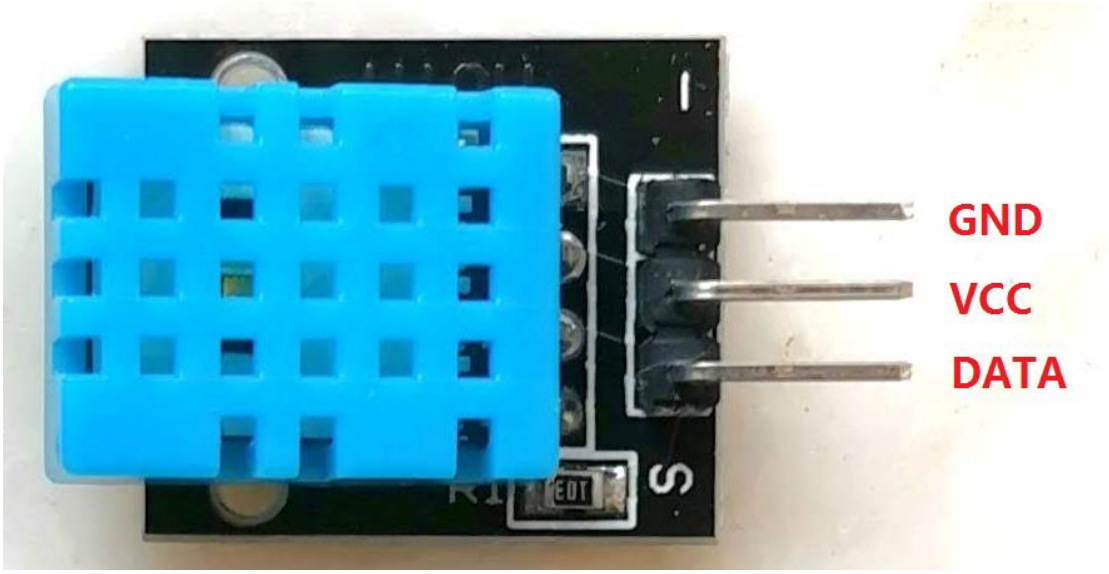


Arduino 串口监视器输出

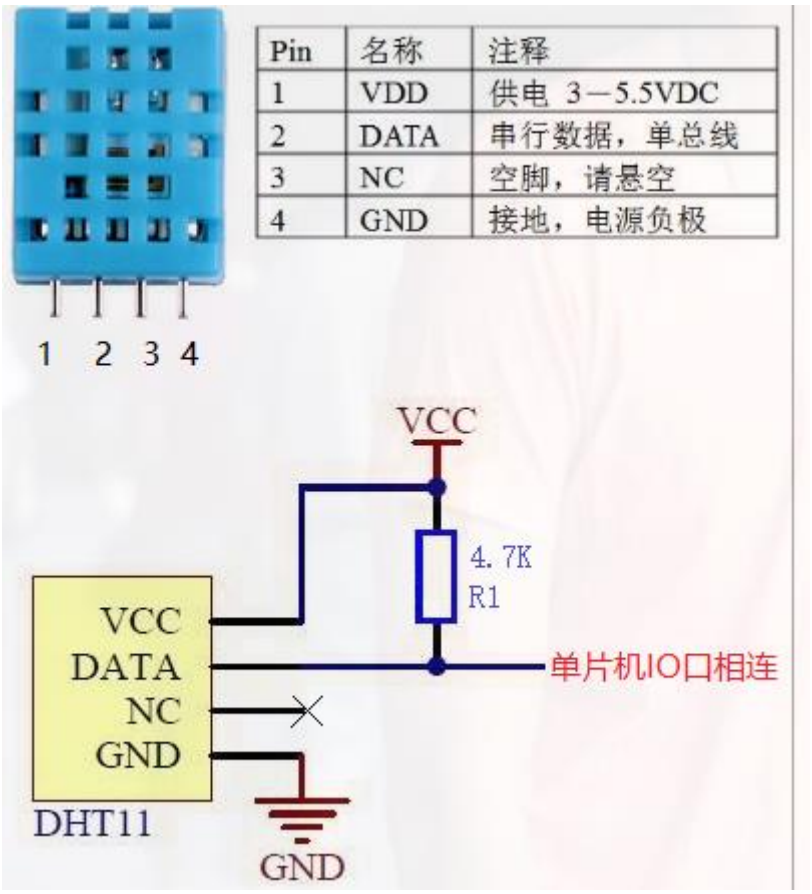
参考资料：在 Arduino 中使用 DS18B20 温度传感器

https://blog.csdn.net/Naisu_kun/article/details/88420357

4、温湿度传感器 DHT11：

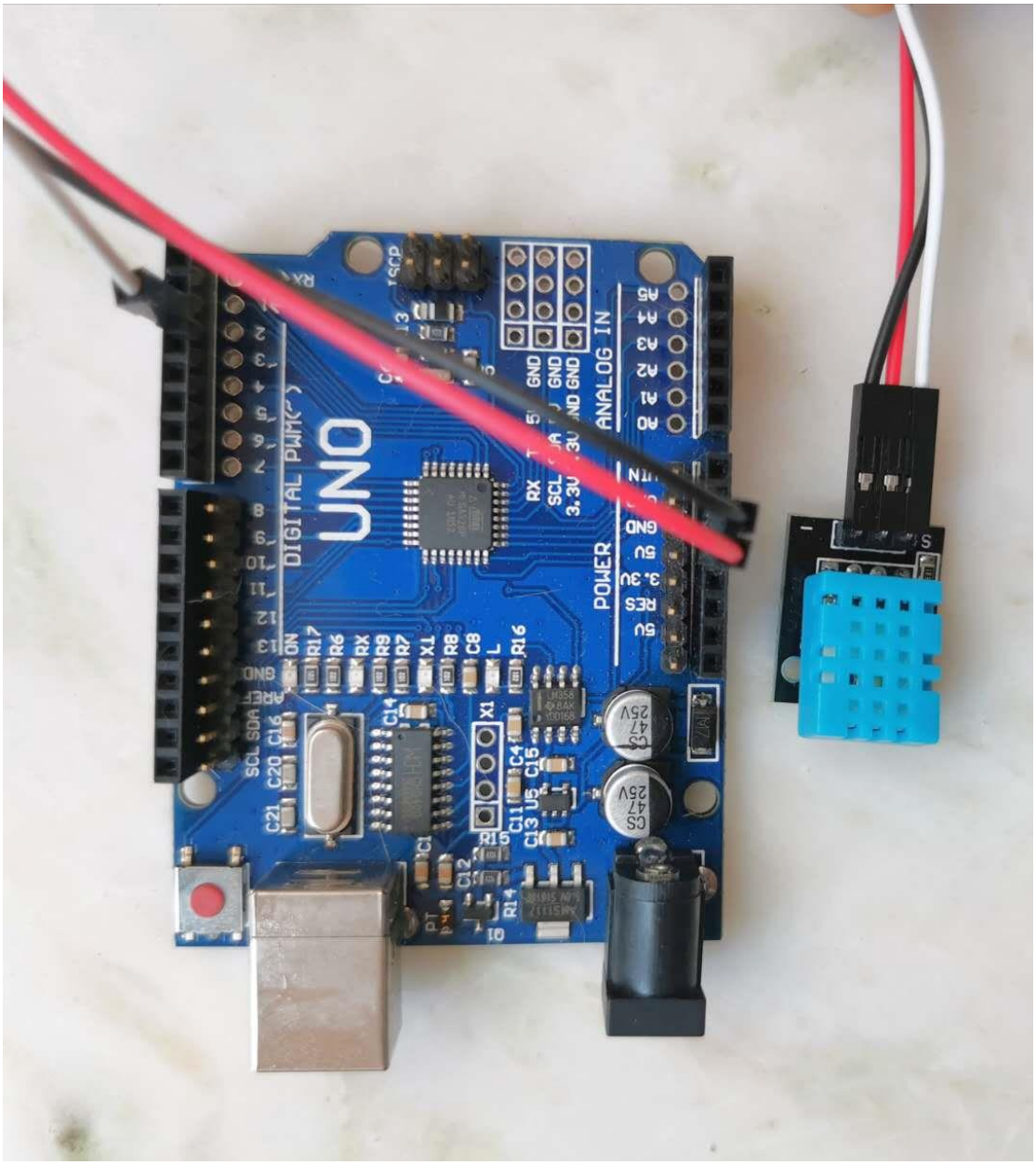


温湿度传感器模块实物图



Pin	名称	注释
1	VDD	供电 3—5.5VDC
2	DATA	串行数据，单总线
3	NC	空脚，请悬空
4	GND	接地，电源负极

DHT11 引脚分布及接线示意图



Arduino 开发板与 DHT11 温湿度模块接线图

DHT11 数字温湿度传感器是一款含有已校准数字信号输出的温湿度复合传感器，它应用专用的数字模块采集技术和温湿度传感技术，确保产品具有极高的可靠性和卓越的长期稳定性。该产品具有品质卓越、超快响应、抗干扰能力强、性价比极高等优点。单线制串行接口，使系统集成变得简易快捷。超小的体积、极低的功耗，信号传输距离可达 20 米以上，使其成为该类应用甚至最为苛刻的应用场合的最佳选择。产品为 4 针单排引脚封装，连接方便。

技术参数 ：

供电电压： 3.3~5.5V DC

输 出： 单总线数字信号

测量范围： 湿度 20-90%RH， 温度 0~50℃

测量精度： 湿度+-5%RH， 温度+-2℃

分 辨 率： 湿度 1%RH， 温度 1℃

长期稳定性： <±1%RH/年

DHT11 的工作原理

DHT11 使用单一总线通信，即 DATA 引脚。总线总是处于**空闲状态**和**通信状态**这个 2 个状态之间。

当单片机没有与 DHT11 交互时，总线处于空闲状态，在上拉电阻的作用下，处于高电平状态。

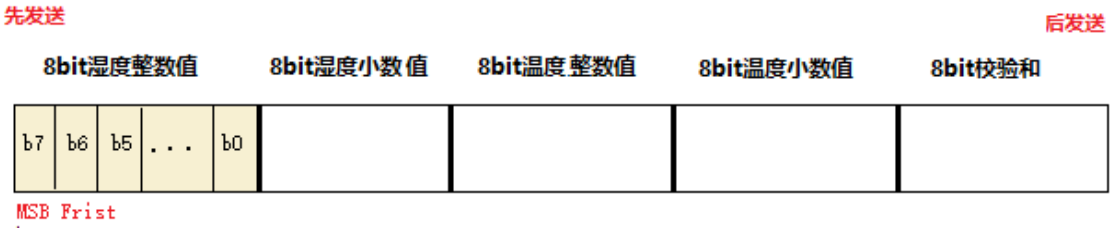
当单片机和 DHT11 正在通信时，总线处于通信状态，一次完整的通信过程如下：

①单片机将驱动总线的 IO 配置为输出模式。准备向 DHT11 发送数据。

②单片机将总线拉低至少 18ms，以此来发送起始信号。再将总线拉高并延时 20~40us，以此来代表起始信号结束。

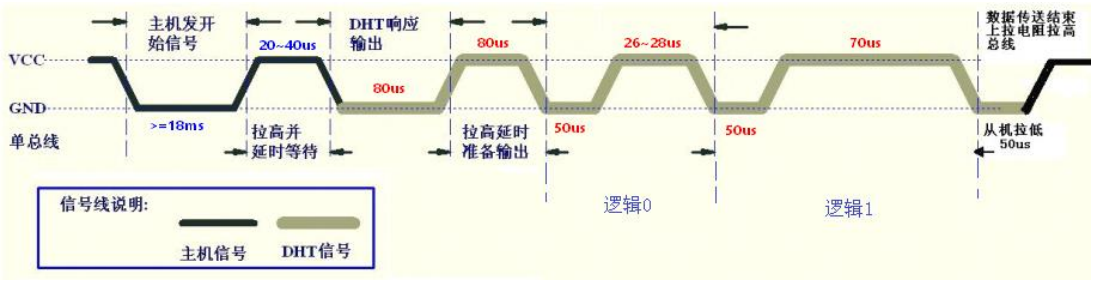
③单片机将驱动总线的 IO 配置为输入模式，准备接收 DHT11 回传的数据。

④当 DHT11 检测倒单片机发送的起始信号后，就开始应答，回传采集到的传感器数据。DHT11 先将总线拉低 80us 作为对单片机的应答（ACK），然后接着将总线拉高 80us，准备回传采集到的温湿度数据。温湿度数据以固定的帧格式发送，具体格式如下图：



可以发现一帧为 40 个 bit，而每一个 bit 的传输时序逻辑为：每一个 bit 都以 50us 的低电平（DHT11 将总线拉低）为先导，然后紧接着 DHT11 拉高总线，如果这个高电平持续时间为 26~28us，则代表逻辑 0，如果持续 70us 则代表逻辑 1。

⑤当一帧数据传输完成后，DHT11 释放总线，总线在上拉电阻的作用下再次恢复到高电平状态。



注意事项：

1、DHT11 上电后，要等待 1 秒 以越过不稳定状态，在此期间不能发送任何指令。

2、DHT11 属于低速传感器，两次通信请求之间的间隔时间不能太短，一般来说要不能低于 1 秒。

3、当前 DHT11 通信帧的小数部分默认都是 0，厂商预留给以后实现。所以一般只读取整数值部分即可。校验和定义为：前 4 个 Byte 的总和的低 8 位。

Arduino 程序:

```
// Example testing sketch for various DHT humidity/temperature sensors
// Written by ladyada, public domain

// REQUIRES the following Arduino libraries:
// - DHT Sensor Library: https://github.com/adafruit/DHT-sensor-library
// - Adafruit Unified Sensor Lib: https://github.com/adafruit/Adafruit_Sensor

#include "DHT.h"

#define DHTPIN 2     // Digital pin connected to the DHT sensor

// Uncomment whatever type you're using!
#define DHTTYPE DHT11   // DHT 11
//#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
//#define DHTTYPE DHT21   // DHT 21 (AM2301)

// Connect pin 1 (on the left) of the sensor to +5V
// NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1
// to 3.3V instead of 5V!
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 4 (on the right) of the sensor to GROUND
// Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the sensor

// Initialize DHT sensor.
// Note that older versions of this library took an optional third parameter to
// tweak the timings for faster processors.  This parameter is no longer needed
// as the current DHT reading algorithm adjusts itself to work on faster procs.
DHT dht(DHTPIN, DHTTYPE);
```

```
void setup() {
  Serial.begin(9600);
  Serial.println(F("DHTxx test!")); //注意 F（ ），不加效果一样，但实质不一样

  dht.begin();
}

void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  float f = dht.readTemperature(true);

  // Check if any reads failed and exit early (to try again).
  //int isnan (double __x)  如果参数 x 表示“非数字”(NaN- Not-a-Number)对象，则函数 isnan()返回 1，否则返回 0。
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }

  // Compute heat index in Fahrenheit (the default)
```

```
  //热指数是指高温时，当相对湿度增加后，人体真正感受到的温度会超过实际温度，也就是体感温度
  float hif = dht.computeHeatIndex(f, h);
  // Compute heat index in Celsius (isFahreheit = false)
  float hic = dht.computeHeatIndex(t, h, false);

  Serial.print(F("Humidity: "));
  Serial.print(h);
  Serial.print(F("%  Temperature: "));
  Serial.print(t);
  Serial.print(F("°C "));
  Serial.print(f);
  Serial.print(F("°F  Heat index: "));
  Serial.print(hic);
  Serial.print(F("°C "));
  Serial.print(hif);
  Serial.println(F("°F"));
}
```

说明：

Serial.println(F("DHTxx test!"));

//注意 F（ ），不加效果一样，但实质不一样

添加 F() 相当于为字符串常量定义了 PROGMEM 属性，常量字符串仍然存储在 FLASH 中，但是程序运行时不会再将常量字符串从 FLASH 中 copy 到 SRAM 中，而是直接读取 FLASH 中的字符串，这样一来就节约了 SRAM，但是代码运行速度就下降了。

参考资料：

https://www.cnblogs.com/lulipro/p/10815338.html