第 2-9 讲：温度传感器 DS18B20

1. 学习目的

1. 了解 DS18B20 数字温度传感器的基本原理及其数据格式。
2. 掌握 IAP15F2K61S2/IAP15W4K61S4 与 DS18B20 单总线通信的程序设计，通信步骤，数据校验等。

2. 硬件电路设计

2.1. DS18B20 简介

2.1.1. DS18B20 主要特性

DS18B20 是 Dallas 半导体公司推出的新一代单总线数字温度传感器，其控制命令和数据都是以数字信号的方式输入输出，相比较于模拟温度传感器，具有功能强大、硬件简单、易扩展、抗干扰性强等特点。DS18B20 主要特性如下：

- 测温范围：-55℃ 到 +125℃。
- 通信接口：1-Wire（单总线）。
- 内置温度报警功能。
- 可寄生供电。
- 具有唯一的序列号，在一根通信线可挂接多个，形成总线结构。

2.1.2. DS18B20 封装

DS18B20 有多种封装，如下图所示，其中我们最常用的封装是 TO-92。

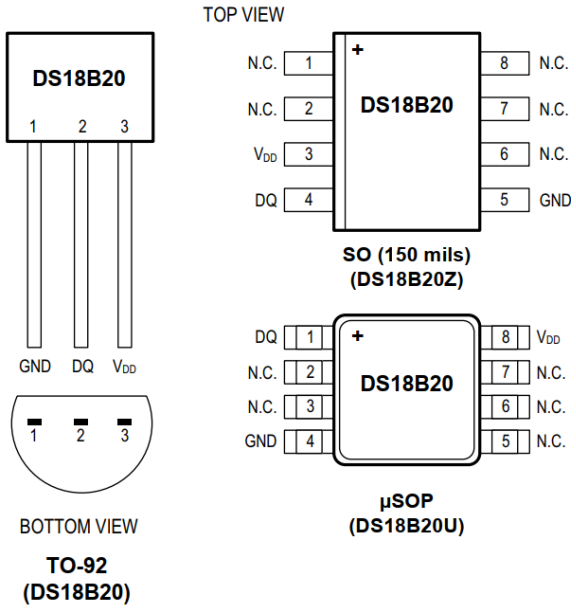


图 1：DS18B20 封装

TO-92 封装的 DS18B20 实物图如下

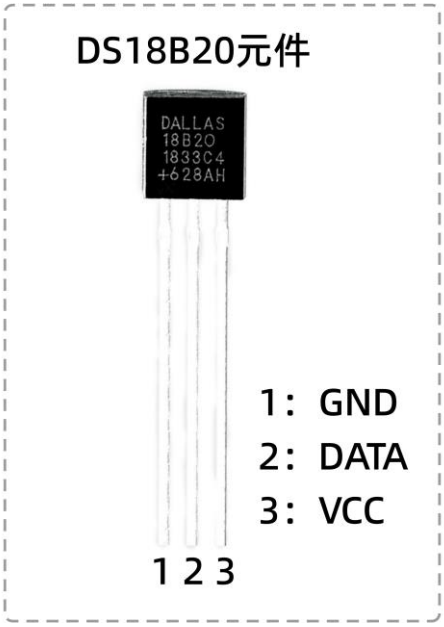


图 2：DS18B20 实物（TO-92 封装）

市面上常见的还有一种将 DS18B20 元件封装起来的防水型 DS18B20，如下图所示，读者在选型时可以根据实际的应用场景选择使用元件型或防水型的。

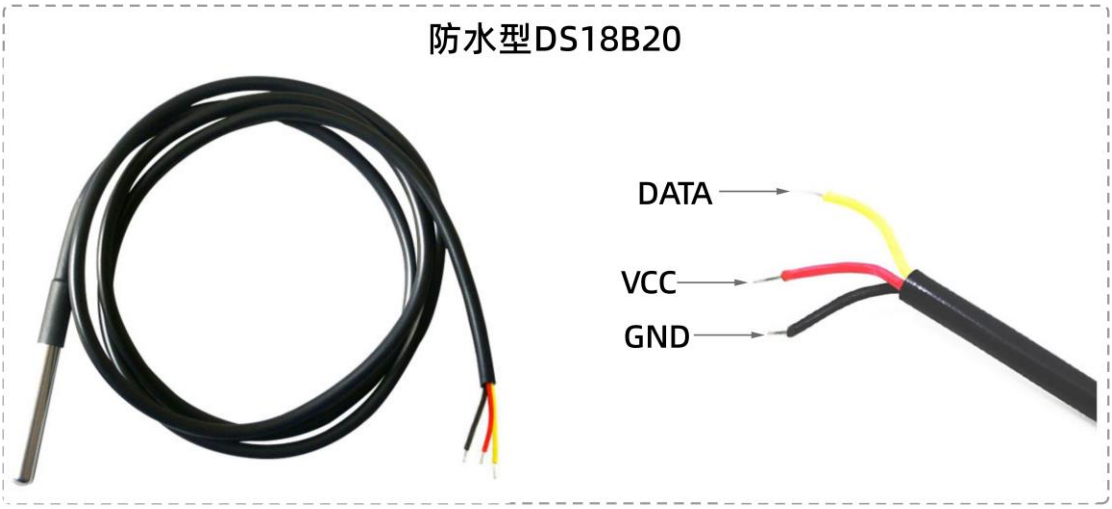
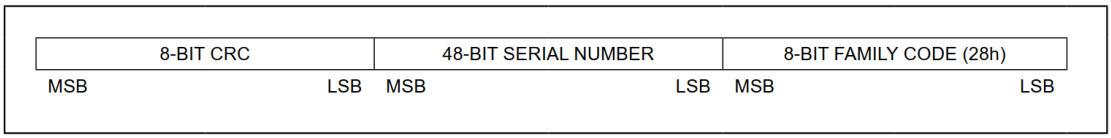


图 3：防水型 DS18B20

2.1.3. DS18B20 寄存器结构

DS18B20 片内包含一个用于存储唯一 ID 的 64 位 ROM 和一个用于存储温度等数据的 RAM 数据暂存器。

1. **ROM 只读存储器**：64 位，用于存放 DS18B20 的 ID 编码，其低 8 位是单线系列编码（DS18B20 的是 28H），中间的 48 位是芯片唯一的序列号，最后 8 位是以上 56 位的循环冗余校验（CRC)值。ROM 在芯片出厂时设置，用户无法更改。



2. RAM 数据暂存器

DS18B20 共 9 个字节 RAM，每个字节为 8 位，如下图所示，我们读取温度数据时就是从这个 RAM 读取的。

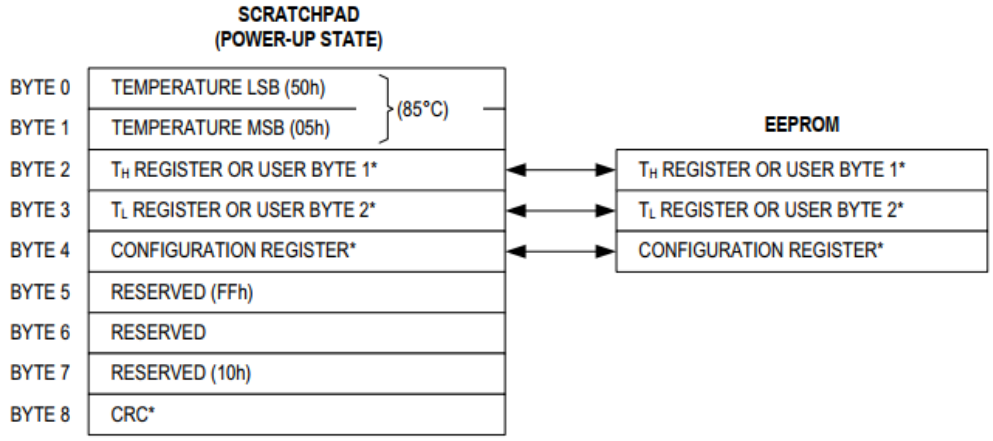


图 4：RAM 数据暂存器

- 字节 0、字节 1：温度转换后的数据值。
- 字节 2、字节 3：用户 EEPROM（温度报警值 TH、TL）的镜像，在上电复位时其值将被刷新。
- 字节 4：配置寄存器，用于配置温度分辨率，可配置为 9、10、11、12 位分辨率（DS18B20 默认的是 12 位的分辨率）。
- 字节 5、字节 6、字节 7：保留。
- 字节 8：前 8 个字节的 CRC 码。

1) 温度寄存器的格式

DS18B20 测得的温度值以二进制补码的形式存放于温度寄存器中。符号位（S）表示温度是正还是负：对于正数 S=0，对于负数 S=1。如果 DS18B20 配置为 12 位分辨率，则温度寄存器中的所有位都将包含；如果配置为 11 位分辨率，位 0 是未定义的；如果配置为 10 位分辨率，比特 1 和 0 是未定义的，而对于 9 比特分辨率，则比特 2、1 和 0 为未定义的。

	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

S = SIGN

图 5：温度寄存器的格式

下表给出了 12 位分辨率的数字输出数据和相应的温度读数的示例。主机读取数据后，先将数据补码变为原码，处理正负后再乘以 0.0625 即可得到温度值。

表 1：12 位分辨率温度示例

温度（℃）	二进制	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

2) 配置寄存器(Configuration Register)：

配置寄存器用于确定温度值的数字转换分辨率，DS18B20 工作时按此寄存器中的分辨率将温度转换为相应精度的数值。DS18B20 默认的是 12 位的分辨率，通常，我们无需配置该寄存器。

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

配置寄存器中的 R1、R0 位的组合决定了温度分辨率，如下表所示。

表 2：温度分辨率配置

R1	R0	分辨率（位）	最大转换时间	
0	0	9	93.75ms	(t_{CONV}/8)
0	1	10	187.5ms	(t_{CONV}/4)
1	0	11	375ms	(t_{CONV}/2)
1	1	12	750ms	(t_{CONV})

2.1.4. DS18B20 的时序

DS18B20 的时序包括发起时序和读写时序，发起时序完成初始化，读写时序完成数据的读和写。

1. 发起时序

发起时序包含发送复位脉冲和存在检测。单片机给 DS18B20 单总线至少 480uS 的低电平信号，当 DS18B20 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（60~240uS 的低电平信号），这个过程称为发起时序，如下图所示。发起时序完成后，单片机即可进行读写操作。

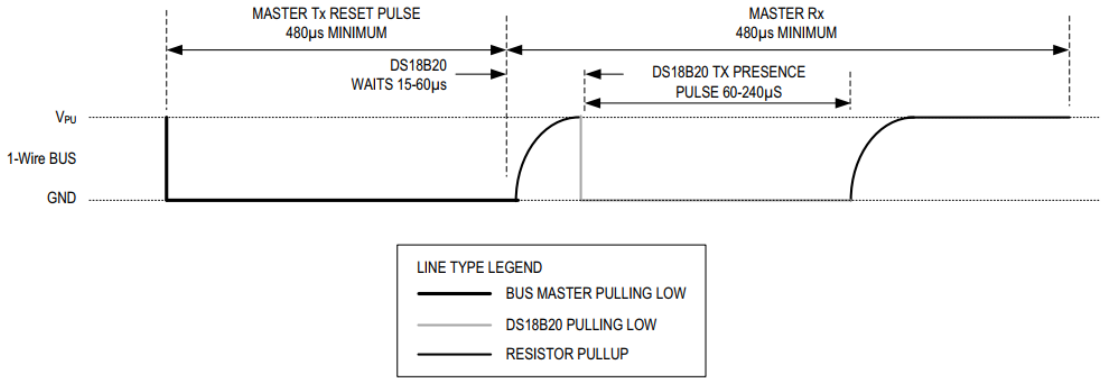


图 6：发起时序（发送复位脉冲和存在检测）

2. 读写时序

DS18B20 的读写时序如下图所示。

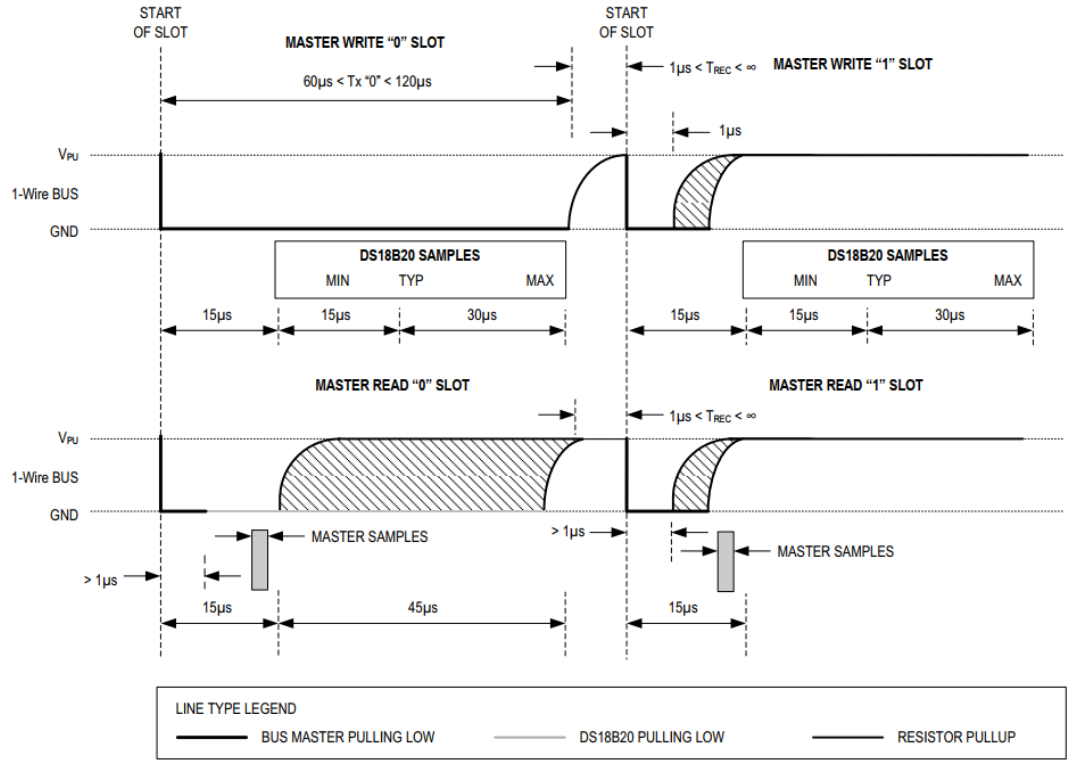


图 7：读写时序

1) 写时序

有两种类型的写入时隙："写入 1"时隙和"写入 0"时隙。总线主机使用写入 1 时隙将逻辑 1 写入 DS18B20，使用写入 0 时隙将逻辑 0 写入 DS18B20。所有写入时隙的持续时间必须至少为 60µs，单个写入时隙之间的恢复时间至少为 1µs。这两种类型的写入时隙都是由主机将 1-Wire 总线拉低启动的。

要生成写入 1 时隙，在将 1-Wire 总线拉低后，总线主机必须在 15µs 内释放 1-Wire 总线。当总线被释放时，上拉电阻将把总线拉高。要生成写入 0 时隙，在将 1-Wire 总线拉低后，总线主机必须在持续时间内继续保持总线低电平时隙（至少 60µs）。

DS18B20 在主机启动写入时隙后持续 15μs 至 60μs 的窗口期间对单线总线进行采样。如果总线在采样窗口期间为高电平，则向 DS18B20 写入 1。如果线路为低电平，则向 DS18B20 写入 0。

2) 读时序

DS18B20 只能在主机发出读取时隙时向主机发送数据。因此，主机必须在发出读取命令后立即生成读取时隙，以便 DS18B20 能够提供所请求的数据。

所有读取时隙的持续时间必须至少为 60μs，时隙之间的恢复时间至少为 1μs。主设备将单线总线拉低至少 1μs，然后释放总线，从而启动读取时隙。

在主机启动读取时隙后，DS18B20 将开始在总线上传输 1 或 0。DS18B20 通过保持总线为高电平来发送 1，并通过拉低总线来发送 0。当发送 0 时，DS18B20 将在时隙结束时释放总线，并且总线将被上拉电阻拉回到其高电平空闲状态。DS18B20 的输出数据在启动读取时隙的下降沿后的 15μs 内有效，因此，主机必须释放总线，然后在从时隙开始的 15μs 内对总线状态进行采样。

2.1.5. DS18B20 指令

DS18B20 指令如下表所示，表中粗斜体字体的指令是我们常用的指令。

表 3：DS18B20 指令

指令类型	指令	功能	描述
ROM 指令	F0H	搜索 ROM	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。如果只挂接一个芯片，使用读 ROM 指令即可。
	33H	***读 ROM***	***当总线上只有一个 DS18B20 时才可以用此指令，允许主设备读取从设备的序列号。***
	55H	匹配 ROM	指令后面紧跟着由主设备发出了 64 位序列号，当总线上有多只 DS18B20 时，只有与主设备发出的序列号相同的芯片才可以做出反应，其他芯片将等待下一次复位。该指令适应单芯片和多芯片挂接。
	CCH	***跳过 ROM***	***允许主设备不提供 64 位 ROM 编码就可以使用功能指令。单芯片时使用，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。***
	ECH	报警搜索	在多芯片挂接情况，报警芯片搜索指令只对符合温度高于 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达不到报警条件为止
功能指令	***44H***	***温度转换***	***用以启动一次温度转换。温度转换指令被执行，产生的温度转换结果数据以 2 个字节的形式被存储在高速暂存器中，而后 DS18B20 保持等待状态。***
	4EH	写暂存器	向 DS18B20 的暂存器写入数据，开始位置在 TH 寄

			存器（暂存器的第 2 个字节），接下来写入 TL 寄存器（暂存器的第 3 个字节），最后写入配置寄存器（暂存器的第 4 个字节）。
	BEH	***读暂存器***	***读取 DS18B20 暂存器的内容（共 9 个字节）。***
	48H	拷贝暂存器	将 TH、TL 和配置寄存器的数据拷贝到 EEPROM 中保存。
	B8H	复制 EEROM	将 TH、TL 和配置寄存器的数据从 EEPROM 拷贝到暂存器。
	B4H	读电源模式	主设备发出此指令，然后发出一个读取时隙，以确定总线上是否有任何 DS18B20 正在使用寄生电源。在读取时隙期间，寄生供电的 DS18B20 将拉低总线，外部供电的 DS18 B20 将使总线保持高电平。

2.2. DS18B20 电路

PK107D 开发板上设计了 DS18B20 温度传感器电路，如下图所示。

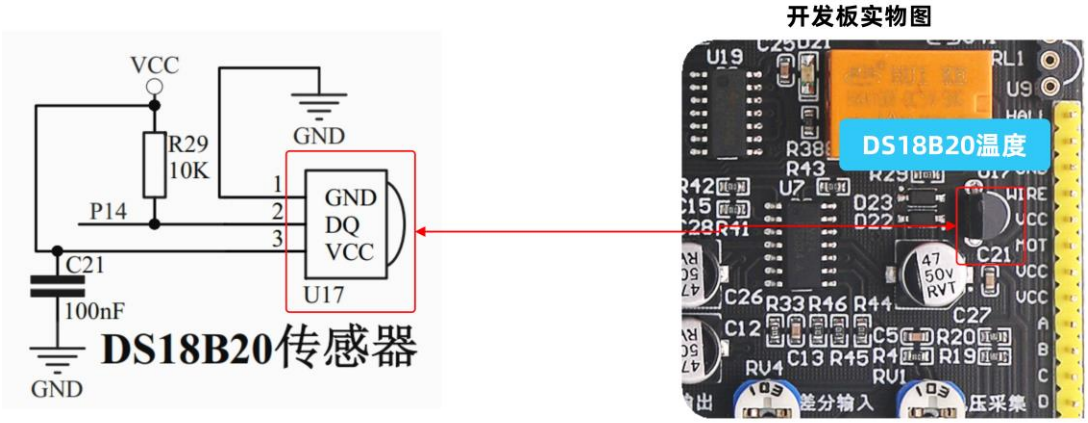


图 8：DS18B20 电路

DS18B20 占用的 IAP15F2K61S2/IAP15W4K61S4 的引脚如下表：

表 4：DS18B20 引脚分配

DS18B20	引脚	说明
DS18B20 数据	P1.4	独立 IO 口

3. 软件设计

本节描述的应用步骤是针对总线挂载单个 DS18B20 温度传感器的情况，对于总线挂载多个 DS18B20，应用的场景较少，这里不予描述。

3.1. DS18B20 应用步骤

使用 DS18B20 时，最常用的操作是读取 ROM 获取串号和读取温度，DS18B20 读 ROM 流程的流程如下图所示。首先由主机发起复位并等待从机响应，由此完成 DS18B20 存在检测，接着主机发送读 ROM 指令读取 64 位（8 个字节）数据，之后进行 CRC8 校验，由此判断读取的数据是否正确。

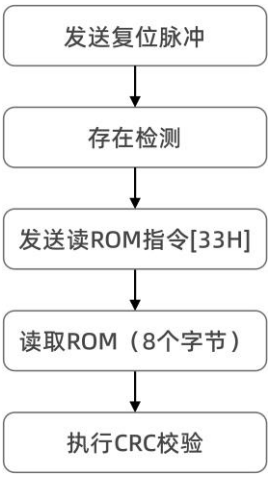


图 9：读 ROM 流程

DS18B20 读取温度的流程如下所示。读取温度数据需要执行两次工作周期，第 1 个周期发送温度转换指令完成温度转换，第 2 个周期发送读取暂存器指令读出暂存器 9 个字节数据，之后进行校验，校验成功后计算出温度。

读取温度时，如果我们不需要校验，也可以只读取暂存器中前 2 个字节的温度数据，而不必将所有数据都读出来。

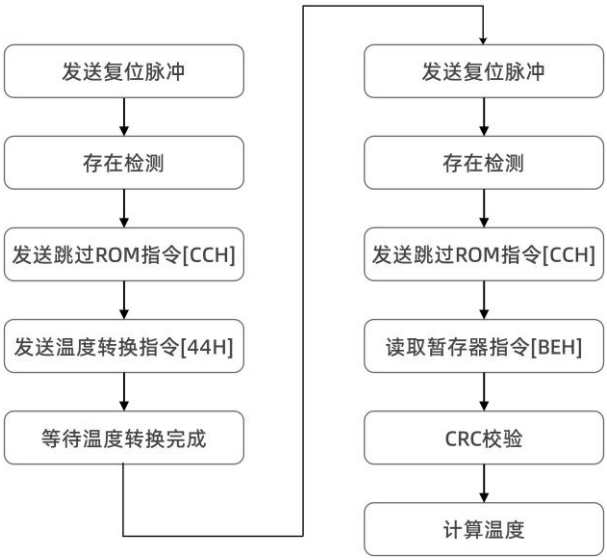


图 10：读温度流程

✧ **注意**：在编写 DS18B20 测温程序时，向 DS18B20 发指令后，需要等待 DS1820 响应，这时候切记不要用死循环去等待响应，因为一旦 DS18B20 接触不好或断线，将没有返

回信号，这会导致程序进入死循环。正确的操作是严格按照时序操作，下面是发起操作中单片机发送复位脉冲后等待 DS18B20 响应的代码，读者可以看一下他们的区别。

- 典型的死循环等待“while(DATA_PIN);”，如果 DS18B20 无响应（DATA_PIN 电平一直为高电平），程序会一直在这循环查询引脚状态。
- 按照时序延时后查询引脚状态

```
delay_us(70);          //按照时序等待 70us
if(DATA_PIN)           // DATA_PIN 为高电平，表示 DS18B20 未响应
{
  return false;        //DS18B20 无响应，程序返回
}
```

3.2. DS18B20 温度读取（数码管显示）实验

✧ 注：本节的实验是在“实验 2-7-1：数码管显示”的基础上修改，本节对应的实验源码是：“实验 2-9-1：DS18B20 温度传感器 - 数码管显示”。

3.2.1. 实验内容

DS18B20 使用默认的 12 位温度分辨率，上电启动后，在主循环中每秒读取一次温度，读取温度时，读取全部暂存器的内容并校验（9 个字节），读取的温度数据计算为温度后在数码管上显示（显示温度小数点后 1 位）。

3.2.2. 代码编写

1. 新建一个名称为“ds18b20.c”的文件及其头文件“ds18b20.h”并保存到工程的“Source”文件夹，并将“ds18b20.c”加入到 Keil 工程中的“SOURCE”组。
2. 引用头文件

因为在“main.c”文件中使用了“ds18b20.c”文件中的函数，所以需要引用下面的头文件“ds18b02.h”。

代码清单：引用头文件

```
1.  //引用 DS18B20 的头文件
2.  #include   "ds18b20.h"
```

3. DS18B20 复位和存在检测

按照前文描述的发起时序编写代码如下。主机拉低信号引脚电平并延时 480us 发送复位脉冲，接着拉高信号引脚电平，延时 70us 后读取信号引脚电平状态，若为低电平，则 DS18B20 存在，若仍然为高电平，则说明无 DS18B20 响应，返回存在检测失败。

存在的时间宽度至少为 480us，因此，读取后 DS18B20 返回的低电平后，还需要延时 410us（前面已经延时了 70us）。

代码清单：DS18B20 复位和存在检测

```
1.  /*******************************************************************************
2.   * 描  述 : DS18B20 初始化：主机发送复位脉冲和设备存在检测
3.   * 参  数 : 无
```

```
4.    * 返回值 : true:DS18B20 正确响应；false:DS18B20 未响应
5.    ********************************************************************************/
6.   bool DS18B20_Initialize(void)
7.   {
8.      DATA_PIN = 0;              //发送低电平复位信号
9.      delay_us(480);             //延时至少 480us
10.     DATA_PIN = 1;              //释放数据线
11.     delay_us(70);              //等待 70us
12.     if(DATA_PIN)               //检测存在脉冲
13.     {
14.        return false;
15.     }
16.     delay_us(410);             //等待设备释放数据线
17.     return true;
18.  }
```

4. 编写读、写一个字节的函数

按照前文描述的写时序编写代码如下。注意发送 1 和 0 的时候时序是不一样的，发送 1 时主机拉低信号引脚电平并延时 1us，接着拉高信号引脚电平并延时 60us；发送 0 时主机拉低信号引脚电平并延时 60us，接着拉高信号引脚电平并延时 10us。

代码清单：写 1 字节函数

```
1.   /*********************************************************************************
2.    * 描   述 : 向 DS18B20 写 1 字节数据
3.    * 参   数 : dat[in]：写入的数据
4.    * 返回值 : 无
5.    *********************************************************************************/
6.   void DS18B20_WriteByte(u8 dat)
7.   {
8.      u8 i;
9.
10.     for (i=0; i<8; i++)
11.     {
12.        if (dat & 0x01)         //写 1
13.        {
14.           DATA_PIN = 0;        //开始时间片
15.           delay_us(1);         //延时等待
16.           DATA_PIN = 1;
17.           delay_us(60);        //等待时间片结束
18.        }
19.        else                    //写 0
20.        {
21.           DATA_PIN = 0;        //开始时间片
22.           delay_us(60);        //延时等待
```

```
23.          DATA_PIN = 1;
24.          delay_us(10);       //等待时间片结束
25.       }
26.       dat >>= 1;             //准备好待发送的下一个位
27.    }
28. }
```

读取数据时，主机拉低信号引脚电平并延时 1us，接着拉高引脚电平延时 1us 后对信号引脚采样并将结果保存到变量“dat”。DS18B20 要求所有读取时隙的持续时间必须至少为 60us，因此，在读取一个位后，延时 60us，代码清单如下。

代码清单：读 1 字节函数

```
1.  /*************************************************************************************
2.   * 描   述 : 从 DS18B20 读 1 字节数据
3.   * 参   数 : 无
4.   * 返回值 : 读取的 1 字节数据
5.   *************************************************************************************/
6.  u8 DS18B20_ReadByte(void)
7.  {
8.     u8 i;
9.     u8 dat = 0;
10.    for (i=0; i<8; i++)
11.    {
12.       dat >>= 1;
13.       DATA_PIN = 0;               //开始时间片
14.       delay_us(1);                //延时等待
15.       DATA_PIN = 1;               //准备接收
16.       delay_us(1);                //接收延时
17.       if (DATA_PIN) dat |= 0x80;  //读取数据
18.       delay_us(60);               //等待时间片结束
19.    }
20.    return dat;
21. }
```

5. 编写读 ROM 函数

按照 DS18B20 应用步骤中读 ROM 的步骤，编写的读 ROM 代码如下所示。

代码清单：读 ROM（8 个字节）

```
1.  /*************************************************************************************
2.   * 描   述 : 读取 ROM
3.   * 参   数 : p_dat[in]：保存读取的数据
4.   * 返回值 : DS18B20_SUCCESS：读取成功；DS18B20_DATA_ERR：数据校验错误
5.   *************************************************************************************/
6.  u8 DS18B20_ReadID(u8 *p_dat)
7.  {
```

```
    u8 i;
    if(DS18B20_Initialize() == false)
    {
        return DS18B20_NACK;
    }
    DS18B20_WriteByte(0x33); //读 ROM 指令
    for(i=0;i<8;i++)          //因为是 8 个字节，所以要读取 8 次
    {
        p_dat[i]=DS18B20_ReadByte();
    }
    //CRC8 校验
    if (CRC8(p_dat, 7) == p_dat[7])
    {
        return DS18B20_SUCCESS;
    }
    else
    {
        return DS18B20_DATA_ERR;
    }
}
```

6. 读取温度数据

我们编写了两个温度读取函数“DS18B20_ReadTemperature()”和“DS18B20_ReadAll ()”，他们的区别如下：

- DS18B20_ReadTemperature()：仅读取暂存器中的前 2 个字节温度数据，无校验。
- DS18B20_ReadAll ()：读取暂存器全部数据，共 9 个字节，读取后，进行 CRC8 校验。

两个温度读取函数的代码清单如下。

代码清单：读温度数据（仅读取暂存器中的前 2 个字节温度数据，无校验）

```
/*********************************************************************************************
 * 描   述 : 温度读取函数,只读取温度，即 9 个字节中的前 2 个字节
 * 参   数 : 无
 * 返回值 : 温度值
 *********************************************************************************************/
u8 DS18B20_ReadTemperature(u8 *p_dat)
{
    if(DS18B20_Initialize() == false)
    {
        return DS18B20_NACK;
    }
    DS18B20_WriteByte(0xCC);        //跳过 ROM 命令
    DS18B20_WriteByte(0x44);        //开始转换命令
    delay_ms(755);                  //等待转换完成
    if(DS18B20_Initialize() == false)
```

```
    {
        return DS18B20_NACK;
    }
    DS18B20_WriteByte(0xCC);        //跳过 ROM 命令
    DS18B20_WriteByte(0xBE);        //读取温度寄存器等（共可读 9 个寄存器） 前两个就是温度
    p_dat[0] = DS18B20_ReadByte(); //读温度低字节
    p_dat[1] = DS18B20_ReadByte(); //读温度高字节
    return DS18B20_SUCCESS;
}
```

代码清单：读取暂存器全部内容（共 9 个字节，第 9 个字节为 CRC 校验）

```
/**************************************************************************************
 * 描  述 : 读取暂存器的全部内容（9 个字节）并校验
 * 参  数 : 无
 * 返回值 : DS18B20_SUCCESS：读取成功；DS18B20_DATA_ERR：数据校验错误
 **************************************************************************************/
u8 DS18B20_ReadAll(u8 *p_dat)
{
    u8 i;
    if(DS18B20_Initialize() == false)
    {
        return DS18B20_NACK;
    }
    DS18B20_WriteByte(0xCC);        //跳过 ROM 命令
    DS18B20_WriteByte(0x44);        //开始转换命令
    delay_ms(755);                  //等待转换完成

    if(DS18B20_Initialize() == false)
    {
        return DS18B20_NACK;
    }
    DS18B20_WriteByte(0xCC);        //跳过 ROM 命令
    DS18B20_WriteByte(0xBE);        //读取温度寄存器等（共可读 9 个寄存器） 前两个就是温度
    for(i=0;i<9;i++)                //因为是 9 个字节，所以要读取 9 次
    {
        p_dat[i]=DS18B20_ReadByte();
    }
    if (CRC8(p_dat, 8) == p_dat[8]) //CRC8 校验
    {
        return DS18B20_SUCCESS;
    }
    else
    {
```

```
33.         return DS18B20_DATA_ERR;
34.     }
35. }
```

7. 温度计算

主机读取数据后，先将数据补码变为原码，根据数据中的“S 位（高 5 位，负温是全部为 1，正温时全部为 0）”判断是正温还是负温，之后乘以 0.0625 即可得到测量的温度值。

代码清单：温度计算函数

```
1.  /**************************************************************************************
2.   * 描  述 : 读取的温度数据转换为温度
3.   * 参  数 : 无
4.   * 返回值 : 温度值
5.   **************************************************************************************/
6.  float DS18B20_TempConvert(u8 LSB, u8 MSB)
7.  {
8.      float temperature;
9.      short dat;
10.
11.     dat = LSB | (MSB << 8);
12.     if (dat & 0x8000) //最高位为1，是负温
13.     {
14.         dat = ~dat + 1;
15.         temperature = 0.0 - (dat * 0.0625); //DS18B20 使用的是默认 12 位分辨率，所以乘以 0.0625
16.         return temperature;
17.     }
18.     temperature = dat * 0.0625;
19.     return temperature;
20. }
```

8. 主函数

主函数中，执行相关的初始化操作后，程序进入主循环后以 1 秒的间隔读取温度数据，计算为温度后在数码管显示，代码清单如下。

代码清单：主函数

```
1. /*************************************************************************
2. 功能描述：主函数
3. 入口参数：无
4. 返 回 值：int 类型
5. *************************************************************************/
6. int main(void)
7. {
8.     u8 i;
9.     u8 temp[6];
10.    float fTemp;
11.    u8 uTemp;
```

```
12.     u8 dat_buf[9];
13.
14.     P2M1 &= 0x1F;   P2M0 |= 0xE0;     //设置 P2.5、P2.6、P2.7 为推挽输出
15.     P0M1 &= 0x00;   P0M0 |= 0xFF;     //设置 P0.0 ~ P0.7 为推挽输出
16.     P1M1 &= 0xEF;   P1M0 &= 0xEF;     //设置 P1.4 为准双向口
17.
18.     SEG_off();                  //控制 8 位数码管/点阵不显示
19.     leds_off();                 //熄灭 D1~D8 指示灯
20.     ULN2003_off();              //控制 ULN2003 输出高电平，关闭蜂鸣器、继电器等
21.     delay_ms(10);               //延时
22.
23.     timer2_init();              //定时器 2 初始化
24.     timer2_start();             //启动定时器 2
25.     EA = 1;                     //使能总中断
26.     delay_ms(200);
27.
28.     while(1)
29.     {
30.         //将 temp 数组初始化（清零）
31.         for(i=0;i<6;i++)
32.         {
33.             temp[i] = 0;
34.         }
35.
36.         //读取暂存器全部数据，若校验正确则计算温度
37.         if(DS18B20_ReadAll(dat_buf) == DS18B20_SUCCESS)             //数据正确
38.         {
39.             fTemp = DS18B20_TempConvert(dat_buf[0],dat_buf[1]);      //计算温度
40.         }
41.         sprintf(temp, "%.01f", fTemp);                  //浮点数转成字符串
42.         uTemp = (u8)fTemp;
43.
44.         //串口打印温度值（温度检测为非负温时）
45.         if((ReadTempFlag==0)&&(uTemp<10))
46.         {
47.              LEDseg_DispUpdata(LEDSEG_5,17,LEDSEG_DP_OFF);//更新第 5 个数码管显示内容
48.              LEDseg_DispUpdata(LEDSEG_6,17,LEDSEG_DP_OFF);//更新第 6 个数码管显示内容
49.              LEDseg_DispUpdata(LEDSEG_7,temp[0]-0x30,LEDSEG_DP_ON);//更新第 7 个数码管显示内容
50.              LEDseg_DispUpdata(LEDSEG_8,temp[2]-0x30,LEDSEG_DP_OFF);//更新第 8 个数码管显示内容
51.         }
52.         else if((ReadTempFlag==0)&&(uTemp>9))
53.         {
54.              LEDseg_DispUpdata(LEDSEG_5,17,LEDSEG_DP_OFF);//更新第 5 个数码管显示内容
```

```
55.             LEDseg_DispUpdata(LEDSEG_6,temp[0]-0x30,LEDSEG_DP_OFF);//更新第 6 个数码管显示内容
56.             LEDseg_DispUpdata(LEDSEG_7,temp[1]-0x30,LEDSEG_DP_ON);//更新第 7 个数码管显示内容
57.             LEDseg_DispUpdata(LEDSEG_8,temp[3]-0x30,LEDSEG_DP_OFF);//更新第 8 个数码管显示内容
58.         }
59.         //串口打印温度值（温度检测为负温时）
60.         if((ReadTempFlag==1)&&(uTemp<10))
61.         {
62.             LEDseg_DispUpdata(LEDSEG_5,17,LEDSEG_DP_OFF);//更新第 5 个数码管显示内容
63.             LEDseg_DispUpdata(LEDSEG_6,16,LEDSEG_DP_OFF);//更新第 6 个数码管显示内容
64.             LEDseg_DispUpdata(LEDSEG_7,temp[0]-0x30,LEDSEG_DP_ON);//更新第 7 个数码管显示内容
65.             LEDseg_DispUpdata(LEDSEG_8,temp[2]-0x30,LEDSEG_DP_OFF);//更新第 8 个数码管显示内容
66.         }
67.         else if((ReadTempFlag==1)&&(uTemp>9))
68.         {
69.             LEDseg_DispUpdata(LEDSEG_5,16,LEDSEG_DP_OFF);//更新第 5 个数码管显示内容
70.             LEDseg_DispUpdata(LEDSEG_6,temp[0]-0x30,LEDSEG_DP_OFF);//更新第 6 个数码管显示内容
71.             LEDseg_DispUpdata(LEDSEG_7,temp[1]-0x30,LEDSEG_DP_ON);//更新第 7 个数码管显示内容
72.             LEDseg_DispUpdata(LEDSEG_8,temp[3]-0x30,LEDSEG_DP_OFF);//更新第 8 个数码管显示内容
73.         }
74.         delay_ms(500);    //延时 500ms 读取一次温度值，有利于观察现象
75.     }
76.}
```

3.2.3. 硬件连接

本实验程序的编写都是基于 IO 模式，所以 J24 端子需要使用短路帽将该端子第 1 引脚和第 2 引脚短接，即选择为 IO 模式。DS18B20 温度传感器是直接焊接到开发板上的，如下图所示。

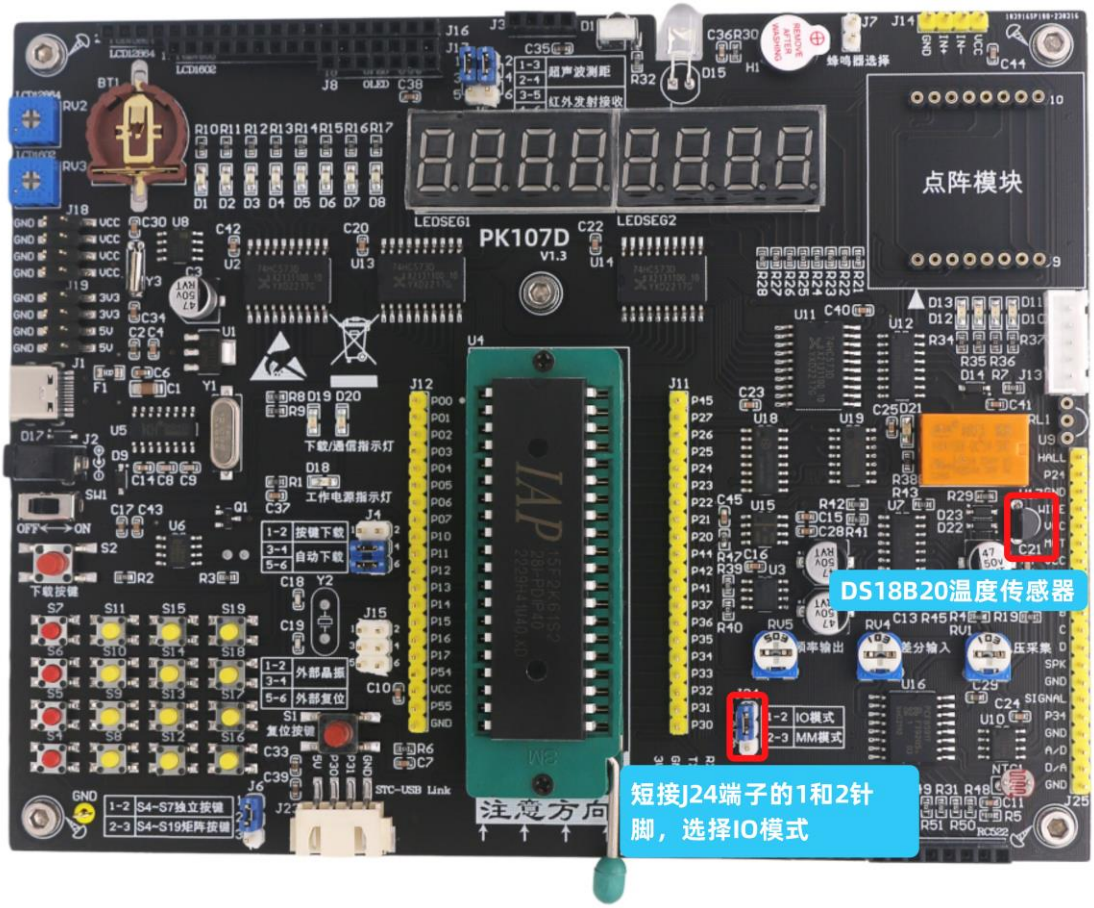


图 11：硬件连接

3.2.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-9-1：DS18B20 温度传感器 - 数码管显示”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”（**这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题**）。
2) 双击“…\ds18b20\project”目录下的工程文件“ds18b20.uvproj”。
3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“ds18b20.hex”位于工程的“…\ds18b20\Project\Object”目录下。
4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：12MHz。
5) 程序运行后，可以在开发板数码管上实时显示温度信息（小数点后 1 位）。

3.3. DS18B20 温度读取（串口显示）实验

✧ 注：本节的实验是在“实验 2-9-1：DS18B20 温度传感器 - 数码管显示”的基础上修改，本节对应的实验源码是：“实验 2-9-2：DS18B20 温度传感器 - 串口显示”。

3.3.1. 实验内容

DS18B20 使用默认的 12 位温度分辨率，上电启动后，首先读取 DS18B20 的 64 位

ROM 并进行校验，校验成功则通过串口输出读取的数据。之后在主循环中每秒读取一次温度，读取温度时，分别使用只读取暂存器中的温度数据（2 个字节）和读取全部暂存器的内容并校验（9 个字节）这两种方式，读取的温度数据计算为温度后通过串口输出。

3.3.2. 代码编写

关于 DS18B20 传感器的相关函数在“实验 2-9-1：DS18B20 温度传感器 - 数码管显示”中已有详细介绍，这里不再赘述。

主函数中，执行相关的初始化操作后，读取 DS18B20 的 64 位 ROM，校验正确则通过串口输出。程序进入主循环后以 1 秒的间隔分别使用两种温度读取函数读取温度数据，计算为温度后通过串口输出，代码清单如下。

代码清单：主函数

```
1. /*********************************************************************
2. 功能描述：主函数
3. 入口参数：无
4. 返 回 值：int 类型
5. *********************************************************************/
6. int main(void)
7. {
8.     u8 i;
9.     float temp_value;
10.    u8 id_buf[8];
11.    u8 dat_buf[9];
12.
13.    P3M1 &= 0xFE;   P3M0 &= 0xFE;     //设置 P3.0 为准双向口（串口 1 的 RxD）
14.    P3M1 &= 0xFD;   P3M0 |= 0x02;     //设置 P3.1 为推挽输出（串口 1 的 TxD）
15.    P2M1 &= 0x1F;   P2M0 |= 0xE0;     //设置 P2.5、P2.6、P2.7 为推挽输出
16.    P0M1 &= 0x00;   P0M0 |= 0xFF;     //设置 P0.0 ~ P0.7 为推挽输出
17.    P1M1 &= 0xEF;   P1M0 &= 0xEF;     //设置 P1.4 为准双向口
18.
19.    SEG_off();          //控制 8 位数码管/点阵不显示
20.    ULN2003_off();      //控制 ULN2003 输出高电平，关闭蜂鸣器、继电器等
21.    leds_off();         //熄灭 D1~D8 指示灯
22.    delay_ms(10);       //延时
23.
24.    uart1_init();       //串口 1 初始化
25.    EA = 1;             //使能总中断
26.
27.    //读取 DS18B20 ROM，串口输出读取的 ROM 数据
28.    if(DS18B20_ReadID(id_buf) == DS18B20_SUCCESS)
29.    {
30.        printf("FAMILY   CODE: 0x%.2bX\r\n",id_buf[0]);//打印 FAMILY CODE
31.        printf("SERIAL NUMBER: ");                     //串口打印序列号
```

```
32.
33.         for(i=1;i<7;i++) //因为是 6 个字节，所以要循环 6 次
34.         {
35.             printf("0x%.2bX ",id_buf[i]);//串口打印序列号
36.         }
37.         printf("\r\nCRC: 0x%.2bX\r\n",id_buf[7]);//串口打印 ROM 数据中的 CRC
38.     }
39.     else //读取 ROM 失败，闪烁 LED1 指示灯提示
40.     {
41.         led_toggle(LED_1);
42.         delay_ms(100);
43.     }
44.
45.     while(1)
46.     {
47.         //读取温度:只读取暂存器中 2 个字节的温度数据，没有校验
48.         if(DS18B20_ReadTemperature(dat_buf) == DS18B20_SUCCESS)     //数据正确
49.         {
50.             temp_value = DS18B20_TempConvert(dat_buf[0],dat_buf[1]); //计算温度
51.             printf("温度： %.2f℃\r\n",temp_value);                  //串口打印温度
52.         }
53.         else
54.         {
55.             printf("read temperature err!");
56.         }
57.
58.         delay_ms(1000);                                             //延时，方便观察数据
59.
60.         //读取暂存器全部数据，若校验正确则计算温度
61.         if(DS18B20_ReadAll(dat_buf) == DS18B20_SUCCESS)             //数据正确
62.         {
63.             temp_value = DS18B20_TempConvert(dat_buf[0],dat_buf[1]);//计算温度
64.             printf("温度： %.2f℃\r\n",temp_value);                    //串口打印温度
65.         }
66.         else
67.         {
68.             printf("read temperature err!");
69.         }
70.         delay_ms(1000);                                 //延时，方便观察数据
71.
72.         led_toggle(LED_2);      //指示灯 D1 状态翻转，指示程序的运行
73.     }
74.}
```

3.3.3. 硬件连接

本实验程序的编写都是基于 IO 模式，所以 J24 端子需要使用短路帽将该端子第 1 引脚和第 2 引脚短接，即选择为 IO 模式。DS18B20 温度传感器是直接焊接到开发板上的，连接 USB 线可用于把读取的温度信息串口显示出来，如下图所示。

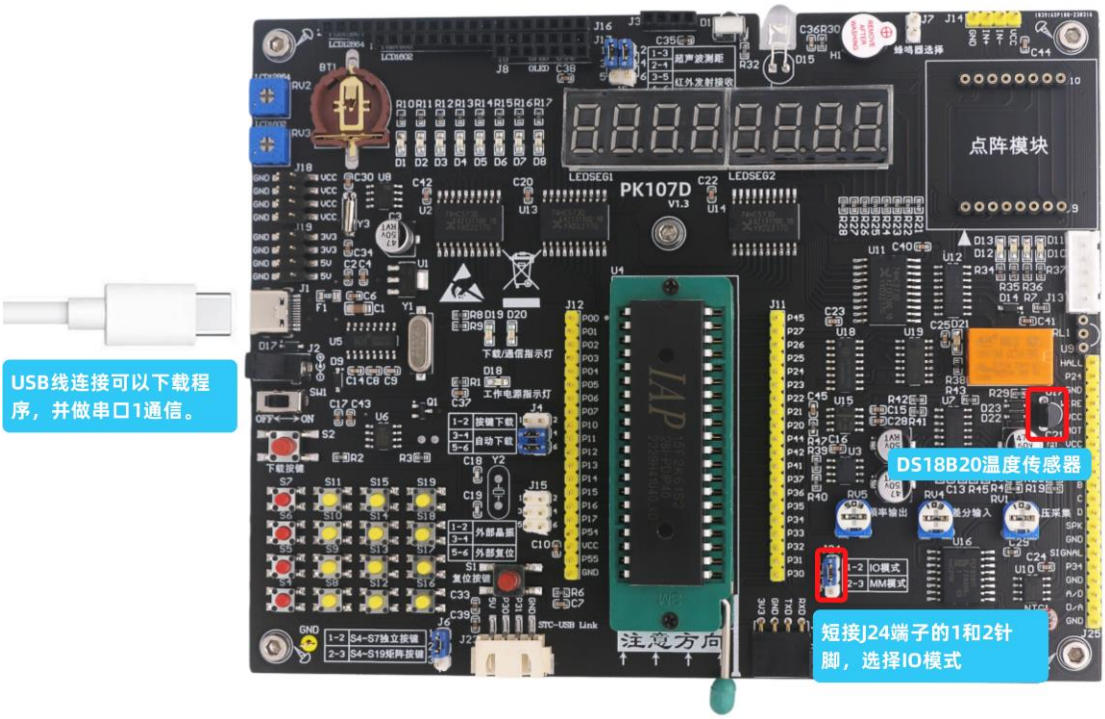


图 12：硬件连接

3.3.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-9-2：DS18B20 温度传感器 - 串口显示”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC15”（**这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题**）。
2) 双击“…\ds18b20\project”目录下的工程文件“ds18b20.uvproj”。
3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“ds18b20.hex”位于工程的“…\ds18b20\Project\Object”目录下。
4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：12MHz。
5) 电脑上打开串口调试助手，选择开发板对应的串口号，将波特率设置为 9600bps。
6) 程序运行后，可以在串口调试助手接收框中观察到开发板发送的 ROM 数据（程序启动时读取）和温度（主循环中读取），如下图所示。

图 12：串口接收的温度信息