STM32 MCU Development

STM32 单片机开发

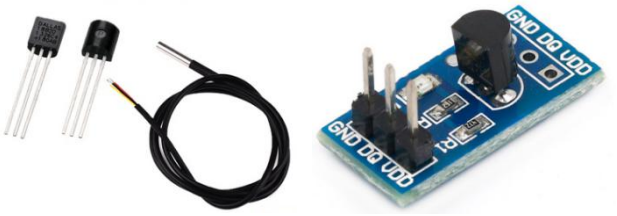
-- 一线协议之 DS18B20 温度采样

目录

1. DS18B20 传感器介绍

1.1. DS18B20 简介

DS18B20 是由 Dallas 半导体公司推出的一种的“一线总线(1-Wire)”接口的温度传感器，它工作在 3~5.5V 的电压范围，其测量温度范围为-55~+125℃ ，精度为±0.5℃。DS18B20 采用多种封装形式，从而使系统设计灵活、方便。与传统的热敏电阻等测温元件相比，它是一种新型的体积小、适用电压宽、接口简单的数字化温度传感器。



每个 DS18B20 芯片在出厂时，都固化烧录了一个唯一的 64 位产品序列号在其 ROM 中，它可以看作是该 DS18B20 的地址序列码。 64 位 ROM 的排列是：前 8 位是产品家族码，接着 48 位是 DS18B20 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)。 ROM 作用是使每一个 DS18B20 都各不相同，这样就可实现一根总线上挂接多个 DS18B20。

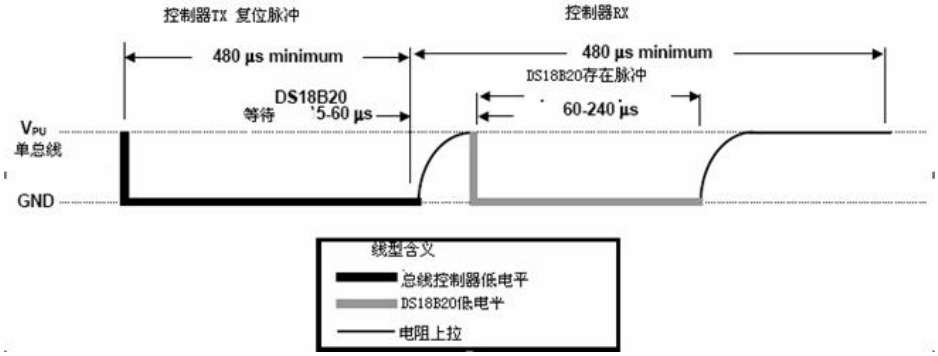
一线总线结构具有简洁且经济的特点，可使用户轻松地组建传感器网络，从而为测量系统的构建引入全新概念。现场温度直接以“一线总线”的数字方式传输，大大提高了系统的抗干扰性。它能直接读出被测温度，并且可根据实际要求通过简单的编程实现 9~12 位的数字值读数方式。

1.2. DS18B20 一线协议

所有的单总线器件要求采用严格的信号时序，以保证数据的完整性。 DS18B20 共有 6 种信号类型：复位脉冲、应答脉冲、写 0/ 1、读 0/1。所有这些信号，除了应答脉冲以外，都由主机发出同步信号，并且发送所有的命令和数据都是字节的低位在前（LSB）。

1）复位脉冲和应答脉冲

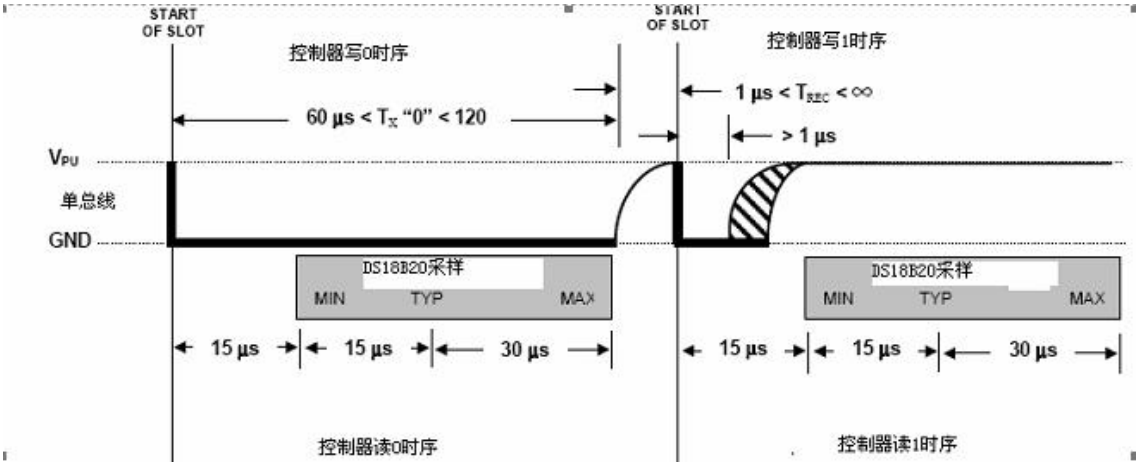
单总线上的所有通信都是以初始化序列开始。主机输出低电平，保持低电平时间至少 480us，以产生复位脉冲。接着主机释放总线， 4.7K 的上拉电阻将单总线拉高，延时 15～60 us，并进入接收模式(Rx)。接着 DS18B20 拉低总线 60~240 us，以产生低电平应答脉冲，若为低电平，再延时 480 us。



2）写时序

写时序包括写 0 时序和写 1 时序。所有写时序至少需要 60us，且在 2 次独立的写时序之间至少需要 1us 的恢复时间，两种写时序均起始于主机拉低总线：

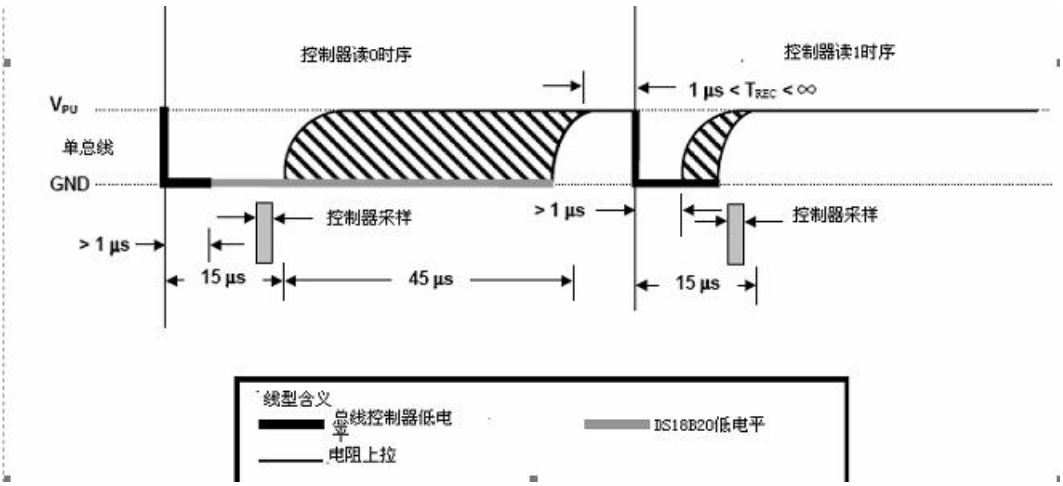
- 写 0 时序：主机输出低电平，延时 60us，然后释放总线，延时 2us；
- 写 1 时序：主机输出低电平，延时 2us，然后释放总线，延时 60us；



3）读时序

单总线器件仅在主机发出读时序时，才向主机传输数据，所以，在主机发出读数据命令后，必须马上产生读时序，以便从机能够传输数据。所有读时序至少需要 60us，且在 2 次独立的读时序之间至少需要 1us 的恢复时间。

当总线控制器把数据线从高电平拉到低电平时，读时序开始，数据线必须至少保持 1us,然后总线被释放。DS18B20 通过拉高或拉低总线上来传输"1"或"0"。当传输逻辑"0"结束后，总线将被释放，通过上拉电阻回到上升沿状态，从 DS18B20 输出的数据在读时序的下降沿出现后 15us 内有效。因此，总线控制器在读时序开始后必须停止把 I/O 口驱动为低电 15us,以读取 I/O 口状态。



1.3. DS18B20 工作流程

DS18B20 工作过程中的协议为：初始化， ROM 操作命令， 存储器操作命令， 处理数据。具体如下：

1. 初始化

单总线上的所有通信都是以初始化序列开始，Master 发出初始化信号后等待从设备的应答信号，已确定从设备是否存在并能正常工作。

2. ROM 操作命令

总线主机检测到 DS18B20 的存在后，便可以发出 ROM 操作命令之一，这些命令如下表所示。一般我们不关心 ROM 中的 16 位产品序列号，通常会发送 0xCC 跳过 ROM 的相关操作。

指令说明	十六进制代码
Read ROM(读 ROM)	[33H]
Match ROM(匹配 ROM)	[55H]
Skip ROM(跳过 ROM]	**[CCH]**
Search ROM(搜索 ROM)	[F0H]
Alarm search(告警搜索)	[ECH]

3. 存储器操作指令

ROM 命令操作完成之后，接下来可以发送相应的高速暂存存储器操作命令，这些命令如下表所示。其中 0x44 命令将通知 DS18B20 温度传感器开始采样， 而 0xBE 命令则将开始读出 DS18B20 的采样值。

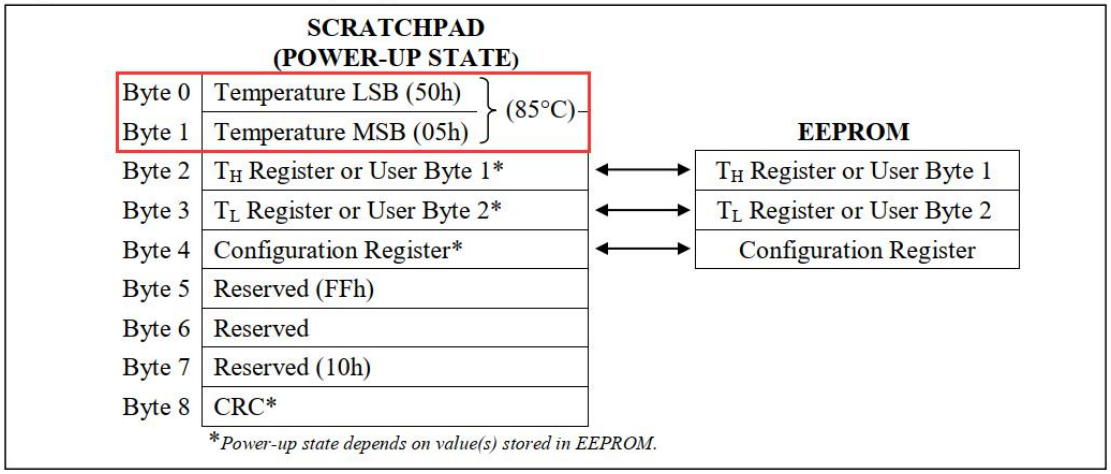
指令说明	十六进制代码
Write Scratchpad(写暂存存储器)	[4EH]
Read Scratchpad(读暂存存储器)	**[BEH]**
Copy Scratchpad(复制暂存存储器)	[48H]
Convert Temperature(温度变换)	**[44H]**
Recall EPROM(重新调出)	[B8H]
Read Power supply(读电源)	[B4H]

4. 数据处理

DS18B20 的高速暂存存储器由 9 个字节组成。当温度转换命令(0x44)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前两个字节。

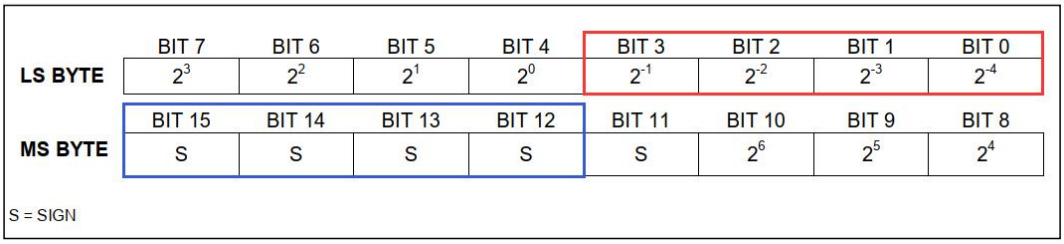
接着单片机可以发送读暂存存储器命令(0xBH)读出存储器里的值， 存储器里的 9 个字节的存储结构如下图所示：

Figure 7. DS18B20 Memory Map



如果我们只关心采样温度值的话，则只需要读前两个字节即可。其中 Byte[0]为温度值的低字节，而 Byte[1]为温度值的高字节。这 16 位数据的格式如下图所示：

Figure 2. Temperature Register Format

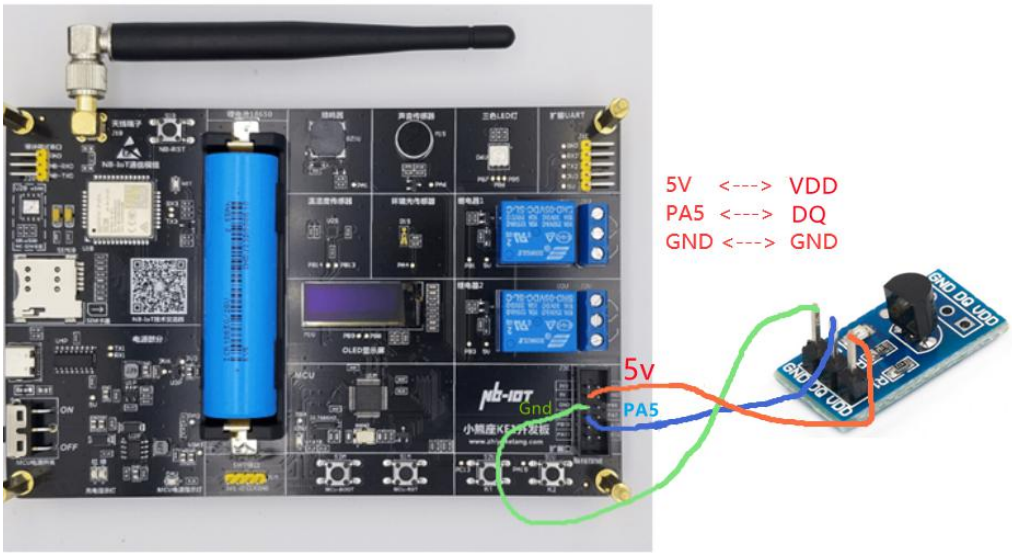


其中 BIT[3:0]为温度值的小数部分，而 BIT[10:4]为温度值的整数部分，BIT[15:11]则为符号位，如果为 0 则温度为正值，如果为 1 则温度为负值。

2. DS18B20 温度采样实现

2.1. 硬件连接

DS18B20 传感器的工作电压范围为 3~5.5V，这里的电源连接 3.3V 和 5V 都可以，此外我们将 DS18B20 的 I/O 口连接到 GPIO 管脚 PA5 上。其实随便接到任意一个扩展出来的 GPIO 口都可以，只需要在下面的 C 代码中作相应的修改即可。



2.2. STM32CubeMX 配置

在接下来的 C 代码中我们将会通过代码来动态配置这个 GPIO 管脚的输入/输出状态，所以这里不需要使用 STM32CubeMX 来配置这个管脚。

2.3. 创建并编写 DS18B20 的驱动源文件 ds18b20.c

```
#include "tim.h"
#include "gpio.h"
#include "main.h"

typedef struct w1_gpio_s
{
        GPIO_TypeDef       *group;
        uint16_t            pin;
} w1_gpio_t;

static w1_gpio_t   W1Dat =  /* IO pin connected to PA5 */
{
         .group = GPIOA,
         .pin   = GPIO_PIN_5,
};

#define W1DQ_Input()         \
{        \
         GPIO_InitTypeDef GPIO_InitStruct = {0}; \
         GPIO_InitStruct.Pin = W1Dat.pin; \
         GPIO_InitStruct.Mode = GPIO_MODE_INPUT; \
         GPIO_InitStruct.Pull = GPIO_PULLUP; \
         GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH; \
         HAL_GPIO_Init(W1Dat.group, &GPIO_InitStruct); \
}

#define W1DQ_Output()        \
{        \
         GPIO_InitTypeDef GPIO_InitStruct = {0}; \
         GPIO_InitStruct.Pin = W1Dat.pin; \
         GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; \
         GPIO_InitStruct.Pull = GPIO_NOPULL; \
         GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH; \
         HAL_GPIO_Init(W1Dat.group, &GPIO_InitStruct); \
}

#define W1DQ_Write(x)   HAL_GPIO_WritePin(W1Dat.group, W1Dat.pin, \
                            (x==1)?GPIO_PIN_SET:GPIO_PIN_RESET)

#define W1DQ_Read()     HAL_GPIO_ReadPin(W1Dat.group,  W1Dat.pin)
```

```
/* Master issues reset pulse and DS18B20s respond with presence pulse */
uint8_t DS18B20_Reset(void)
{
        uint8_t          rv = 0;
        uint8_t          retry;

        /* Setup W1 DQ pin as output and high level*/
        W1DQ_Output();
        W1DQ_Write(1);
        delay_us(2);

        /* Reset pulse by pulling the DQ pin low >=480us */
        W1DQ_Write(0);
        delay_us(480);

        /* Master releases bus to high. When DS18B20 detects this rising edge, it waits 15µs to
60µs */
        W1DQ_Write(1);
        delay_us(60);

        /* Then DS18B20 transmits a presence pulse by pulling the W1 bus low for 60µs to 240µs
*/
        while( W1DQ_Read() && retry <240)
        {
             retry++;
             delay_us(1);
        }

        if(retry >= 240)
         rv = 1;

        delay_us(240);

        /* Master Rx time must >= 480us */
        W1DQ_Output();
        W1DQ_Write(1);
        delay_us(240);

        return rv;
}
```

```
void DS18B20_WriteByte(uint8_t byte)
{
        uint8_t           i = 0;

        W1DQ_Output();

        for(i=0; i<8; i++)
        {
             /* Write 1: pull low <= 15us, Write 0: pull low 15~60us*/
             W1DQ_Write(0);
             delay_us(10);

             /* DS18B20 bit sent by LSB (lower bit first) */
             if( byte & 0x1 )
                  W1DQ_Write(1);
             else
                  W1DQ_Write(0);

             /* Write 1/0 slot both >= 60us, hold the level for 60us */
             delay_us(60);

             /* Release W1 bus to high */
             W1DQ_Write(1);
             delay_us(2);

             /* Prepare for next bit */
             byte >>= 1;
        }
}
```

```
uint8_t DS18B20_ReadByte(void)
{
        uint8_t           i = 0;
        uint8_t           byte = 0;

        for(i=0; i<8; i++)
        {
             /* Read time slot is initiated by master pulling the W1 bus
              * low for minimum of 1µs and then releasing the bus */
             W1DQ_Output();
             W1DQ_Write(0);
             delay_us(2);
             W1DQ_Write(1);
             delay_us(2);

             /* After master initiates read time slot, DS18B20 will begin
             * transmitting a 1 or 0 on bus */
             W1DQ_Input();

             /* DS18B20 bit sent by LSB (lower bit first) */
             if( W1DQ_Read() )
             {
                  byte |= 1<<i;
             }

             /* Read slot for >= 60us */
             delay_us(60);

             /* Release W1 bus to high */
             W1DQ_Output();
             W1DQ_Write(1);
             delay_us(2);
        }

        return byte;
}
```

```
static inline int DS18B20_Start_Convert(void)
{
        /* Master issues reset pulse and DS18B20s respond with presence pulse */
        if( 0 != DS18B20_Reset() )
         return -1;

        /* Master issues Skip ROM command */
        DS18B20_WriteByte(0xCC);

        /* Master issues Convert T command. */
        DS18B20_WriteByte(0x44);

        return 0;
}

static inline int DS18B20_Start_Read(uint8_t *buf, int bytes)
{
        /* Master issues reset pulse and DS18B20s respond with presence pulse */
        if( 0 != DS18B20_Reset() )
         return -1;

        /* Master issues Skip ROM command */
        DS18B20_WriteByte(0xCC);

        /* Master issues Read Scratchpad command. */
        DS18B20_WriteByte(0xBE);

        buf[0] = DS18B20_ReadByte();   /* Temperature LSB */
        buf[1] = DS18B20_ReadByte();   /* Temperature MSB */

        /*Don't care followed 7 bytes data */

        return 0;
}
```

```
int DS18B20_SampleData(float *temperature)
{
        uint8_t             byte[2];
        uint8_t             sign;
        uint16_t            temp;

        if( !temperature )
             return -1;

        if( 0 != DS18B20_Start_Convert() )
             return -2;

        if( 0 != DS18B20_Start_Read(byte, 2) )
             return -3;

        /* Temperature byte[0] is LSB, byte[1] is MSB, total 16 bit:
         * Byte[0]:  bit[3:0]: decimal bits,  bit[7:4]: integer bits
         * bYTE[1]:  bit[2:0]: integer bits,  bit[7:3]: sign bits
         */
        if( byte[1]> 0x7 ) /* bit[7:3] is 1 */
        {
             temp = ~(byte[1]<<8|byte[0]) + 1;  //补码
             sign=0; //温度为负
        }
        else
        {
             sign=1;//温度为正
             temp = byte[1]<<8 | byte[0];
        }

        /*byte[1]的低三位和 byte[0]的高四位组成温度值的整数部分，
        而 byte[0]的低四位为小数精度部分,且精度系数为 0.0625 */
        *temperature = (temp>>4) + (temp&0xF)*0.0625 ;
        if( !sign )
        {
             *temperature = -*temperature;
        }

        return 0;
}
```

2.4. 创建并编写 DS18B20 的驱动头文件 ds18b20.h

```
#ifndef INC_DS18B20_H_
#define INC_DS18B20_H_

extern int DS18B20_SampleData(float *temperature);

#endif /* INC_DS18B20_H_ */
```

2.5. DS18B20 测试代码

修改 main.c 文件，实现每隔 3s 采样当前的温度值并打印输出

```
... ...

在文件开始处添加用户头文件 ds18b20.h ：
/* USER CODE BEGIN Includes */
#include "ds18b20.h"
/* USER CODE END Includes */
... ...

在 main 开始处相应位置添加两个用来保存采样温度度值的变量：
int main(void)
{
  /* USER CODE BEGIN 1 */
  float       temperature;
  /* USER CODE END 1 */

  ... ...

  /* USER CODE BEGIN WHILE */
  sysled_hearbeat();
  beep_start(2, 300);
  printf("Start BearKE1 5G NB-IoT Board Example Program v1.0\r\n");
  while (1)
  {
          if( DS18B20_SampleData(&temperature) < 0 )
          {
              printf("ERROR: DS18B20 Sample Data failure\r\n");
          }
          else
          {
              printf("DS18B20 Sample Temperature: %.3f \r\n", temperature);
          }

          HAL_Delay(3000);
 ... ...
```

2.6. 运行测试

重新编译并烧录运行程序，使用串口调试助手监控串口输出，这时候会发现每隔 3s 单片机就会打印输出当前的温度值。将 DS18B20 传感器放到笔记本电脑的散热口处，我们会发现温度传感器会有明显变化。