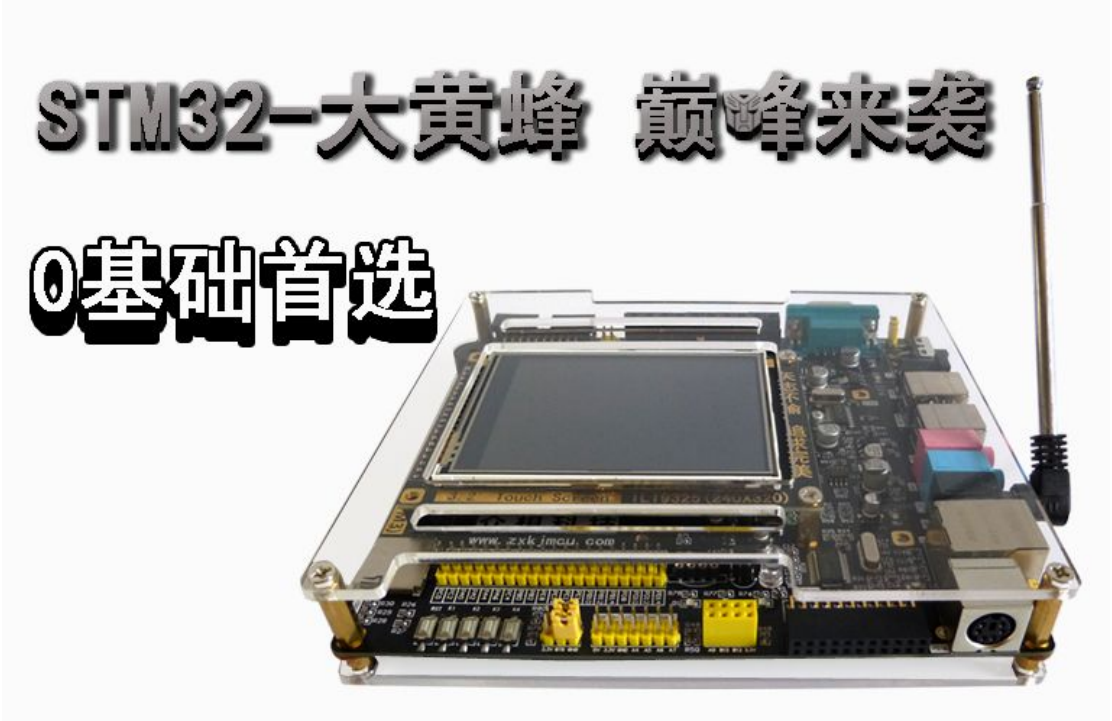
学 ARM 从 STM32 开始

STM32 开发板库函数教程--实战篇



官方网站：http://www.zxkjmcu.com
官方店铺：http://zxkjmcu.taobao.com
官方论坛：http://bbs.zxkjmcu.com
刘洋课堂：http://school.zxkjmcu.com

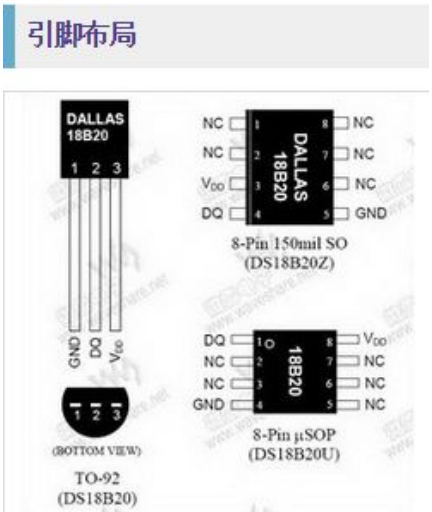
4.25 STM32 DS18B20 温度传感工作原理实验

4.25.1 概述

DS18B20 数字温度传感器提供 9~12 位摄氏温度的测量，拥有非易失性用户可编程最高与最低触发点告警功能。DS18B20 通过单总线实现通信，单总线通常是 DS18B20 连接 。它能够感应温度的范围为-55℃~+125℃，在-10℃~+85℃的测量的精度是±0.5℃，而且 DS18B20 可以直接从数据线上获取供电（寄生 电源）而不需要一个额外的外部电源。 因为每个 DS18B20 拥有一个独特的 64 序列号，因此它允许多个 DS18B20 在一条单总线上，可以实现组网多点测温。所以很方便使用一个微控制器来控制多个分布在较大范围内的 DS18B20。

4.25.2 DS18B20 通用特性

- 独特单总线接口，只需要一个端口引脚线即可实现通信。
- 每个器件的片上 ROM 有一个独特 64 位串行码存储。
- 不需要外围元件配合。
- 多点能力使分布式温度检测应用得到简化。
- 可以使用数据线供电，供电的范围 3.0V～5.5V。
- 测量温度的范围：-55℃～+125℃ （-67℉～+257℉）



- 负电压特性：电源极性接反时，芯片不会因为发热而烧坏，但不能正常工作。

4.25.2.1 DS18B20 测量操作

DS18B20 的核心功能是它的直接数字温度传感器器，为软件设置为 9、10、11 或者 12 位，相应的分辨率分别 0.5℃、0.25℃、0.125℃和 0.0625℃。上电时默认的分辨率是 12 位。DS18B20 上电时处于低电源的理想状态，初始化温度测量和 A—TO—D 转换，主机必须确保发送一个温度转换的指令[44H]。接下来开始转换，转换后的温度数据保存在暂存寄存器中的两字节温度寄存器中，然后 DS18B20 返回到它的理想状态。如果 ds18b20 有外部电源供电，主器件发送温度转换指令后要确保“读时隙”，DS18B20 在温度转换的处理中将返回一个 0 作为相应，转换完以后 将返回一个 1。如果 DS18B20 采用寄生电源供电，那么上面的注意事项不在应用，因为在 整个温度转换的过程中总线被一个强的上拉电流拉为高电平。这个数据表总线要求将会在 DS18B20 的供电部分做详细的解释 。

4.25.2.2 DS18B20 输出温度被校准为摄氏温度

准换程序。温度数据以 16 位完整的数据存储在两个温度寄存器中（见图 1）符号位 S 表 示测量的温度是正还是负，测量数据为正 S=0，测量数据为负 S=1。如果 DS18B20 被配置 为 12 位的分别率，那么问对寄存器中所有的数据都是有效的数据，对于 11 位的分辨率， 位 0 没有被定义，对于 10 位的分辨率，位 0 和位 1 没有被定义，而对于 9 位分辨率的， 位 2、1、0 位都没有定义，图 2 给出了一个分辨率为 12 位转换后的数字输出数据和相应读取的温度的值。

LS Byte

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}

MS Byte

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
S	S	S	S	S	2^6	2^5	2^4

图 1 温度寄存器格式

温度 /℃	二进制表示	十六进制表示
+125	0000 0111 1101 0000	07D0H
+85	0000 0101 0101 0000	0550H
+25.0625	0000 0001 1001 0000	0191H
+10.125	0000 0000 1010 0001	00A2H
+0.5	0000 0000 0000 0010	0008H
0	0000 0000 0000 1000	0000H
-0.5	1111 1111 1111 0000	FFF8H
-10.125	1111 1111 0101 1110	FF5EH
-25.0625	1111 1110 0110 1111	FE6FH
-55	1111 1100 1001 0000	FC90H

* 上电复位后温度寄存器的值为+85℃。

图 2 温度和数据的关系

4.25.2.3 DS18B20 报警操作信号：

在 DS18B20 温度转换以后，温度值将和用户定义存储在一字节 TH 和 TL 寄存器中两个完整的报警触发值相比较，见图 5。 符号位表明测量值是正还是负，对于正值 S=0， 对于负值 S=1。TH 和 TL 寄存器是非易失性的（E^2PROM）因此在器件掉点后能够保存数据。TH 和 TL 能够能够通过暂存器的字节 2 和 3 来访问，这正如这个数据表在存储器部解释的一样。在 TH 和 TL 的比较中温度寄存器中只使用了 11~4 位，因为 TH 和 TL 是 8 位寄存 器，如果测量的温度低于，或等于 TL，或高于 TH。报警情况将会发

生，同时 DS18B20 内部将设置符号标志，这个符号标志将在每次测量后都得到更新。因此，如果报警情况发生以后，每次温度转换完以后符号位将关闭。

4.25.2.4 DS18B20 64 位激光 ROM 编码

每一个 DS18B20 包含一个独特 64 位编码存储在 ROM 中。在 ROM 编码中低八位有效数据包含了 DS18B20 单总线的类码：28H。接下来的 48 位包含了一个序列码。高 8 位是用来计算开始 56ROM 编码的 CR（周期边界检测）字节。CRC 的详细说明在 CRC 的产生部分说明。

64 位编码具体位置分布情况

8 位 CRC	48 位序列号	8 位系列码

4.25.2.5 DS18B20 CRCD 校验

CRC 作为 DS18B20 64 位 ROM 码的一部分在暂存寄存器第九个字节中。CRC 是计算 ROM 码中开始的 56 位码，这包含 ROM 中绝大部分有意义的字节。暂存寄存器 CRC 是计算存储在暂存寄存器中的数据，因此当暂存寄存器中的数据发生改变时它也必须改变。CRC 提供总线主器件一种确认从 DS18B20 读出的数据有效的方法。为了证 实数据被正确的读出来，总线主器件必须接受到数据的 CRC，然后用这个值与每个 ROM 码的值相比较（读 ROM 时）或者与每个暂存寄存器的 CRC 相比较（读暂存寄存器）如果计算的 CRC 与读的 CRC 向匹配，那么说明接受的数据没有错误。比较 CRC 的值和决定下一步操作完全取决主器件，如果 DS18B20CRC（ROM 和暂存寄存器）与主器件产生的 CRC 值不匹配，在 DS18B20 内部没有一个电

路能够阻止指令序列的继续进行。

CRC（ROM 和暂存寄存器）等效的多项式函数如下：

$$CRC = X^8 + X^5 + X^4 + 1 \qquad (1)$$

主器件通过使用多项式产生器重新计算 CRC 并和来自 DS18B20 的 CRC 相比较。这个电路包含有移位寄存器和异或门，移位寄存器的位全部初始化为 0。以 ROM 码中最低有效位或者暂存寄存器中的字节 0 开始，每次有一位被移入到移位寄存器中，在移完 ROM 中第 56 位或者暂存寄存器字节 7 的最高有效位后，多项式产生器将得到重新计算的 CRC。接下来 8 位 ROM 码或者来自 DS18B20 暂存寄存器的 CRC 必须被移入这个电路。在这一点上，如果重新计算的 CRC 正确的，移位寄存器将的到全 0。详细应用请查询相关手册。

4.25.2.6 DS18B20 工作时序

一、初始化时序

主机首先发出一个 480-960 微秒的低电平脉冲，然后释放总线变为高电平，并在随后的 480 微秒时间内对总线进行检测，如果有低电平出现说明总线上有器件已做出应答。若无低电平出现一直都是高电平说明总线上无器件应答。

做为从器件的DS18B20在一上电后就一直在检测总线上是否有480-960微秒的低电平出现，如果有，在总线转为高电平后等待 15-60 微秒后将总线电平拉低 60-240 微秒做出响应存在脉冲，告诉主机本器件已做好准备。若没有检测到就一直在检测等待。

二、写操作

写周期最少为 60 微秒，最长不超过 120 微秒。写周期一开始做为主机先把总线拉低 1 微秒表示写周期开始。随后若主机想写 0，则继续拉低电平最少 60 微秒直至写周期结束，然后释放总线为高电平。若主机想写 1，在一开始拉低总线电平 1 微秒后就释放总线为高电平，一直到写周期结束。而做为从机的 DS18B20 则在检测到总线被拉底后等待 15 微秒然后从 15us 到 45us 开始对总线采样，在采样期内总线为高电平则为 1，若采样期内总线为低电平则为 0。

三、读操作

对于读数据操作时序也分为读 0 时序和读 1 时序两个过程。读时隙是从主机把单总线拉低之后，在 1 微秒之后就得释放单总线为高电平，以让 DS18B20 把数据传输到单总线上。DS18B20 在检测到总线被拉低 1 微秒后，便开始送出数据，若是要送出 0 就把总线拉为低电平直到读周期结束。若要送出 1 则释放总线为高电平。主机在一开始拉低总线 1 微秒后释放总线，然后在包括前面的拉低总线电平 1 微秒在内的 15 微秒时间内完成对总线进行采样检测，采样期内总线为低电平则确认为 0。采样期内总线为高电平则确认为 1。完成一个读时序过程，至少需要 60us 才能完成。

四、指令集

具体参考详细的 DS18B2（数字温度传感器）使用手册。

4.25.3 实验目的

通过这个实验我们要熟练使用数字温度传感器 DS18B20，通过程序设计要熟练掌握读写 DS18B20 温度传感器的方法和原理。再有次要目的就是要熟

练掌握器件时序图，因为所有程序的编写都要严格按照硬件工作的时序来设计，只有这样硬件电路才能正常工作。

4.25.4 硬件设计

CPU 的第 3 管脚直接控制和 DS18B20 数据端相连接，硬件设计简单实用。硬件设计见图 4.25.1 数字温度传感器原理图。

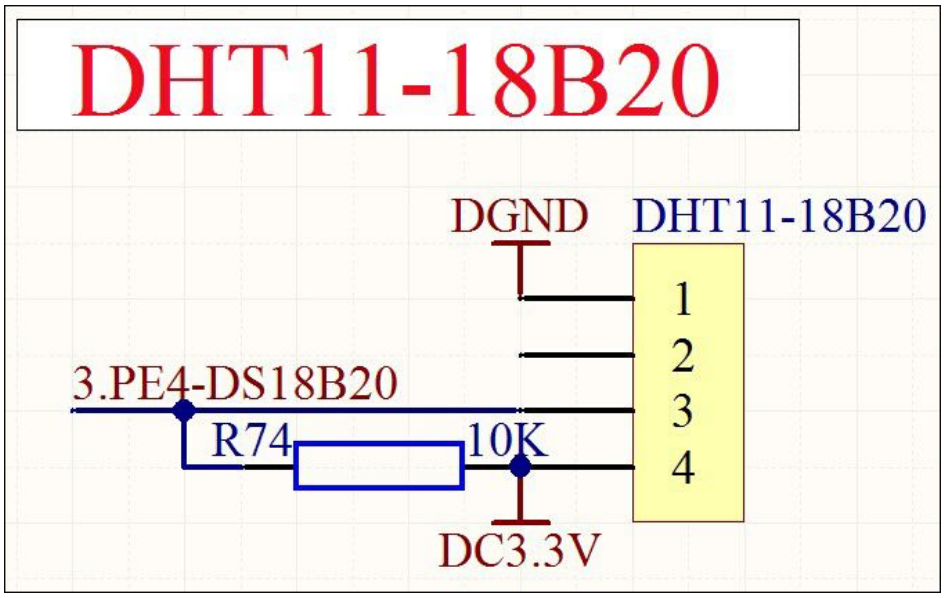


图 4.25.1 数字温度传感器原理图

4.25.5 软件设计

4.25.5.1 软件设计说明

利用实验板上集成的数字温度传感器电路，读取 DS18B20 的温度值，让后送到串口打印输出到计算机。这套程序严格按照 DS18B20 的工作时序编写。

在这节程序设计中，用到了外部中断函数；prinif 重定向打印输出函数； USART 串口通讯函数；定时器函数。

4.25.5.2 STM32 库函数文件

```
stm32f10x_gpio.c
stm32f10x_rcc.c
Misc.c  // 中断控制字（优先级设置）库函数
stm32f10x_exti.c // 外部中断库处理函数
stm32f10x_tim.c  // 定时器库处理函数
stm32f10x_usart.c  // 串口通讯函数
```

以上库文件包含了本次实验所有要用到的函数功能。

4.25.5.3 自定义头文件

```
pbdata.h
pbdata.c
```

我们已经创建了两个公共的文件，这两个文件主要存放我们自定义的公共函数和全局变量，以方便以后每个功能模块之间传递参数。

4.25.5.4 pbdata.h 文件里的内容是

```
#ifndef _pbdata_H
#define _pbdata_H

#include "stm32f10x.h"
#include "misc.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_tim.h"
#include "stm32f10x_usart.h"
#include "sttdio.h"
#include "ds18b20.h"

extern u8 dt;       //定义全局变量

void RCC_HSE_Configuration(void); //定义函数
void delay(u32 nCount);
void delay_us(u32 nus);
void delay_ms(u16 nms);

#endif
```

语句 #ifndef、#endif 是为了防止 pbdata.h 文件被多个文件调用时出现错误提示。如果不加这两条语句，当两个文件同时调用 pbdata 文件时，

会提示重复调用错误。“ds18b20.h”是我们自定义的，为了是在大程序中方便移植和功能分类独立新建。

4.25.5.5 pbdata.c 文件里的内容是

下面是 pbdata.c 文件详细内容，在文件开始还是引用“pbdata.h”文件。

```
    #include "pbdata.h"

    u8  dt=0;        //定义全局变量

    void RCC_HSE_Configuration(void) //HSE 作为 PLL 时钟，PLL 作为 SYSCLK
    {
       RCC_DeInit(); /*将外设 RCC 寄存器重设为缺省值 */
       RCC_HSEConfig(RCC_HSE_ON);   /*设置外部高速晶振（HSE） HSE 晶振打开
(ON)*/

       if(RCC_WaitForHSEStartUp() == SUCCESS) {  /*等待 HSE 起振,  SUCCESS：HSE
晶振稳定且就绪*/

       RCC_HCLKConfig(RCC_SYSCLK_Div1);/*设置 AHB 时钟(HCLK)RCC_SYSCLK_Div1——
—AHB 时钟 = 系统时*/
       RCC_PCLK2Config(RCC_HCLK_Div1); /*设置高速 AHB 时钟(PCLK2)RCC_HCLK_Div1
———APB2 时钟 = HCLK*/
       RCC_PCLK1Config(RCC_HCLK_Div2); /*设置低速 AHB 时钟(PCLK1)RCC_HCLK_Div2
———APB1 时钟 = HCLK / 2*/

       RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);/*设置 PLL 时钟源及
倍频系数*/
       RCC_PLLCmd(ENABLE);   /*使能 PLL */
       while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET) ; /*检查指定的 RCC
标志位(PLL 准备好标志)设置与否*/

       RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);  /*设置系统时钟（SYSCLK） */
       while(RCC_GetSYSCLKSource() != 0x08);     /*0x08：PLL 作为系统时钟 */

     }
    }

    void delay(u32 nCount)
    {
       for(;nCount!=0;nCount--);
    }
```

```
/*******************************************************************************
* 名    称：delay_us(u32 nus)
* 功    能：微秒延时函数
* 入口参数：u32  nus
* 出口参数：无
* 说    明：
* 调用方法：无
*******************************************************************************/
void delay_us(u32 nus)
{
    u32 temp;
    SysTick->LOAD = 9*nus;
    SysTick->VAL=0X00;//清空计数器
    SysTick->CTRL=0X01;//使能，减到零是无动作，采用外部时钟源
    do
    {
     temp=SysTick->CTRL;//读取当前倒计数值
    }while((temp&0x01)&&(!(temp&(1<<16))));//等待时间到达

    SysTick->CTRL=0x00; //关闭计数器
    SysTick->VAL =0X00; //清空计数器
}

/*******************************************************************************
* 名    称：delay_ms(u16 nms)
* 功    能：毫秒延时函数
* 入口参数：u16 nms
* 出口参数：无
* 说    明：
* 调用方法：无
*******************************************************************************/
void delay_ms(u16 nms)
{
     //注意 delay_ms 函数输入范围是 1-1863
    //所以最大延时为 1.8 秒
    u32 temp;
    SysTick->LOAD = 9000*nms;
    SysTick->VAL=0X00;//清空计数器
    SysTick->CTRL=0X01;//使能，减到零是无动作，采用外部时钟源
    do
    {
     temp=SysTick->CTRL;//读取当前倒计数值
    }while((temp&0x01)&&(!(temp&(1<<16))));//等待时间到达
    SysTick->CTRL=0x00; //关闭计数器
```

```
    SysTick->VAL =0X00; //清空计数器
}
```

4.25.6 STM32 系统时钟配置 SystemInit()

每个工程都必须在开始时配置并启动 STM32 系统时钟。

4.25.7 GPIO 引脚时钟使能

```
SystemInit();//72m
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE,ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);//设置串口 1 时钟使
能

RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//功能复用 IO 时钟使能
```

本节实验用到了 PE 端口，所以要把 PE 端口的时钟打开；串口 1 时钟源是通过 APB2 预分频器得到的，串口 1 时钟初始化；因为要与外部芯片通讯，所以要打开功能复用时钟。

4.25.8 GPIO 管脚电平控制函数

在主程序中采用 while(1)循环语句，等待 DS18B20 传感器数据读入，当数据读入后直接送打印输出计算机显示。主程序打印输出“当前环境温度: %0.4lf ℃\r\n ,temp”。

```
  while(1)
   {
      temp=DS18B20_Get_wd();
     printf("当前环境温度: %0.4lf ℃\r\n",temp);
   }
}
```

4.25.9 stm32f10x_it.c 文件里的内容

在中断处理 stm32f10x_it.c 文件里中有串口 1 子函数。

```
#include "stm32f10x_it.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_rcc.h"
#include "misc.h"
#include "pbdata.h"

void NMI_Handler(void)
{
}

void USART1_IRQHandler(void)
{
   if(USART_GetITStatus(USART1,USART_IT_RXNE)!=RESET)
   {
      USART_SendData(USART1,USART_ReceiveData(USART1));
      while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
   }
}
```

4.25.10 ds18b20.h 文件里的内容是

函数 ds18b20.h 在这里是为了数字温度传感器声明变量自定义的功能函数，

ds18b20.h 的内容如下：

```
#ifndef  _DS18B20_H
#define _DS18B20_H
#include "pbdata.h"

#define IO_DS18B20 GPIO_Pin_4  //定义中间变量 IO_DS18B20 代替 GPIO_Pin_4，方便程序移植
#define GPIO_DS18B20 GPIOE  //定义中间变量 GPIO_DS18B20 代替 GPIOE，方便程序移植
#define DS18B20_DQ_High GPIO_SetBits(GPIO_DS18B20,IO_DS18B20)//定义中间变量 DS18B20_DQ_High 拉高函数，在以后的程序中引用它，就是间接调用 GPIO_SetBits(GPIO_DS18B20,IO_DS18B20)函数
#define DS18B20_DQ_Low GPIO_ResetBits(GPIO_DS18B20,IO_DS18B20)//同上

void DS18B20_IO_IN(void);//声明输入函数
void DS18B20_IO_OUT(void);
u8 DS18B20_Read_Byte(void);
```

```
void DS18B20_Write_Byte(u8 dat);
void DS18B20_Reset(void);
double DS18B20_Get_wd(void);

#endif
```

4.25.11 ds18b20.c 文件里的内容是

自定义函数 ds18b20.c 中是读取 DS18B20 的过程，他是读写数组温度传感器的核心程序，其内容如下：

```
    #include "pbdata.h"

    void DS18B20_IO_IN(void) //自定定义 DS18B20 输入函数，下面是函数初始化
    {
       GPIO_InitTypeDef GPIO_InitStructure;

       GPIO_InitStructure.GPIO_Pin=IO_DS18B20;
       GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING;
       GPIO_Init(GPIO_DS18B20,&GPIO_InitStructure);
    }

    void DS18B20_IO_OUT(void) //自定定义 DS18B20 输出函数，下面是函数初始化
    {
       GPIO_InitTypeDef GPIO_InitStructure;

       GPIO_InitStructure.GPIO_Pin=IO_DS18B20;
       GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;//速度
       GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP; //推挽输出
       GPIO_Init(GPIO_DS18B20,&GPIO_InitStructure);
    }

    u8 DS18B20_Read_Byte(void)
    {
       u8 i=0,TempData=0;//TempData 临时变量，把读回来的数据暂时存在这个变量
中
       for(i=0;i<8;i++)  //严格按照 DS18B20 时序图步骤编写程序
       {
          TempData>>=1;

          DS18B20_IO_OUT();//输出模式
```

```
        DS18B20_DQ_Low;    //拉低
        delay_us(4);//延时 4 微妙
        DS18B20_DQ_High;//抬高，释放总线
        delay_us(10);//延时 10 微妙
        DS18B20_IO_IN();//转换为输入模式

        if(GPIO_ReadInputDataBit(GPIO_DS18B20,IO_DS18B20)==1)
        {
           TempData|=0x80;//读数据 从低位开始，最高位或上 0x80
        }

        delay_us(45);//延时 45 微妙
    }

    return TempData;
}

void DS18B20_Write_Byte(u8 dat)
{
    u8 i=0;
    DS18B20_IO_OUT();//输出

    for(i=0;i<8;i++)
    {
        DS18B20_DQ_Low;    //拉低
        delay_us(15);//延时 15 微妙

        if((dat&0x01)==1)
        {
           DS18B20_DQ_High;
        }
        else
        {
           DS18B20_DQ_Low;
        }
        delay_us(60);//延时 60 微妙
        DS18B20_DQ_High;

        dat>>=1;//准备下一位数据的发送
    }

void DS18B20_Reset(void)
{
    DS18B20_IO_OUT();//输出
```

```
    DS18B20_DQ_Low;
    delay_us(480);//延时 480 微妙
    DS18B20_DQ_High;
    delay_us(480);//延时 480 微妙
}

double DS18B20_Get_wd(void)
{
    u8 TL=0,TH=0;
    u16 temp=0;
    double wd=0;

    DS18B20_Reset();//复位
    DS18B20_Write_Byte(0xCC); //跳过 ROM 命令
    DS18B20_Write_Byte(0x44); //温度转换命令

    delay_ms(800);//延时 800 毫秒
    DS18B20_Reset();//复位
    DS18B20_Write_Byte(0xCC); //跳过 ROM 命令
    DS18B20_Write_Byte(0xBE); //读温度命令

    TL=DS18B20_Read_Byte();//LSB
    TH=DS18B20_Read_Byte();//MSB

    temp=TH;
    temp=(temp<<8)+TL;

    if((temp&0xF800)==0xF800)//负温度判断
    {
        temp=~temp;
        temp=temp+1;
        wd=temp*(-0.0625);
    }
    else
    {
        wd=temp*0.0625;
    }

    return wd;
}
```

4.25.12 main.c 文件里的内容是

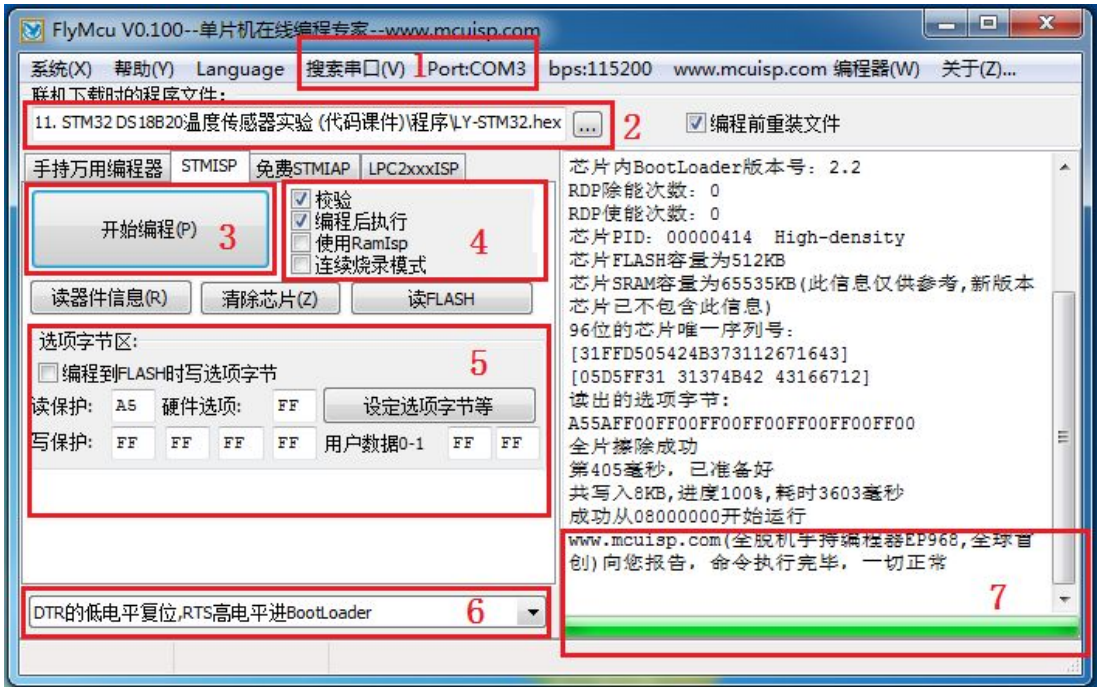
```
#include "pbdata.h"

void RCC_Configuration(void);
void GPIO_Configuration(void);
void NVIC_Configuration(void);
void USART_Configuration(void);

int fputc(int ch,FILE *f)
{
   USART_SendData(USART1,(u8)ch);
   while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
   return ch;
}

int main(void)
{
   double temp=0;
   RCC_Configuration(); //系统时钟初始化
   GPIO_Configuration();//端口初始化
   USART_Configuration();
   NVIC_Configuration();

   while(1)
   {
      temp=DS18B20_Get_wd();
     printf("当前环境温度: %0.4lf ℃\r\n",temp);
   }
}

void RCC_Configuration(void)
{
    SystemInit();//72m
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE,ENABLE);
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
}

void GPIO_Configuration(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
   //LED
   GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9;//TX
   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;//RX
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA,&GPIO_InitStructure);
}

void NVIC_Configuration(void)
{
      NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_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_HardwareFlowContr
ol_None;
    USART_InitStructure.USART_Mode=USART_Mode_Rx|USART_Mode_Tx;

    USART_Init(USART1,&USART_InitStructure);
    USART_ITConfig(USART1,USART_IT_RXNE,ENABLE);
    USART_Cmd(USART1,ENABLE);
    USART_ClearFlag(USART1,USART_FLAG_TC);
}
```

4.25.13 程序下载

请根据下图所指向的 7 个重点区域配置。其中（1）号区域根据自己机

器的实际情况选择，我的机器虚拟出来的串口号是 COM3。（2）号区域请自己选择程序所在的文件夹。（7）号区域当程序下载完后，进度条会到达最右边，并且提示一切正常。（4、5、6）号区域一定要按照上图显示的设置。当都设置好以后就可以直接点击（3）号区域的开始编程按钮下传程序了。



本节实验的源代码在光盘中：（LY-STM32 光盘资料\1.课程\2,外设篇\外设篇 11. STM32 DS18B20 温度传感实验\程序）

4.25.14 实验效果图

在这个实验中我们把 DS18B20 直接插入到大黄蜂实验板 U16 接口，打开众想科技多功能监控软件，选择 1 号区域串口通讯功能，在 2 号区域设置通讯格式，这时打开串口，在 3 号区域是可以观察到采集上来的温度值。现在室温是 24.3 摄氏度，很正常。这说明程序设计很成功。具体实验感官请参考下图。

图 4.25.14.1 DS18B20 读取温度值实验效果图二