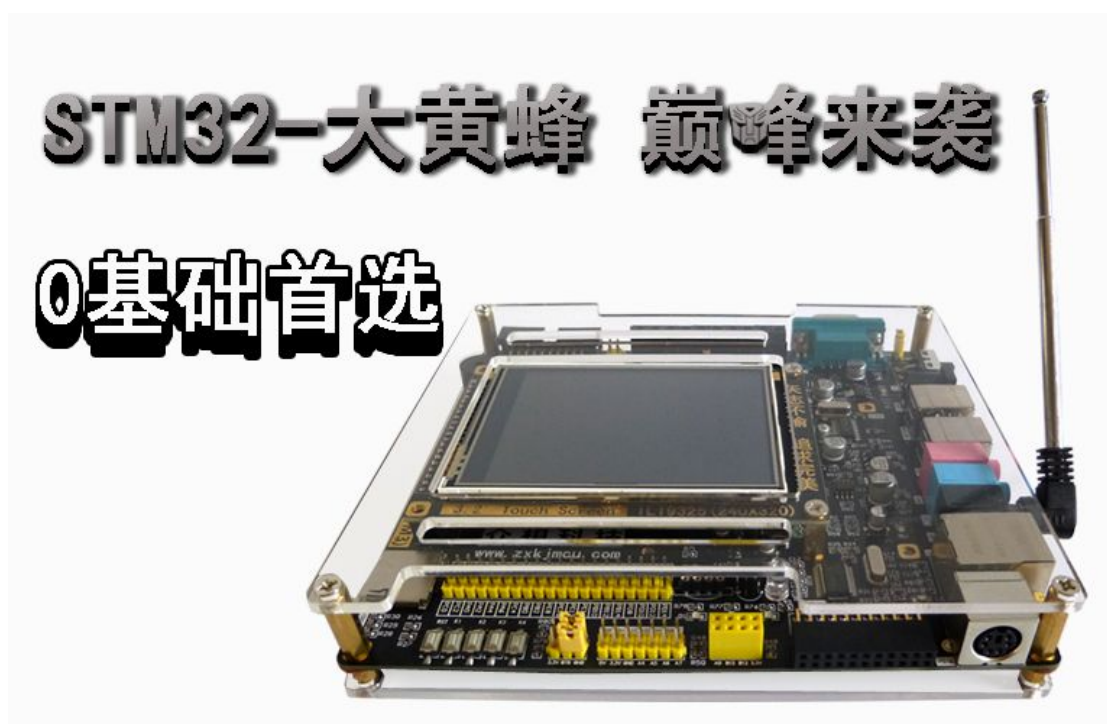


学 ARM 从 STM32 开始

STM32 开发板库函数教程—模块篇



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>

目 录

5.04.1 概述.....	3
5.04.2 实验目的.....	5
5.04.3 硬件设计.....	5
5.04.4 软件设计.....	6
5.04.5 STM32 系统时钟配置 SystemInit()	9
5.04.6 GPIO 引脚时钟使能.....	9
5.04.7 GPIO 管脚电平控制函数.....	9
5.04.8 stm32f10x_it.c 文件里的内容是.....	10
5.04.9 ds18b20.h 文件里的内容是.....	10
5.04.10 ds18b20.c 文件里的内容是.....	11
5.04.11 main.c 文件里的内容是.....	16
5.04.12 程序下载.....	18
5.04.13 实验效果图.....	19

5.04 防水型 DS18B20 程序设计

5.04.1 概述

美国 Dallas 半导体公司生产的数字化温度传感器 DS18B20, 采用导热性高的密封胶灌封, 保证了温度传感器的高灵敏性, 极小的温度延迟。该温度传感器支持“一线总线”接口 (1-Wire), 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。适合于恶劣环境的现场温度测量。DS18B20 数字温度传感器都具有唯一的编号, 温度采集设备通过编号来识别对应的温度传感器。



在外设篇我们详细介绍了 DS18B20 数字温度传感器, 下面重复简单介绍一下 DS18B20 独特的优点:

(1) 采用单总线的接口方式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。单总线具有经济性好, 抗干扰能力强, 适合于恶劣环境的现场温度测量, 使用方便等优点, 使用户可轻松地组建传

传感器网络，为测量系统的构建引入全新概念。

(2) 测量温度范围宽，测量精度高 DS18B20 的测量范围为 $-55\text{ }^{\circ}\text{C} \sim +125\text{ }^{\circ}\text{C}$ ；在 $-10\text{ }^{\circ}\text{C} \sim +85\text{ }^{\circ}\text{C}$ 范围内，精度为 $\pm 0.5\text{ }^{\circ}\text{C}$ 。

(3) 在使用中不需要任何外围元件。

(4) 持多点组网功能 多个 DS18B20 可以并联在惟一的单线上，实现多点测温。

(5) 供电方式灵活 DS18B20 可以通过内部寄生电路从数据线上获取电源。因此，当数据线上的时序满足一定的要求时，可以不接外部电源，从而使系统结构更趋简单，可靠性更高。

(6) 测量参数可配置 DS18B20 的测量分辨率可通过程序设定 9~12 位。

(7) 耐压特性电源极性接反时，温度计不会因发热而烧毁，但不能正常工作。

(8) 掉电保护功能 DS18B20 内部含有 EEPROM，在系统掉电以后，它仍可保存分辨率及报警温度的设定值。

DS18B20 具有体积更小、适用电压更宽、更经济、可选更小的封装方式，更宽的电压适用范围，适合于构建自己的经济的测温系统，因此也就被设计者们所青睐。

防水探头采用全新原装进口 DS18B20 温度传感器芯片，芯片每个引脚均用热缩管隔开，防止短路，内部密封胶，防水防潮，不锈钢头常规的引线 1 米，钢管 $\Phi 6 \times 50\text{mm}$ 。每个探头经过严格测试，DC 3.0V~5.5V 供电。

5.04.2 实验目的

通过我们选用的是防水型 DS18B20 传感器，实验时把它置于盛装热水（冷水）水杯，测量水温。我们设计好的程序把测量结果输出打印至计算机显示。观察水温的变化和传感器的灵敏度是否符合要求。

5.04.3 硬件设计

选用大黄蜂实验板，把防水型 DS18B20 通过 PWM 端子连接到实验板上，通过程序设计把采集到的水温打印输出到计算机显示。硬件设计见图 5.04.1 防水型 DS18B20 连线图。

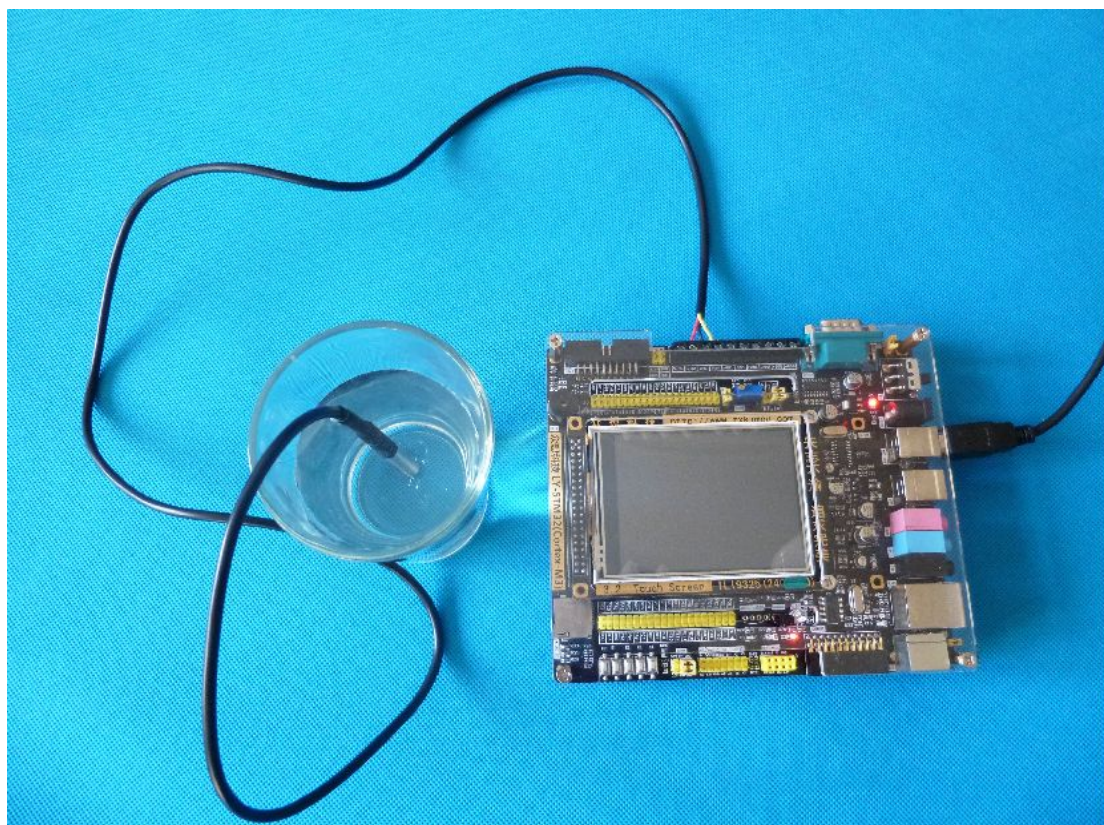


图 5.04.1 防水型 DS18B20 连线图

附表 1 接线方法

STM32 开发板端子	防水温度传感器 (DS18B20)
3.3V	1. 红色线
GND	2. 黑色线
PWM (PA8)	3. 黄色线

5.04.4 软件设计

5.04.4.1 软件设计说明

防水型 DS18B20 是采购的成品，直接和大黄蜂实验板连接好后可以进行程序设计了，按照在《外设篇 10. STM32 DS18B20 温度传感工作原理》中的讲解我们就可以很轻松编写出这篇采集程序，我们还是采用库函数的方式进行程序设计。

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。

5.04.4.2 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
```

本节实验及以后的实验我们都是用到库文件，其中 `stm32f10x_gpio.h` 头文件包含了 GPIO 端口的定义。`stm32f10x_rcc.h` 头文件包含了系统时钟配置函数以及相关的外设时钟使能函数，所以我们要把这两个头文件对应的 `stm32f10x_gpio.c` 和 `stm32f10x_rcc.c` 加到工程中；`Misc.c` 库函数主要包含了中断优先级的设置，`stm32f10x_exti.c` 库函数主要包含了外部中断设置参数，`tm32f10x_tim.c` 库函数主要包含定时器设置，`tm32f10x_usart.c` 库函数主要包含串行通讯设置，这些函数也要添加到函数库中。以上库文件包含了本次实验所有要用到的函数使用功能。

5.04.4.3 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。

5.04.4.4 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 "stdio.h"
#include "ds18b20.h"

//定义函数
void delay_us(u32 nus); //微妙延时程序函数
void delay_ms(u16 nms); //毫秒延时程序函数

#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。

5.04.4.5 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"
/*****
* 名 称: delay_us(u32 nus)
* 功 能: 微妙延时函数
* 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
}
```

5.04.5 STM32 系统时钟配置 SystemInit()

我们总在强调，每个工程都必须在开始时配置并启动 STM32 系统时钟，这次也不例外。

5.04.6 GPIO 引脚时钟使能

```
SystemInit(); //72m
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); //设置 PA 输出端口
RCC_APB1PeriphClockCmd(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 PA 端口，所以要把 PA 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。

5.04.7 GPIO 管脚电平控制函数

在主程序中采用 while(1) 循环语句，等待外部中断的到来后，主程序中读取缓冲区的温度值，并就打印输出到屏幕。

```
while(1)
{
    temp=DS18B20_Get_wd();
    printf("当前水环境温度: %.4lf °C\r\n", temp);
}
```

5.04.8 stm32f10x_it.c 文件里的内容是

在中断处理 stm32f10x_it.c 文件里中仅串口 1 子函数非空，进入中断处理函数后，只有串口 1 有参数输出。

```
/* Includes
-----*/
#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);
    }
}
```

5.04.9 ds18b20.h 文件里的内容是

函数 ds18b20.h 在这里是为了红外程序自定义的功能函数，ds18b20.h 的内容如下：

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

#define IO_DS18B20 GPIO_Pin_8//定义中间变量，放编程序移植，用
“IO_DS18B20”代替“GPIO_Pin_8”
#define GPIO_DS18B20 GPIOA

#define DS18B20_DQ_High GPIO_SetBits(GPIO_DS18B20, IO_DS18B20)
#define DS18B20_DQ_Low GPIO_ResetBits(GPIO_DS18B20, IO_DS18B20)

void DS18B20_IO_IN(void);
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);//温度计算子函数

#endif
```

5.04.10 ds18b20.c 文件里的内容是

我们先详细介绍 DS18B20 时序图，然后再按照时序图要求编写程序。

5.04.10.1 DS18B20 初始化时序

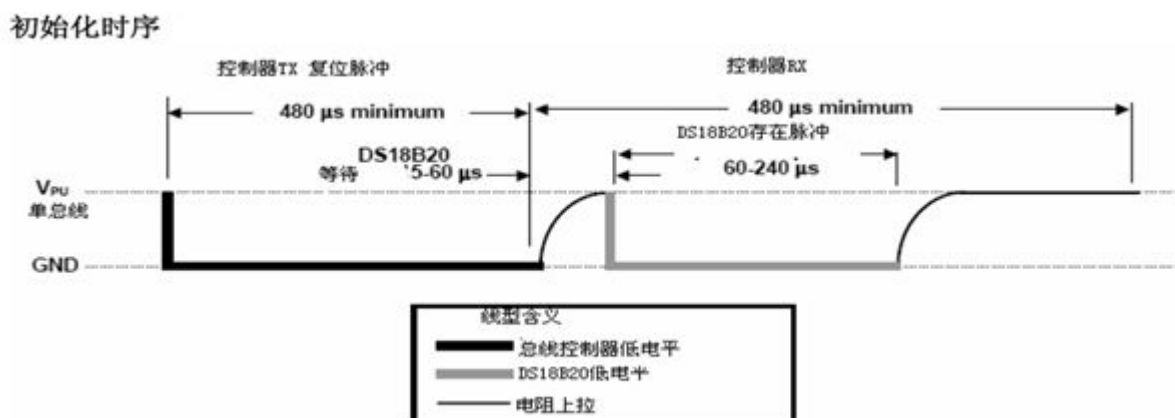


图 5.04.2 DS18B20 初始化时序

- (1). 数据线拉到低电平“0”。
- (2). 延时 480 微妙（该时间的时间范围可以从 480 到 960 微妙）。
- (3). 数据线拉到高电平“1”。
- (4). 延时等待 80 微妙。如果初始化成功则在 15 到 60 微妙时间内产生一个由 DS18B20 所返回的低电平“0”。根据该状态可以来确定它的存在，但是应注意不能无限的进行等待，不然会使程序进入死循环，所以要进行超时判断。
- (5). 若 CPU 读到了数据线上的低电平“0”后，还要做延时，其延时的时间从发出的高电平算起（第（3）步的时间算起）最少要 480 微妙。

5.04.10.2 DS18B20 读时序

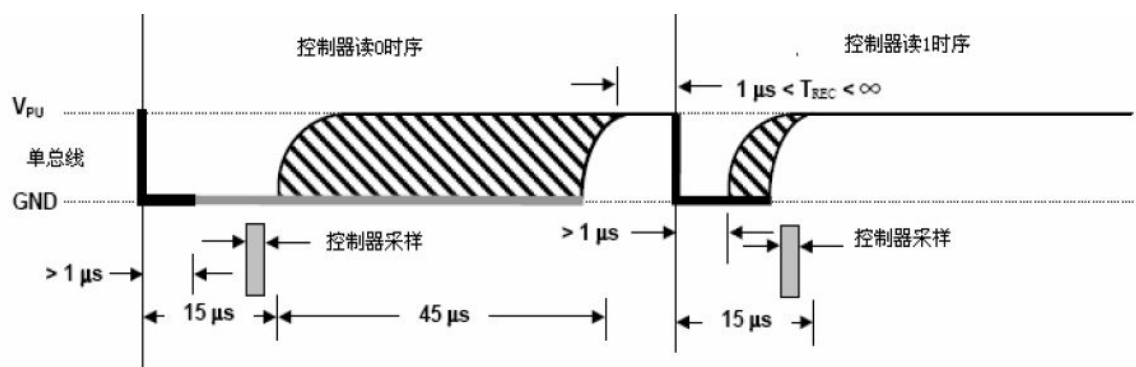


图 5.04.3 DS18B20 读时序

- (1). 将数据线拉低“0”。
- (2). 延时 4 微妙。
- (3). 将数据线拉高“1”, 释放总线准备读数据。
- (4). 延时 10 微妙。
- (5). 读数据线的状态得到 1 个状态位, 并进行数据处理。
- (6). 延时 45 微妙。
- (7). 重复 1~7 步骤, 直到读完一个字节。

5.04.10.3 DS18B20 写时序

- 1). 数据线先置低电平“0”
- (2). 延时 15 微妙。
- (3). 按从低位到高位顺序发送数据(一次只发送一位)。
- (4). 延时 60 微妙。
- (5). 将数据线拉到高电平。
- (6). 重复 1~5 步骤, 直到发送完整的字节。
- (7). 最后将数据线拉高。

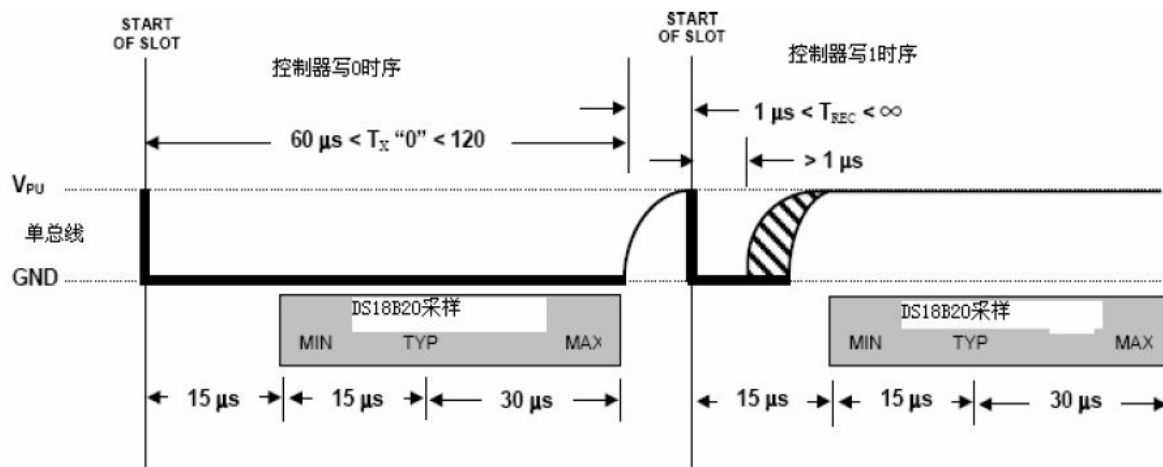


图 5.04.4 DS18B20 写时序

自定义函数 ds18b20.c 的内容如下：

```
#include "pbdata.h"

void DS18B20_IO_IN(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

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

void DS18B20_IO_OUT(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    GPIO_InitStructure.GPIO_Pin=IO_DS18B20;
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;//设置工作频率
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;//推挽输出
    GPIO_Init(GPIO_DS18B20,&GPIO_InitStructure); //初始化
}

u8 DS18B20_Read_Byte(void) //自定义读 DS18B20 子函数
{
    u8 i=0, TempData=0;//TempData 是临时变量，把读回来的数据存储在 TempData
    for(i=0;i<8;i++)
    {
```

中

```
TempData>>=1;//读数据之前一定先把临时变量向右移动一位，存储最低位
数据
    DS18B20_IO_OUT();//把总线置为输出模式
    DS18B20_DQ_Low;    //拉低
    delay_us(4);//延时 4 微妙
    DS18B20_DQ_High;
    delay_us(10);//延时 10 微妙
    DS18B20_IO_IN();//把总线置为输入模式

    if(GPIO_ReadInputDataBit(GPIO_DS18B20, IO_DS18B20)==1)//读输入的数据，如果为 1 就顺序执行
    {
        TempData|=0x80;//读数据 如果是 1，就把 TempData 或上 0x80，从低位
        开始
    }
    delay_us(45);//延时 45 微妙
}
return TempData;
}

void DS18B20_Write_Byte(u8 dat)//自定义写 DS18B20 子函数，一次写 8 位
{
    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)//数据与上 0x01，保留最低位，为发送做准备，如果是
        1，就顺序执行
        {
            DS18B20_DQ_High;//发送高电平
        }
        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)//复位子函数  
{  
    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)//返回一个 double 类型的温度值，写命令子函数  
{  
    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); //发送命令要求 DS18B20 进行温度转换  
  
    delay_ms(800); //延时 800 毫秒  
    DS18B20_Reset(); //发送读命令之前也要先复位  
    DS18B20_Write_Byte(0xCC); //跳过 ROM 命令  
    DS18B20_Write_Byte(0xBE); //读温度命令  
  
    TL=DS18B20_Read_Byte(); //读低字节数据 LSB  
    TH=DS18B20_Read_Byte(); //读高字节数据 MSB  
  
    temp=TH;  
    temp=(temp<<8)+TL; //组合起来是一个 16 位温度值  
  
    if((temp&0xF800)==0xF800) //负温度判断  
    {  
        temp=~temp; //数据取反  
        temp=temp+1; //数据再加 1  
        wd=temp*(-0.0625); //负温度真实值  
    }  
    else  
    {  
        wd=temp*0.0625;  
    }  
    return wd;  
}
```

```
}
```

5.04.11 main.c 文件里的内容是

```
#include "pdata.h"

void RCC_Configuration(void); //声明
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 °C\r\n", temp);
    }
}

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

void GPIO_Configuration(void)
```

```
{
    GPIO_InitTypeDef GPIO_InitStructure;

    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_HardwareFlowControl_None;

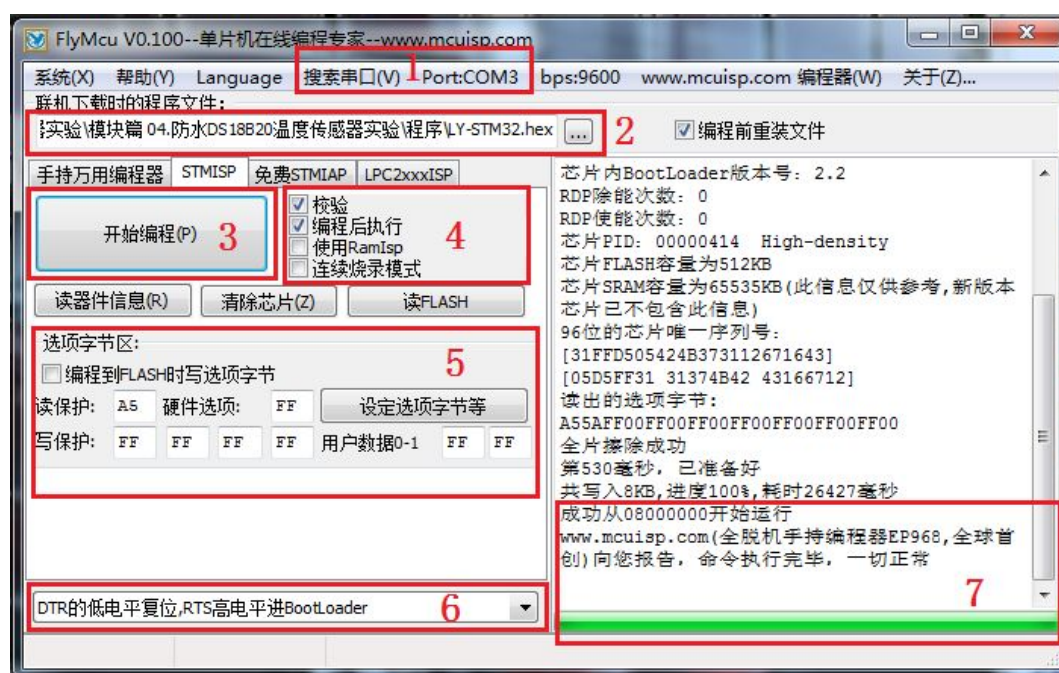
    USART_InitStructure.USART_Mode=USART_Mode_Rx|USART_Mode_Tx;

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

5.04.12 程序下载

在这一章节中要掌握 DS18B20 的工作时序，了解常用的温度传感器功能和原理。

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代码所在的文件夹。（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载程序了。



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\模块篇 04. 防水 DS18B20 温度传感器实验\程序）

5.04.13 实验效果图

把防水温度计放入盛温水的烧杯中，如“图 5.04.13.1 防水 DS18B20 测试水温实验效果图”，打开众想科技多功能监控软件，然后打开串口，我们在接收区可以观察到实测采集到的水温数据，水温大约 60 摄氏度左右，说明我们防水型 DS18B20 工作正常，程序编写的很成功。

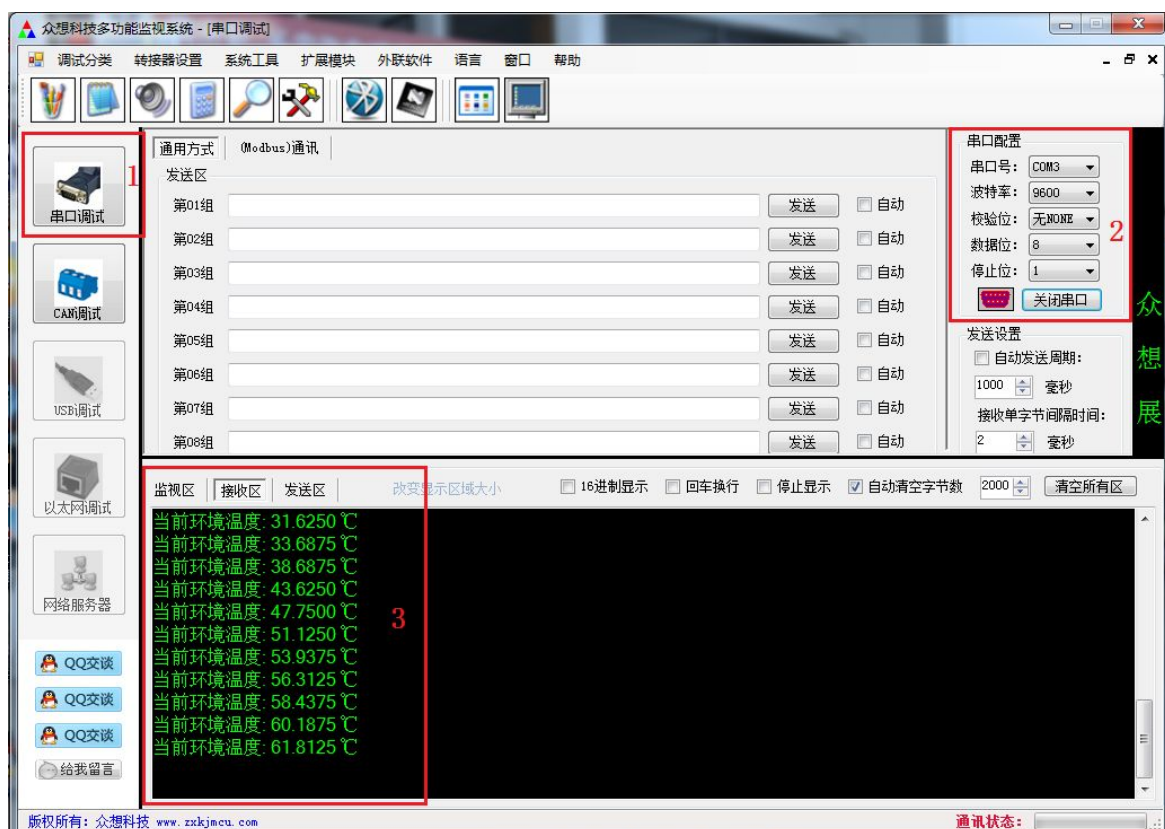


图 5.04.13.1 防水 DS18B20 测试水温实验效果图