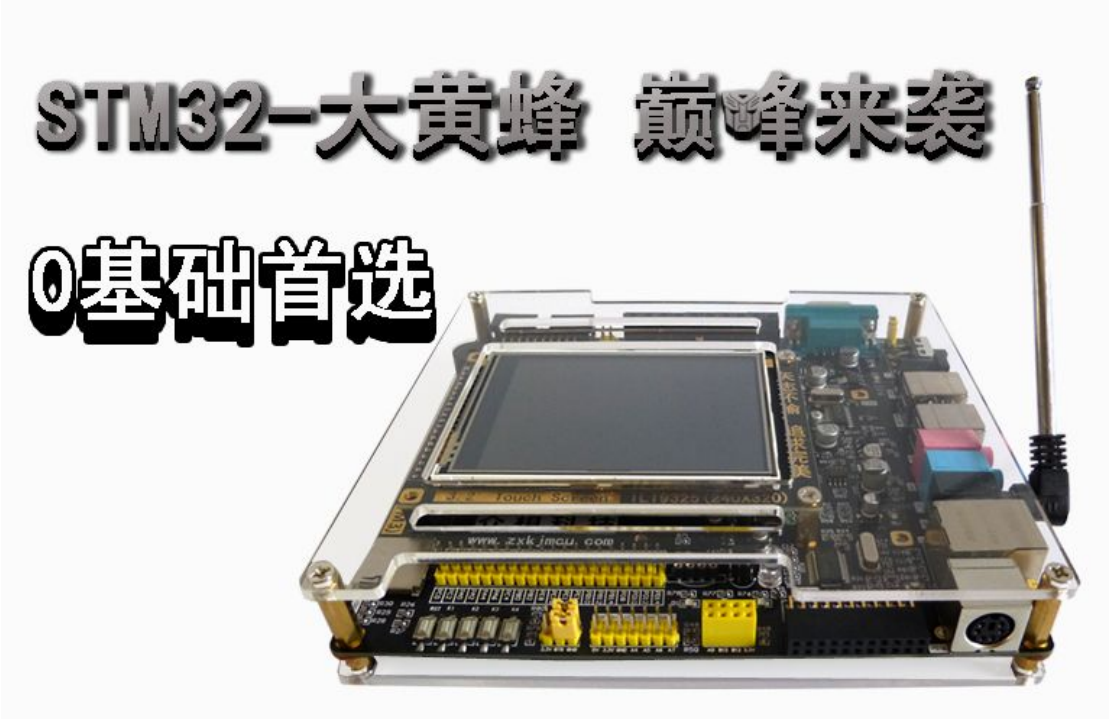
# 学 ARM 从 STM32 开始

## 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

# 4.24 STM32 红外线发射实验

## 4.24.1 概述

红外线遥控器已经被广泛的使用在各类型的家电产品上，它的出现给使用电器提供了很多的便利。一些彩电红外遥控器，其红外发光管的工作脉冲占空比约为 1/3-1/4；一些电器产品红外遥控器，其占空比是 1/10。减小脉冲占空比还可使小功率红外发光二极管的发射距离增加。

红外编码调制通常有通过脉冲宽度来实现信号调制的脉宽调制（PWM）和通过脉冲串之间的时间间隔来实现信号调制（PPM)两种方法。

### 4.24.1.1 红外遥控载波频率

红外遥控常用的载波频率为 38kHz。详细参考《4.23 STM32 外设篇-红外线接收工作原理及程序设计》

## 4.24.2 数据格式

具体参考《4.23 STM32 外设篇-红外线接收工作原理及程序设计》。

## 4.24.3 实验目的

我们用定时器程序产生 38KHz 的载波频率，使红外发射管发射的编码频率位 38KHz。达到红外编码的频率要求。

## 4.24.4 硬件设计

CPU 的第 59 管脚直接控制三极管 Q6 的通断频率来使红外发光二极管

（IrDA）跟随发光。。硬件设计见图 4.24.1　红外线发送原理图。

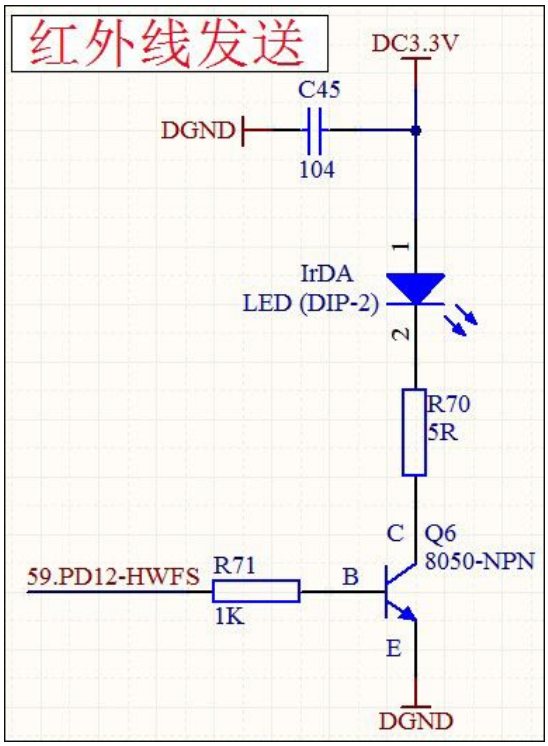



图 4.24.1　红外线发送原理图

## 4.24.5 软件设计

### 4.24.5.1 软件设计说明

利用实验板上集成的红外线发送电路，通过定时器模拟出来 38KHz 的调制信号，使红外发光二极管（IrDA）跟随这个频率发射红外线。

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。

### 4.24.5.2 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。

### 4.24.5.3 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。

### 4.24.5.4 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 "hw_fs.h"

extern u8 dt;       //定义全局变量
extern u32 hw_jsm;  //定义全局变量
extern u8  hw_jsbz; //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。“hw_fs.h”是我们自定义的，为了是更明确地说明红外发送这个功能，独立新建。

### 4.24.5.5 pbdata.c 文件里的内容是

pbdata.c 文件的开头先定义全局变量“u32 hw_jsm=0；u8 hw_jsbz=0”在整个程序中要调用他们。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
u32 flag=0;     //定义全局变量
u8  js_count=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
}
```

## 4.24.6 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。

## 4.24.7 GPIO 引脚时钟使能

```
SystemInit();//72m
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD,ENABLE);
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
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
```

本节实验用到了 PD 端口，所以要把 PD 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。

## 4.24.8 GPIO 管脚电平控制函数

在主程序中采用 while(1)循环语句，等待定时器中断的到来，当定时器中断来了以后，主程序打印输出“字符串输出 d4=%s\r\n”，如果是说明有红外线发送成功。

```
while(1)
  {
    //printf("字符串输出 d4=%s\r\n",d4);
    hw_fs(dt);
    delay_ms(1000);
  }
```

## 4.24.9 stm32f10x_it.c 文件内容解读

在中断处理 stm32f10x_it.c 文件里中有串口 1 子函数非空和 TIM2 定时器中断处理程序，进入中断处理函数后，按照顺序执行。在介绍程序前我们先熟悉一个 GPIO_ResetBits 函数,这个函数在以下的程序中起到很重要的作用。

### 4.24.9.1 GPIO_ResetBits 函数说明

表 1 函数 GPIO_ResetBits

| | |
|---|---|
| 函数名称 | GPIO_ResetBits |
| 函数原型 | void GPIO_ResetBits(GPIO_TypeDef* GPIOx, u16 GPIO_Pin) |
| 功能描述 | 清除指定的数据端口位 |
| 输入参数 1 | GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设 |
| 输入参数 2 | GPIO_Pin：待清除的端口位<br>该参数可以取 GPIO_Pin_x(x 可以是 0-15)的任意组合<br>参阅 Section：GPIO_Pin 查阅更多该参数允许取值范围 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

例：

```
/* Clears the GPIOA port pin 10 and pin 15 */
GPIO_ResetBits(GPIOA, GPIO_Pin_10 | GPIO_Pin_15);
```

### 4.24.9.2 GPIO WriteBit 函数说明

表 2 函数 GPIO_WriteBit

| | |
|---|---|
| 函数名称 | GPIO_WriteBit |
| 函数原型 | void GPIO_WriteBit(GPIO_TypeDef* GPIOx, u16 GPIO_Pin, BitAction BitVal) |
| 功能描述 | 设置或者清除指定的数据端口位 |
| 输入参数 1 | GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设 |
| 输入参数 2 | GPIO_Pin：待设置或者清除指的端口位<br>该参数可以取 GPIO_Pin_x(x 可以是 0-15)的任意组合<br>参阅 Section：GPIO_Pin 查阅更多该参数允许取值范围 |
| 输入参数 3 | BitVal：该参数指定了待写入的值<br>该参数必须取枚举 BitAction 的其中一个值<br>Bit_RESET：清除数据端口位<br>Bit_SET： 设置数据端口位 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |

| 被调用函数 | 无 |
|---|---|

例：

```
/* Set the GPIOA port pin 15 */
GPIO_WriteBit(GPIOA, GPIO_Pin_15, Bit_SET);
```

### 4.24.9.3 GPIO_ReadInputData 函数说明

表 3 函数 GPIO_ReadInputData

| 函数名称 | GPIO_ReadInputData |
|---|---|
| 函数原型 | u16 GPIO_ReadInputData(GPIO_TypeDef* GPIOx) |
| 功能描述 | 读取指定的 GPIO 端口输入 |
| 输入参数 | GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设 |
| 输出参数 | 无 |
| 返回值 | GPIO 输入数据端口值 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

例：

```
/*Read the GPIOC input data port and store it in ReadValue
variable*/
u16 ReadValue;
ReadValue = GPIO_ReadInputData(GPIOC);
```

### 4.24.9.4 stm32f10x_it.c 文件里的内容

```
    #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);
       }
    }
//以下程序很重要！！
void TIM2_IRQHandler(void)
{
   if(js_count>0) js_count--;
```

```
    if(flag==1)   //如果全局变量标志位等于 1，顺序执行
    {

GPIO_WriteBit(GPIOD,GPIO_Pin_12,(BitAction)((1-GPIO_ReadOutputDataBit(GPIOD,
GPIO_Pin_12))));   //向 PD-12 管脚每隔一定时间写与上次取反的状态值
    }
    else
    {
        GPIO_ResetBits(GPIOD,GPIO_Pin_12);

    }

    TIM_ClearITPendingBit(TIM2,TIM_IT_Update); //清中断，以备下次中断使用
}
```

## 4.24.10 hw_fs.h 文件里的内容是

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

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

    void TIM2_Configuration(void);
    void hw_fs(u32 dt);
    #endif
```

## 4.24.11 hw_fs.c 文件里的内容是

自定义函数 hw_fs.c 的内容如下：

```
    #include "pbdata.h"

    void TIM2_Configuration(void)
    {
       TIM_TimeBaseInitTypeDef TIM_TimeBaseStruct;

       TIM_ClearITPendingBit(TIM2,TIM_IT_Update);
```

```
    TIM_TimeBaseStruct.TIM_Period=947;//初值
    TIM_TimeBaseStruct.TIM_Prescaler=0;//预分频
    TIM_TimeBaseStruct.TIM_ClockDivision=0;
    TIM_TimeBaseStruct.TIM_CounterMode=TIM_CounterMode_Up;//向上

    TIM_TimeBaseInit(TIM2,&TIM_TimeBaseStruct);

    TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE);
    TIM_Cmd(TIM2,ENABLE);
}

void hw_fs(u32 dt)
{
    u8 i=0;

    flag=0;//初始化管脚
    js_count=683;//9ms 起始信号低电平
    flag=1;
    while(js_count); //等待

    js_count=341;//4.5ms 起始信号高电平
    flag=0;
    while(js_count); //等待

    for(i=0;i<32;i++)
    {
        js_count=42;//低电平 常数 0.56ms
        flag=1;
        while(js_count); //等待

        if((dt&0x80000000)==0)
        {
           js_count=43;//0.565ms  发射 “0”
        }
        else
        {
           js_count=128;//1.69ms  发射 “1”
        }

        flag=0;
        while(js_count); //等待
        dt=dt<<1;  //向左移动一位
    }
```

```
    js_count=20; //0.263ms
    flag=1;
    while(js_count); //等待
    flag=0;
}
```

## 4.24.12 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)
{
    u32 dt=0x00FF30CF;
    RCC_Configuration(); //系统时钟初始化
    GPIO_Configuration();//端口初始化
    USART_Configuration();
    NVIC_Configuration();
    TIM2_Configuration();

    while(1)
    {
      //printf("字符串输出 d4=%s\r\n",d4);
      hw_fs(dt);
      delay_ms(1000);
    }
}

void RCC_Configuration(void)
{
```

```
    SystemInit();//72m
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD,ENABLE);
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
   RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,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);

   GPIO_InitStructure.GPIO_Pin=GPIO_Pin_12;//红外发射
   GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;
   GPIO_Init(GPIOD,&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);

   NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
   NVIC_Init(&NVIC_InitStructure);
}
```

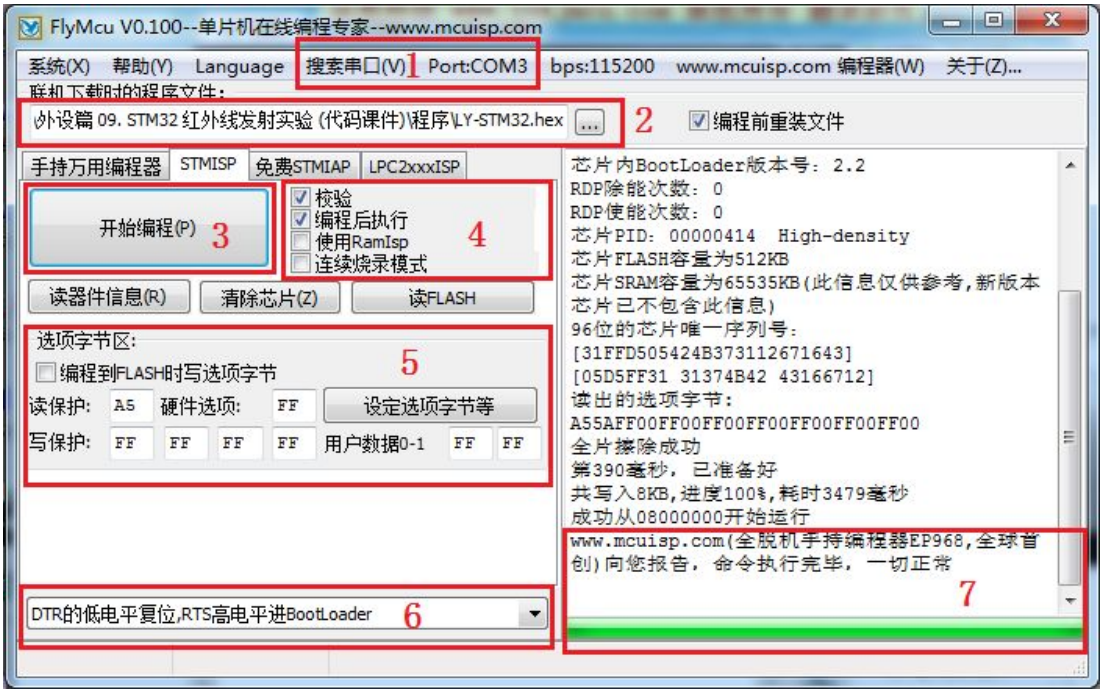
```
    void USART_Configuration(void)
    {
        USART_InitTypeDef  USART_InitStructure;

       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);
    }
```

## 4.24.13 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 09. STM32 红外线发射实验\程序）

## 4.24.14 实验效果图

在这个实验中我们使用两块大黄蜂实验板来实现红外线的发送和接收功能。我们先使其中的一块始终处于红外线编码发送状态，发送的数据码是十六进制“00FF30CF”，另一块处于红外线编码接收状态。使用众想科技多功能监控软件，可以观察到大黄蜂实验板接收到的十六制编码，发送实验板和接收实验板两者的数据是相同的，说明程序设计很成功。具体实验感官请参考下图。

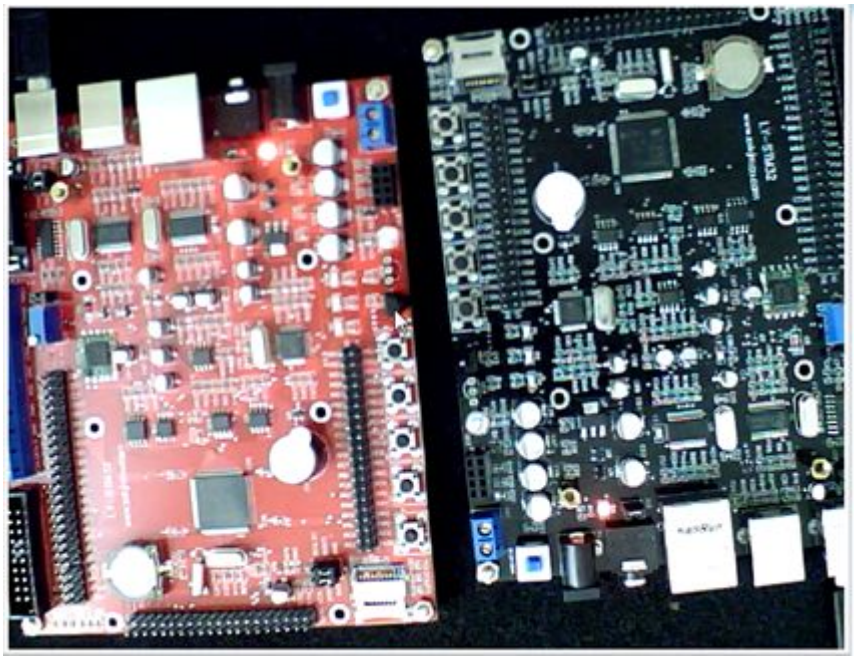


图 4.24.14.1 红外线发送/接收实验效果图一

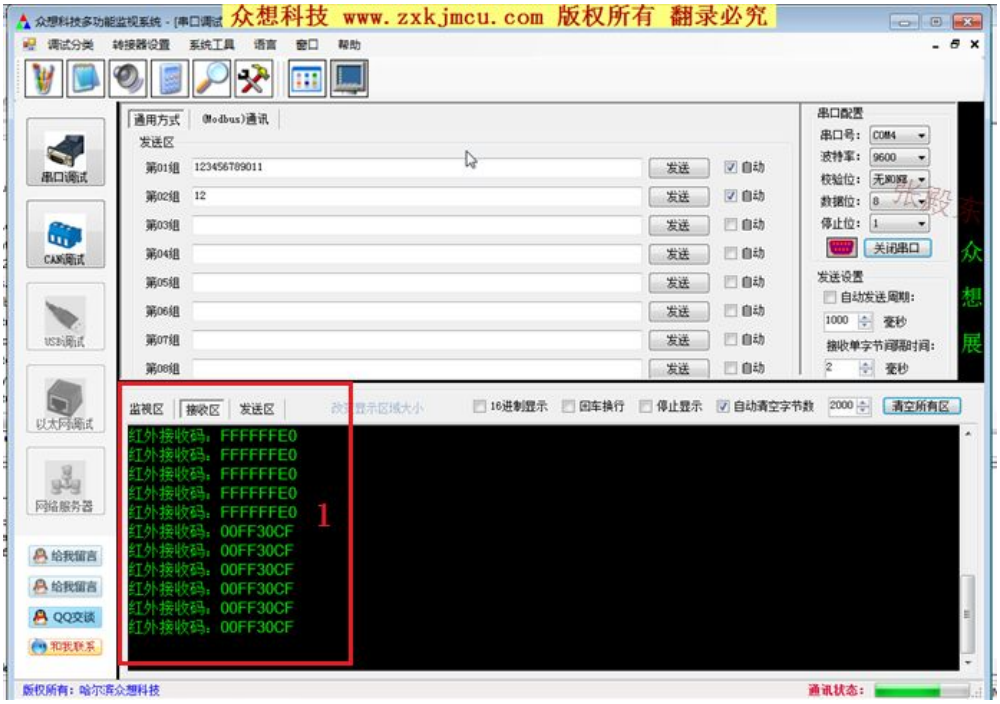



图 4.24.14.2 红外线接收实验效果图二