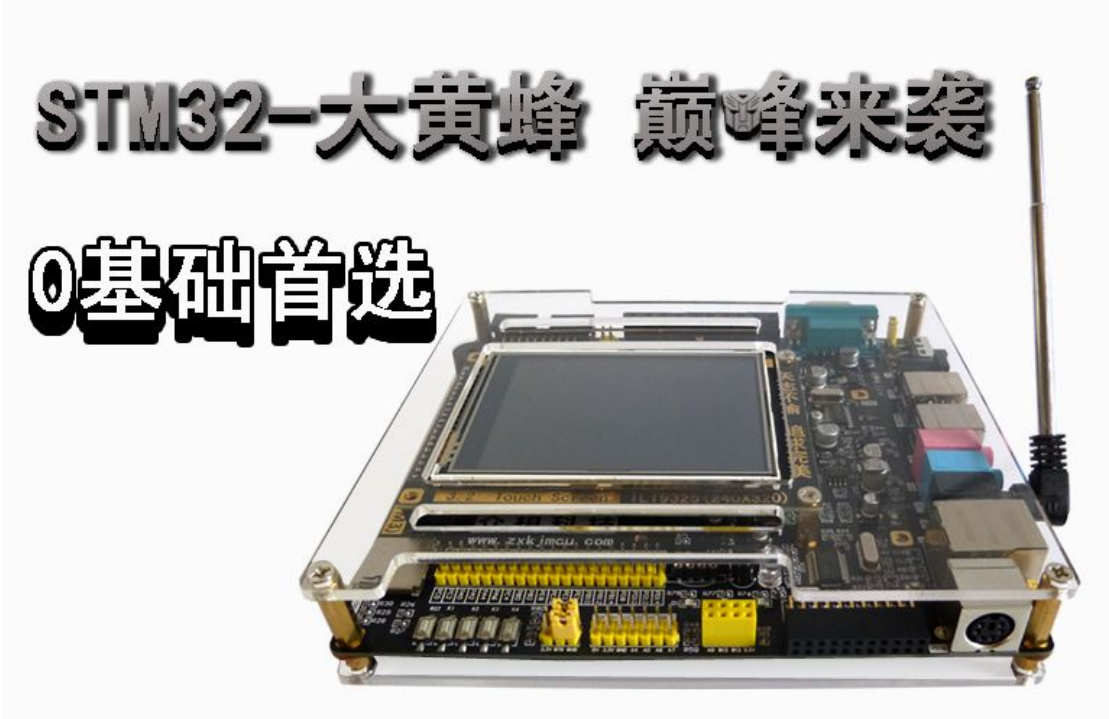
学 ARM 从 STM32 开始

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

4.6 STM32 脉冲宽度调制（PWM）输出实验

4.6.1 概述

4.6.1.1 脉冲宽度调制（PWM）概述

脉冲宽度调制（PWM）是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术，广泛应用在从测量、通信到功率控制与变换的许多领域中。

脉冲宽度调制（PWM）是一种模拟控制方式，其根据相应载荷的变化来调制晶体管基极或 MOS 管栅极的偏置，来实现晶体管或 MOS 管导通时间的改变，从而实现开关稳压电源输出的改变。这种方式能使电源的输出电压在工作条件变化时保持恒定，是利用微处理器的数字信号对模拟电路进行控制的一种非常有效的技术。

4.6.1.2 脉冲宽度调制（PWM）基本原理

控制方式就是对逆变电路开关器件的通断进行控制，使输出端得到一系列幅值相等的脉冲，用这些脉冲来代替正弦波或所需要的波形。也就是在输出波形的半个周期中产生多个脉冲，使各脉冲的等值电压为正弦波形，所获得的输出平滑且低次谐波少。按一定的规则对各脉冲的宽度进行调制，即可改变逆变电路输出电压的大小，也可改变输出频率。

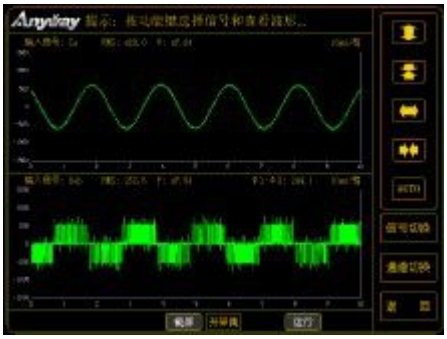
例如，把正弦半波波形分成 N 等份，就可把正弦半波看成由 N 个彼此相连的脉冲所组成的波形。这些脉冲宽度相等，都等于 Π/n ，但幅值不等，且脉冲顶部不是水平直线，而是曲线，各脉冲的幅值按正弦规律变化。如果

把上述脉冲序列用同样数量的等幅而不等宽的矩形脉冲序列代替，使矩形脉冲的中点和相应正弦等分的中点重合，且使矩形脉冲和相应正弦部分面积（即冲量）相等，就得到一组脉冲序列，这就是 PWM 波形。可以看出，各脉冲宽度是按正弦规律变化的。根据冲量相等效果相同的原理，PWM 波形和正弦半波是等效的。对于正弦的负半周，也可以用同样的方法得到 PWM 波形。

在 PWM 波形中，各脉冲的幅值是相等的，要改变等效输出正弦波的幅值时，只要按同一比例系数改变各脉冲的宽度即可，因此在（交-直-交）变频器中，PWM 逆变电路输出的脉冲电压就是直流测电压的幅值。

根据上述原理，在给出了正弦波频率，幅值和半个周期内的脉冲数后，PWM 波形各脉冲的宽度和间隔就可以准确计算出来。按照计算结果控制电路中各开关器件的通断，就可以得到所需要的 PWM 波形。

下图为变频器输出的 PWM 波的实时波形。



PWM 实际波形图

4.6.2 STM32 中用定时器产生脉冲宽度调制（PWM）输出

4.6.2.1 STM32 中的定时器除了 TIM6 和 TIM7，其它的定时器都可以用来产生 PWM 输出。其中高级定时器 TIM1、TIM8 可以同时产生多达 7 路的 PWM 输出。而通用定时器也能同时产生多达 4 路的 PWM 输出，由此看出 STM32 最多可以同时产生 30 路 PWM 输出！！！

4.6.2.2 STM32 中占空比寄存器

STM32 的 PWM 是 TIMx_ARR 寄存器确定频率（周期）、由能 TIMx_CCRx 寄存器确认占空比的信号。在 TIM1_CCMRx 寄存器中的 OCxM 位写入“110”（PWM 模式 1）或“111”（PWM 模式 2），能够独立地设置每个通道工作在 PWM 模式，每个 OCx 输出一路 PWM。必须通过设置 TIM1_CCMRx 寄存器 OCxPE 位使能相应的预装载寄存器，最后还要设置 TIM1_CR1 寄存器的 ARPE 位使能自动重装载的预处理计数器。

4.6.2.3 STM32 PWM 边沿对齐模式

- 向上计数的配置

当 TIM1_CR1 寄存器中的 DIR 位为低的时候执行向上计数。

在 PWM 模式 1，TIM1_CNT<TIM1_CCRx 时 PWM 参考信号，OCxREF 为高，否则为低。如果，TIM1_CCRx 中的比较值大于自动重装载值（TIM1_ARR），则 TIM1_OCxREF 保持为“1”，如果比较值为“0”，则 TIM1_OCxREF 保持为“0”。

- 向下计数的配置

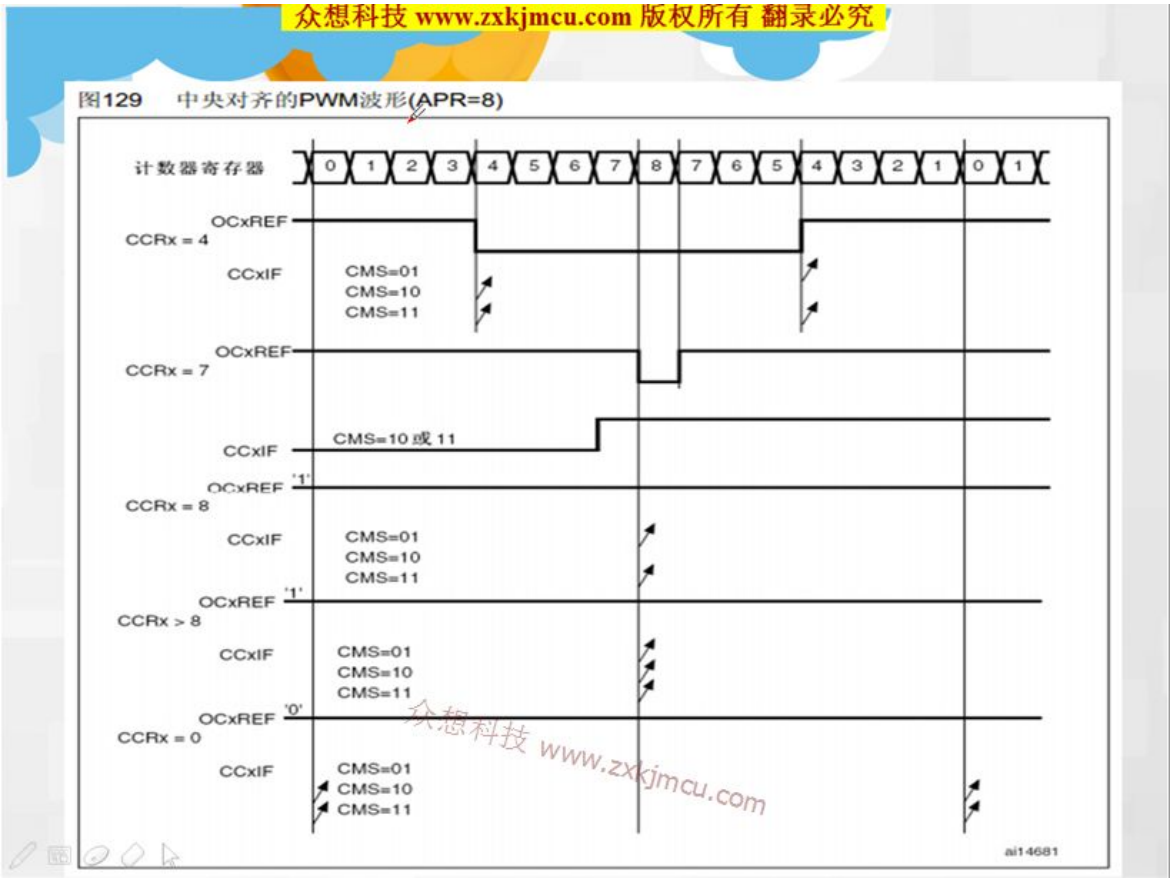
当 TIM1_CR1 寄存器中的 DIR 位为高的时候执行向下计数。

在 PWM 模式 1，TIM1_CNT>TIM1_CCRx 时 PWM 参考信号，OCxREF 为低，否则为高。如果，TIM1_CCRx 中的比较值大于自动重装载值（TIM1_ARR），则 TIM1_OCxREF 保持为“1”，该模式下不产生 0%的 PWM 波形。

4.6.2.4 STM32 PWM 中央对齐模式

当 TIM1_CR1 寄存器中的 CMS 位不为 0 时为中央对其模式（所有其他的配置对 OCxREF/OCx 信号都有相同的作用）。根据不同的 CMS 位的设置，比

较标志可能在计数器向上计数时被置 1、在计数器向下计数时被置 1、或者在计数器向上/向下计数时被置 1.TIM1_CR1 寄存器中的计数方向位（DIR）由硬件更新，不要用软件修改它。如下图所示：



从上图可以清晰的看出，在 PWM 模式 1，TIM1_CNT>TIM1_CCRx 时 PWM 参考信号，OCxREF 为高，否则为低。如果，TIM1_CCRx 中的比较值大于自动重装载值（TIM1_ARR），则 TIM1_OCxREF 保持为“1”，IM1_CCRx 中的比较值为“0”时，则 TIM1_OCxREF 保持为“0”。

4.6.3 实验目的

通过点亮发光二极管，使发光二极管先渐渐变亮，直至全亮，然后由全亮逐渐变暗，如此反复来检验 PWM 输出的模拟量特性。

4.6.4 硬件设计

在前几章的基础上做这个实验，硬件环境还是使用 LED 发光二极管，电路大家都熟悉了（参考原理图纸）。在这里就不再重复说明。

4.6.5 软件设计

在这个小程序中我们用到计数器和定时器。所以还要结合上一节的内容进行扩展编程，初学者逐步熟悉各种情况，由浅入深，由易入难，循序渐进。

我们全部使用库函数编写程序，要注意“清空”的操作，比如进入定时器处理程序时，也要清空定时器标志；熟悉端口复用设置;主要要掌握软件的映射功能。

4.6.5.1 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
```

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。在本次试验中我们使用定时器产生 PWM 波形，所以以上库文件就足够使用了。

4.6.5.2 自定义头文件

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

同时我们自己也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。

4.6.5.3 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"

extern u8 dt;  //定义变量

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。

4.6.5.4 pbdata.c 文件里的内容是

```
#include "pbdata.h" //很重要，引用这个头文件

/*******************************************************************************
* 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
  }
```

4.6.6 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。

4.6.7 GPIO 引脚时钟使能

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);//设置定时器 3 时钟
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//功能复用 IO 时钟函数
```

本节实验用到了 PB 端口,所以要把 PB 端口的时钟打开;定时器 3(TIM3)时钟源是通过 APB1 预分频器得到的，时器 3 时钟初始化；因为要产生 PWM 波形，所以要打开功能复用时钟。

4.6.8 GPIO 管脚电平控制函数

在主程序中采用 while(1)循环语句，主程序每间隔 10ms 改变 PWM 波形的占空比，通过 PB5 端口输出，LED1 会随着 PWM 波形逐渐改变亮度，逐渐变亮后逐渐再变暗，周而复始。

```
while(1)
  {
      delay_ms(10);//延时 10ms
   if(led_fx==1)//如果中间变量 led_fx==1，就做加法运算，每次加 1
   {
      led_dt++;
   }
   else       //如果中间变量 led_fx==0，就做减法运算，每次减 1
   {
     led_dt--;
   }

   if(led_dt>300)   led_fx=0;//对加法或减法运算的上下限限制
   if(led_dt==0)    led_fx=1;
```

```
    TIM_SetCompare2(TIM3,led_dt);//设置TIMx捕获比较2寄存器,使PWM按照程序控制产生符合要求的波形
  }
```

4.6.9 main.c 文件里的内容是

中断是指 CPU 对系统发生的某个事件作出的一种反应：CPU 暂停正在执行的程序，保留现场后自动转去执行相应的处理程序，处理完该事件后再返回断点继续执行被"打断"的程序，在这个试验中主程序大部分是初始化函数和几个子函数，主要执行过程就使等待中断的产生。下面是主程序的内容。

```
#include "pbdata.h"

void RCC_Configuration(void);
void GPIO_Configuration(void);
void TIM3_Configuration(void);

int main(void)
{
   u8 led_fx=1;//临时中间变量led_fx（字节），并付初值
   u16 led_dt=0;//临时中间变量led_dt（字），并付初值

   RCC_Configuration();//系统时钟初始化
   GPIO_Configuration();//端口初始化
   TIM3_Configuration();//定时器和pwm配置
   while(1)
   {
    delay_ms(10);
    if(led_fx==1)
    {
      led_dt++;
    }
    else
    {
     led_dt--;
    }

    if(led_dt>300)  led_fx=0;
    if(led_dt==0)    led_fx=1;

    TIM_SetCompare2(TIM3,led_dt);
   }
```

```
    }

    void RCC_Configuration(void)
    {
        SystemInit();
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);//TIM3 定时器通过
APB1 预分频器得到基准时钟
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//TIM3 定时器通过
APB2 预分频器打开功能复用 IO 时钟
    }

    void GPIO_Configuration(void)
    {
        GPIO_InitTypeDef GPIO_InitStructure;
        //LED
        GPIO_InitStructure.GPIO_Pin=GPIO_Pin_5;
        GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
        GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;//端口复用推挽输出功能
        GPIO_Init(GPIOB,&GPIO_InitStructure);//
    }

    void TIM3_Configuration(void)
    {
        TIM_TimeBaseInitTypeDef TIM_TimeBaseStruct;//TIM3 定时器初始化子函数
        TIM_OCInitTypeDef TIM_OCInitStructure;//PWM 初始化子函数

        GPIO_PinRemapConfig(GPIO_PartialRemap_TIM3,ENABLE);//TIM3 复用功能部分
映射使能

        //定时器初始化
        TIM_TimeBaseStruct.TIM_Period=899;//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(TIM3,&TIM_TimeBaseStruct);


        //pwm 初始化
        TIM_OCInitStructure.TIM_OCMode=TIM_OCMode_PWM1;
        TIM_OCInitStructure.TIM_OutputState=TIM_OutputState_Enable;
        TIM_OCInitStructure.TIM_OCPolarity=TIM_OCPolarity_Low;
```

```
        TIM_OC2Init(TIM3,&TIM_OCInitStructure);//初始化参数传递

        TIM_OC2PreloadConfig(TIM3,TIM_OCPreload_Enable);//使能或者失能 TIMx 在
CCR2 上的预装载寄存器
        TIM_Cmd(TIM3,ENABLE);//打开定时器外设
    }
```

在 main(void)程序体中代码比较多，有几个函数理解起来有些困难，下面对函数 GPIO_PinRemapConfig 和函数 TIM_OC2PreloadConfig 做一下比较详细的说明。

1、函数 GPIO_PinRemapConfig

表 1 GPIO_PinRemapConfig 说明

函数名	GPIO_PinRemapConfig
函数原型	Void GPIO_PinRemapConfig(u32 GPIO_Remap,FunctionalState NewState)
功能描述	改变指定管脚的映射
输入参数 1	GPIO_Remap：选择映射的管脚 参阅 Section: GPIO_Remap 查阅更多该参数允许取值范围
输入参数 2	NewState：管脚重映射的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

GPIO_Remap

GPIO_Remap 用以选择用作事件输出的 GPIO 端口，下表给出了该参数可能的取值

附表 1 GPIO_Remap 值

GPIO_Remap	描述
GPIO_Remap_SPI1	SPI1 复用功能映射
GPIO_Remap_I2C1	I2C1 复用功能映射
GPIO_Remap_USART1	USART1 复用功能映射
GPIO_PartiaRemap_USART3	USART2 复用功能映射
GPIO_FullRemap_USART3	USART3 复用功能完全映射
GPIO_PartiaRemap_TIM1	USART3 复用功能部分映射
GPIO_FullRemap_TIM1	TIM1 复用功能完全映射
GPIO_PartiaRemap1_TIM2	TIM2 复用功能部分映射 1
GPIO_PartiaRemap2_TIM2	TIM2 复用功能部分映射 2
GPIO_FullRemap_TIM2	TIM2 复用功能完全映射

GPIO_PartiaRemap_TIM3	TIM3 复用功能部分映射
GPIO_FullRemap_TIM3	TIM3 复用功能完全映射
GPIO_Remap_TIM4	TIM4 复用功能映射
GPIO_Remap1_CAN	CAN 复用功能映射 1
GPIO_Remap2_CAN	CAN 复用功能映射 2
GPIO_Remap_PD01	PD01 复用功能映射
GPIO_Remap_SWJ_NoJTRST	除 JTRST 外 SWJ 完全使能（JTAG+SW_DP）
GPIO_Remap_SWJ_JTAGDisable	JTAG_DP 失能+SW_DP 使能
GPIO_Remap_SWJ_Disable	SWJ 完全失能（JTAG+SW_DP）

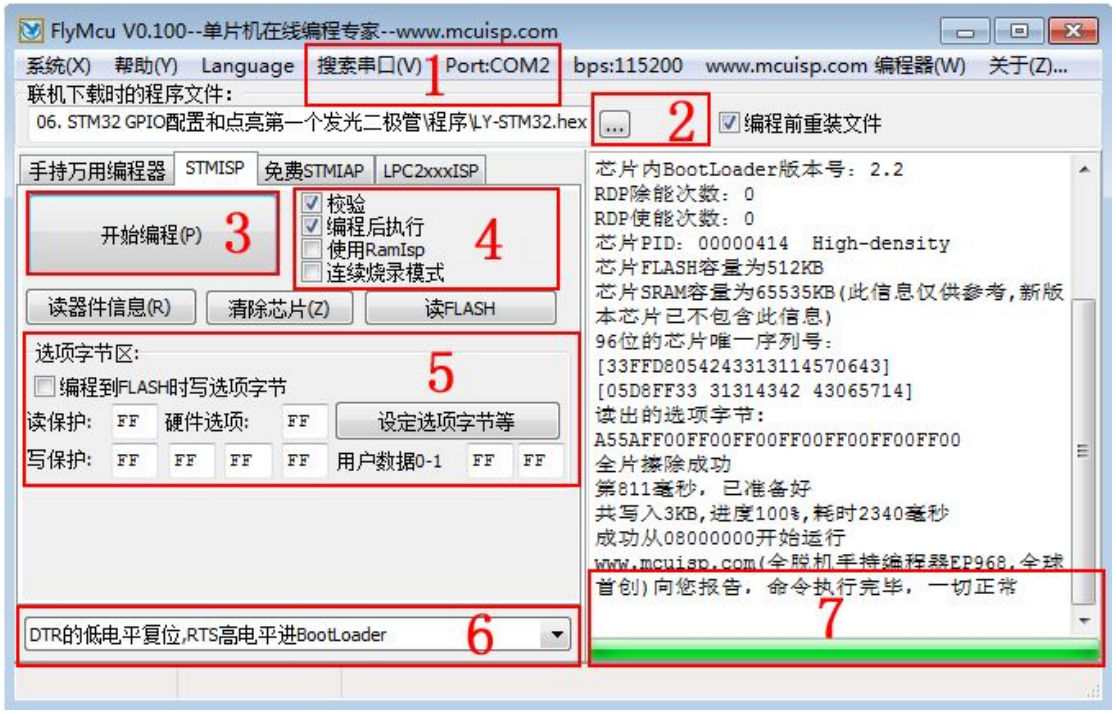
2、函数 TIM_OC2PreloadConfig

表 2 TIM_OC2PreloadConfig 说明

函数名	TIM_OC2PreloadConfig
函数原型	Void TIM_OC2PreloadConfig(TIM_TypeDef*TIMx,u16 TIMx_OCPreload)
功能描述	使能或者失能 TIMx 在 CCR2 上的预装载寄存器
输入参数 1	TIMx：x 可以是 2、3、或者 4，来选择 TIM 外设
输入参数 2	TIMx_OCPreload：输出比较预装载状态 参阅 Section：TIMx_OCPreload 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

4.6.10 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 COM2。（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\1,基础篇\基础篇 13. STM32 PWM 输出实验\程序）

4.6.11 实验效果图

外部中断程序写入实验板后，可以看出 LED 发光二极管有规律的逐渐变亮和逐渐变暗，交替渐变。PWM 波形输出的处理是一个比较难理解的东西，他的设置比价多，还有一些小的细节需要特别注意，需要花费多一些的时间去理解和编程实验。只要学习者认真看书和看视频，再加上勤奋，一定会很快掌握。