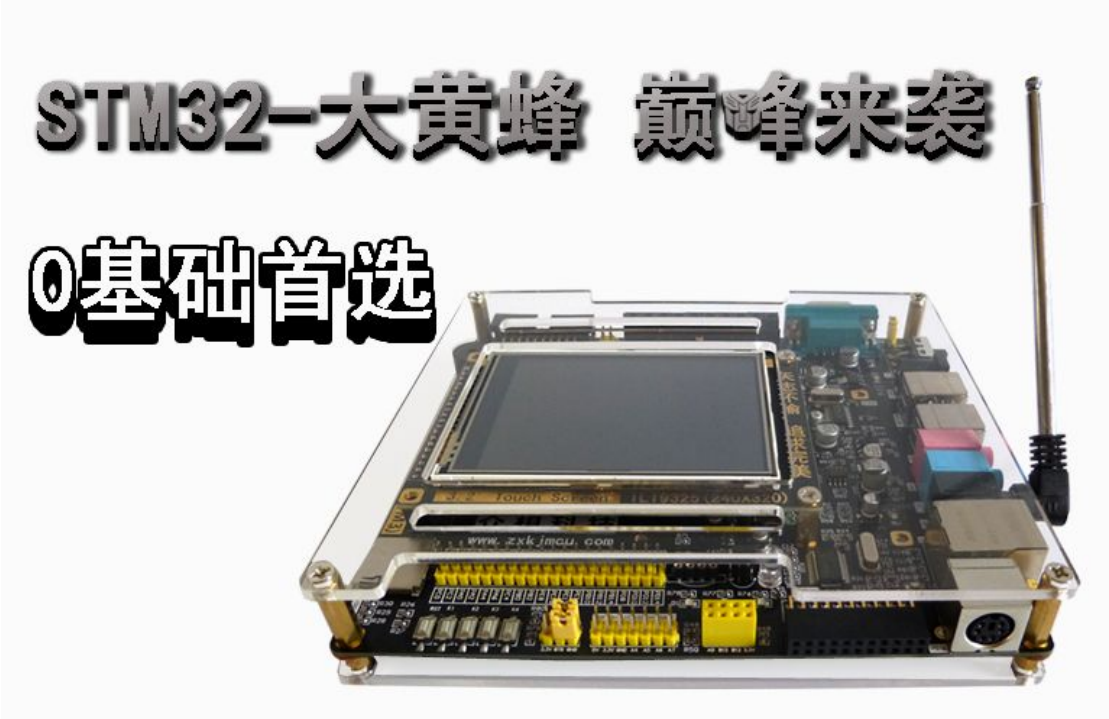
学 ARM 从 STM32 开始

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

4.5 STM32 定时器中断操作实验

4.5.1 概述

4.5.1.1 定时器概述

人类最早使用的定时工具是沙漏或水漏，但在钟表诞生发展成熟之后，人们开始尝试使用这种全新的计时工具来改进定时器，达到准确控制时间的目的。

1876 年，英国外科医生索加取得一项定时装置的专利，用来控制煤气街灯的开关。它利用机械钟带动开关来控制煤气阀门。起初每周上一次发条，1918 年使用电钟计时后，就不用上发条了。

定时器确实是一项了不起的发明，使相当多需要人控制时间的工作变得简单了许多。人们甚至将定时器用在了军事方面，制成了定时炸弹，定时雷管。现在的不少家用电器都安装了定时器来控制开关或工作时间。

4.5.1.2 定时器种类

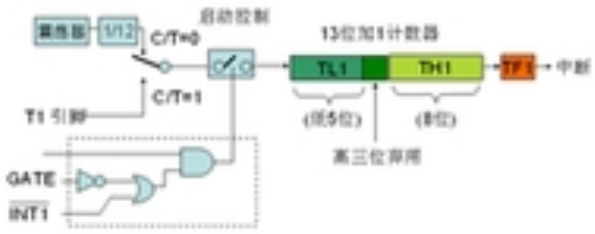
1、 接通延时型定时器

2、 断开延时型定时器

3、保持型接通延时定时器：

4、脉冲型定时器：

5、扩张型脉冲定时器：



4.5.2 STM32 中定时器说明

4.5.2.1 STM32 中有 11 个定时器，其中两个高级控制定时器，4 个通用定时器和 2 个基本定时器，以及 2 个看门狗定时器和 1 个系统滴答定时器。其中系统滴答定时器就是前文所描述的 SysTick。

定时器	计数器分辨率	计数器类型	预分频系数	产生DMA请求	捕获/比较通道	互补输出
TIM1 TIM8	16位	向上，向下，向上/向下	1-65536之间的任意数	可以	4	有
TIM2 TIM3 TIM4 TIM5	16位	向上，向下，向上/向下	1-65536之间的任意数	可以	4	没有
TIM6 TIM7	16位	向上	1-65536之间的任意数	可以	0	没有

表格一 定时器详细说明

其中 TIM2~TIM5 是通用定时器，TIM6 和 TIM7 是基本定时器。其时钟由 APB1 产生。

4.5.2.2 TIMx 主要功能

通用 TIMx（TIM2、TIM3、TIM4、TIM5）定时器主要功能包括：

- 16 位向上、向下、向上/向下自动装载定时器
- 16 位可编程（可实时修改）预分频器 ，计数器时钟频率的分频系数为 1~65536 任意数值。
- 4 个通道：输入捕获、输出比较、PWM 生成（边缘或中间对齐模式）、单脉冲模式输出。

- 使用外部信号控制定时器和定时器互连的同步电路。
- 如下事件发生时产生中断/DMA

更新：计数器向上/向下溢出，计数器初始化（通过软件或者内部/外部触发）

触发事件（计数器启动、停止、初始化或者由内部/外部触发计数）

输入捕获

输出比较

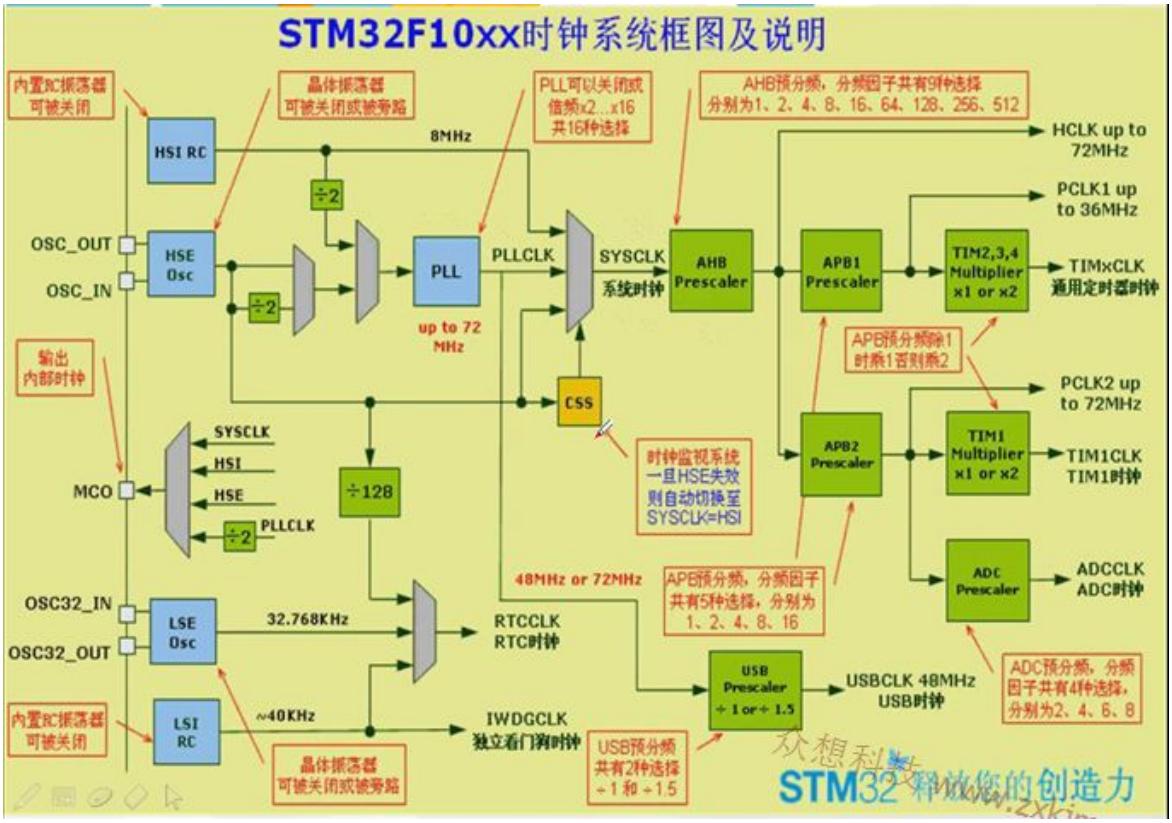
- 支持针对定位的增量（正交）编码器和霍尔传感器电路。
- 触发输入作为外部时钟或者安周期的电流管理。

4.5.2.3 时钟来源

计数器时钟可以由下列时钟源提供：

- 内部时钟（CK_INT）；
- 外部时钟模式 1（外部输入脚 TIx）；
- 外部时钟模式 2（外部触发输入 ETR）；
- 内部触发输入（ITRx），使用一个定时器作为另一个定时器的预分频器，如可以配置一个定时器 Timer1 作为另一个定时器 Timer2 的预分频器。

系统时钟拓补图见图二，我们在第 5 节已经详细介绍了这个图，在这里还要使用这个图。从图中可以看出系统时钟（sysclk）是 72M，AHB 预分频器的分频因子有 9 种选择。在这个实验中分频因子选择 1，所以系统总线上的最高频率是 72M。

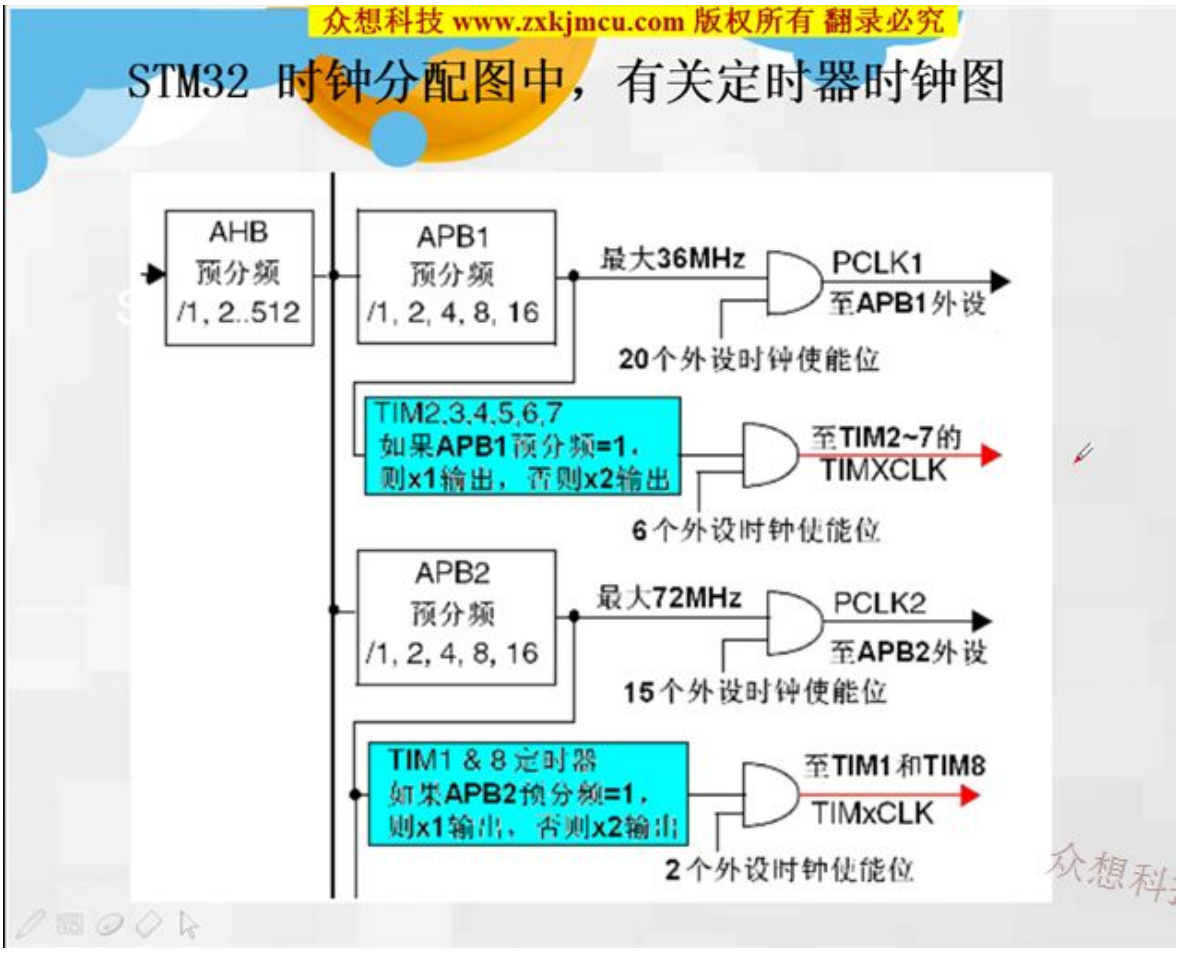


图二 STM32 系统时钟框图

图三是更详细的说明 AHB 预分频器后的总线频率。在 AHB 之后是怎样分频的，最高频率能达到多大，都有很详细的说明。

- APB1 预分频器，分频因数按照/1、2、4、8、16，能得到五种分频方式，但是 APB1 预分频器后总线设计的最高频率只能是 36M；所以 APB1 预分频器在输入频率为 72M 时只能采用 2、4、8、16 四种分频方式。从图上可与看出 TIM2、3、4、5、6、7 输入端的最高频率是 36M（最高频率由芯片设计决定）。根据 TIM2、3、4、5、6、7 输出端频率的设计原则（蓝色框文字说明），当输入端的频率是 36M 时，6 个外设时钟输出端频率是 72M。
- APB2 预分频器，分频因数按照/1、2、4、8、16，能得到五种分频方式，APB2 预分频器后总线设计的最高频率等达到 72M；所以 APB2 预分频器只

能采用 1、2、4、8、16 五种分频方式。从图上可以看出 TIM1、8 输入端的最高频率是 72M。根据 TIM1、8 输出端频率的设计原则（蓝色框文字说明），当输入端的频率是 72M 时，2 个外设时钟输出端频率是 72M。



图三 STM32 定时器时钟图

4.5.2.3 计数器模式

TIM2-TIM5 可以由向上计数、向下计数、向上向下双向计数。向上计数模式中，计数器从 0 计数到自动加载值（TIMx_ARR 计数器内容），然后重新从 0 开始计数并产生一个计数器溢出事件。在向下计数模式中，计数器从自动装入的值（TIMx_ARR）开始向下计数到 0，然后从自动装入的值重新开始，并产生一个计数器向下溢出事件。而中央对齐模式（向上/向下计数）是计数器从 0 开始计数到自动装入值-1，产生一个计数器溢出事件，然后向下计数

到 1 并产生一个计数器溢出事件；然后再从 0 开始重新计数。

4.5.3 实验目的

通过定时器 TIM3 产生间隔 1 秒一次的中断，在中断中控制 LED 发光二极管，每次中断都使发光二极管状态取反。

4.5.4 硬件设计

在前几章的基础上做这个实验，硬件环境还是使用键盘和 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了。

4.5.5 软件设计

在这个小程序中我们要熟悉 TIM3_Configuration()这个函数。

我们全部使用库函数编写程序，在这里要注意“清空”的操作，比如进入中断处理程序后，要先清空中断标志；进入定时器处理程序时，也要清空定时器标志。

4.5.5.1 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处理程序，中断处理程序各个子程序框架设计好了，但是具体的中断处理程序要用户自己编写来实现中断处理功能。tm32f10x_tim.c 库函数主要包含定

时器设置。

4.5.5.2 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。

4.5.5.3 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"

#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。 stm32f10x.h 头文件是我们每个工程都需要调用的，里面包括了 STM32 内部寄存器地址的定义；misc.h 里边包含了中断优先级变量表；stm32f10x_exti.h 头文件里包含了外部中断设置参数；stm32f10x_tim.h 头文件里包含了定时器设置参数。

4.5.5.4 pbdata.c 文件里的内容是

```
#include "pbdata.h" //很重要，引用这个头文件
```

4.5.6 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。

4.5.7 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
```

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。

4.5.8 GPIO 管脚电平控制函数

在主程序中采用 while(1)循环语句，程序就是等待定时器 3 (TIM3)产生中断，不来我就一直等待，如果有中断就跳转到中断处理程序，如果监测到中断产生，就使 LED 输出端口状态取反，以此类推，周而复始。

```
    while(1);
```

4.5.9 STM32 中断处理函数 stm32f10x_it.c

在本次试验中，中断处理函数是非常重要的函数，大家一定要掌握好，现在我们再详细介绍一下这个函数。在中断处理函数入口要引入下列文件。

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

下面是定时器中断 TIM3 处理程序。

```
    /*******************************************************************************
    * 名    称：void TIM3_IRQHandler(void)
    * 功    能：TIM3 中断处理程序
    * 入口参数：无
    * 出口参数：无
    * 说    明：
    * 调用方法：无
    *******************************************************************************/
    void TIM3_IRQHandler(void)
    {
       TIM_ClearITPendingBit(TIM3,TIM_IT_Update);//清空定时器标志

          if(GPIO_ReadOutputDataBit(GPIOB,GPIO_Pin_5)==Bit_RESET)//把 PC5 端口取反再送回 PC5 端口
          {
             GPIO_SetBits(GPIOB,GPIO_Pin_5);//LED 熄灭

          }
```

```
        else
        {
           GPIO_ResetBits(GPIOB,GPIO_Pin_5);//LED 发光
        }
    }
```

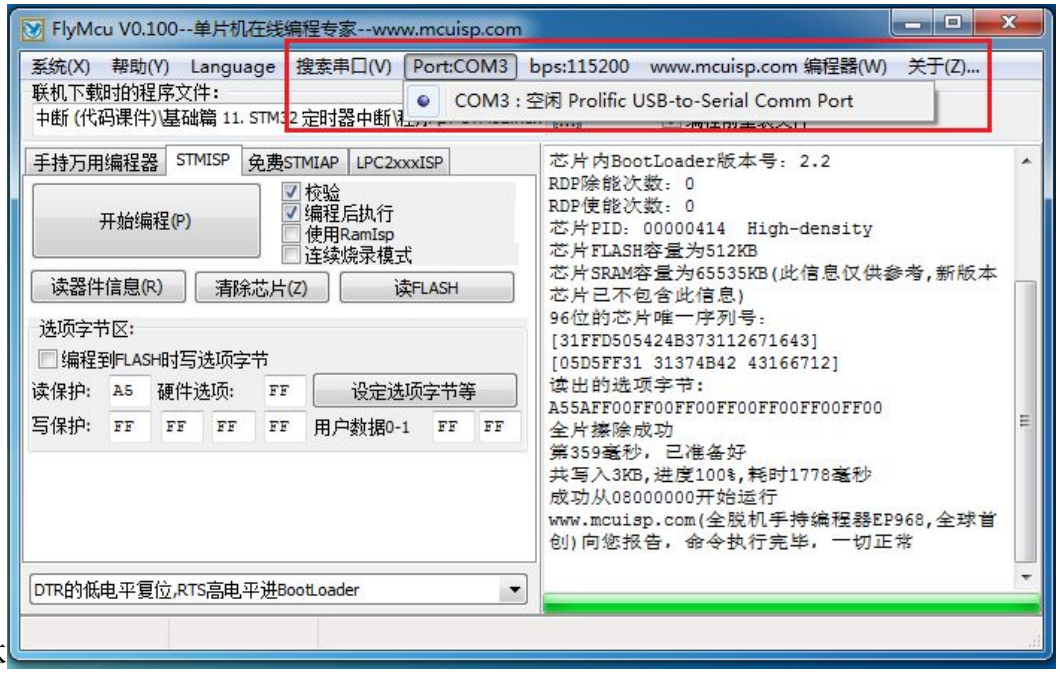
4.5.9 main.c 文件里的内容是

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"  //很重要，引用这个头文件

void RCC_Configuration(void);//声明
void GPIO_Configuration(void);
void NVIC_Configuration(void);
void TIM3_Configuration(void);

int main(void)
{
   RCC_Configuration();  //系统时钟初始化
   GPIO_Configuration();//端口初始化
   TIM3_Configuration();//定时器初始化
   NVIC_Configuration();//中断优先级初始化

   while(1);
}

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
}

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_Out_PP;
    GPIO_Init(GPIOB,&GPIO_InitStructure);
}

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

    TIM_ClearITPendingBit(TIM3,TIM_IT_Update);//清空定时器溢出标志

    TIM_TimeBaseStruct.TIM_Period=2000;//初值
    TIM_TimeBaseStruct.TIM_Prescaler=35999;//预分频
    TIM_TimeBaseStruct.TIM_ClockDivision=0;
    TIM_TimeBaseStruct.TIM_CounterMode=TIM_CounterMode_Up;//向上

    TIM_TimeBaseInit(TIM3,&TIM_TimeBaseStruct);

    TIM_ITConfig(TIM3,TIM_IT_Update,ENABLE);
    TIM_Cmd(TIM3,ENABLE);    //打开定时器外设
}

void NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

    NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

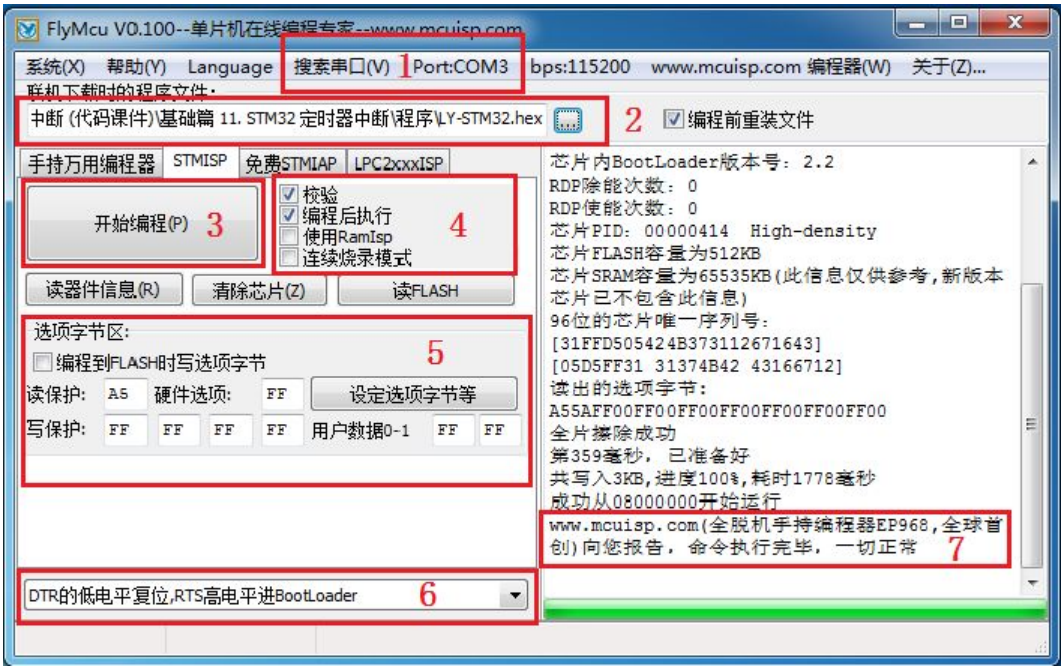
在 main(void)程序体中代码不多，开始调用了四个函数，分别是系统时钟初始化函数 RCC_Configuration()、端口初始化函数 GPIO_Configuration()、中断优先级设置函数 NVIC_Configuration()、定时器（TIM3）设置函数 TIM3_Configuration()，接下来进入 while(1)循环



体，在循环体中，就是等待定时器（TIM3）产生中断的。如果中断产生，就转入定时器中断处理函数，处理中断并相应的输出，使 LED 发光二极管随着每一次定时器计数时间到产生的中断点亮和熄灭。

4.5.10 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 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\1,基础篇\基础篇 11. STM32 定时器中断\程序）

4.5.11 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发光和熄灭，间隔时间为 1 秒钟。定时器中断的处理是一个比较难理解的东西，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。