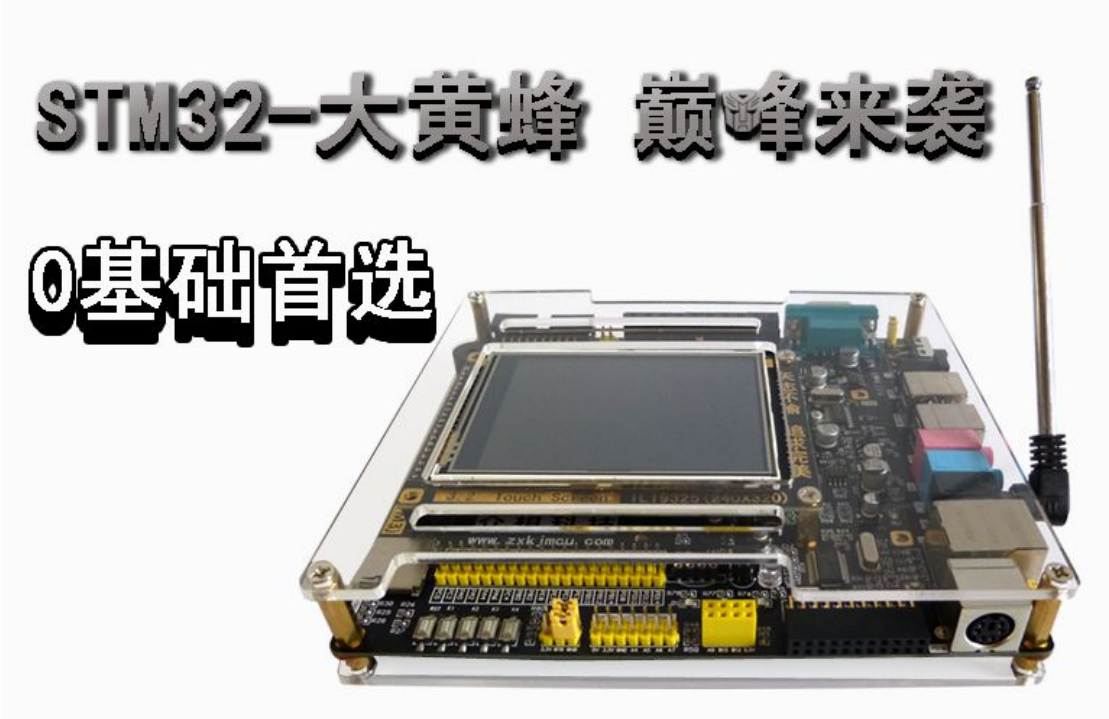
学 ARM 从 STM32 开始

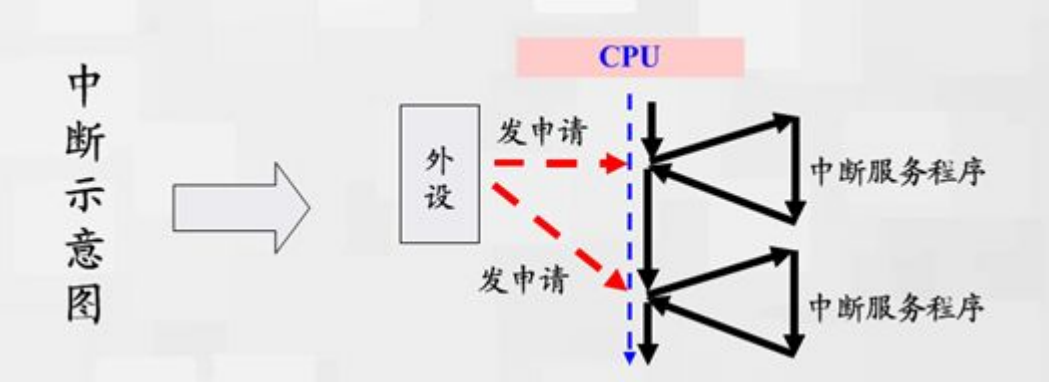
STM32 开发板库函数教程--实战篇



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

4.4 STM32 中断和外部中断操作实验

中断装置和中断处理程序统称为中断系统。中断系统是计算机的重要组成部分。实时控制、故障自动处理、计算机与外围设备间的数据传送往往采用中断系统。中断系统的应用大大提高了计算机效率。



4.4.1.1 中断简介

不同的计算机其硬件结构和软件指令是不完全相同的。因此，中断系统也是不相同的。计算机的中断系统能够加强 CPU 对多任务事件的处理能力。中断机制是现代计算机系统中的基础设施之一，它在系统中起着通信网络作用，以协调系统对各种外部事件的响应和处理。中断是实现多通道程序设计的必要条件。 中断是 CPU 对系统发生的某个事件作出的一种反应。 引起中断的事件称为中断源。中断源向 CPU 提出处理的请求称为中断请求。发生中断时被打断程序的暂停点称为断点。CPU 暂停现行程序而转为响应中断请求的过程称为中断响应。处理中断源的程序称为中断处理程序。CPU 执行有关的中断处理程序称为中断处理。而返回断点的过程称为中断返回。中断的实现实行软件和硬件综合完成，硬件部分叫做硬件装置，软件部分称为软件处理程序。

4.4.1.2 响应处理

大多数中断系统都具有如下几方面的操作，这些操作是按照中断的执行先后次序排列的。①接收中断请求。②查看本级中断屏蔽位，若该位为 1 则本级中断源参加优先权排队。③中断优先权选择。④处理机执行完一条指令后或者这条指令已无法执行完，则立即中止现行程序。接着，中断部件根据中断级去指定相应的主存单元，并把被中断的指令地址和处理机当前的主要状态信息存放在此单元中。⑤中断部件根据中断级又指定另外的主存单元，从这些单元中取出处理机新的状态信息和该级中断控制程序的起始地址。⑥执行中断控制程序和相应的中断服务程序。⑦执行完中断服务程序后，利用专用指令使处理机返回被中断的程序或转向其他程序。

4.4.1.3 相关概念

程序状态字和向量中断

这是两个与中断响应和处理有密切关系的概念。

① 程序状态字：每个程序均有自己的程序状态字。现行程序的程序状态字放在处理机的程序状态字寄存器中。程序状态字中最主要的内容有指令地址、条件码、地址保护键，中断屏蔽和中断响应时的中断源记录等。中断响应和处理操作的第④步和第⑤步就是交换程序状态字操作。

② 向量中断:对应每一级中断都有一个向量，这些向量顺序存放在主存的指定单元中。向量的内容是：相应的中断服务程序起始地址和处理机状态字（主要是指令地址）。在中断响应时，由中断部件提供中断向量的地址，就可取出该向量。中断响应和处理操作的第⑤步就是取中断向量操作。在采用

向量中断的机器中一般不再使用程序状态字。

4.4.1.4 系统功能

1）实现中断响应和中断返回

当 CPU 收到中断请求后，能根据具体情况决定是否响应中断，如果 CPU 没有更急、更重要的工作，则在执行完当前指令后响应这一中断请求。CPU 中断响应过程如下：首先，将断点处的 PC 值（即下一条应执行指令的地址）推入堆栈保留下来，这称为保护断点，由硬件自动执行。然后，将有关的寄存器内容和标志位状态推入堆栈保留下来，这称为保护现场，由用户自己编程完成。保护断点和现场后即可执行中断服务程序，执行完毕，CPU 由中断服务程序返回主程序，中断返回过程如下：首先恢复原保留寄存器的内容和标志位的状态，这称为恢复现场，由用户编程完成。然后，再加返回指令 RETI，RETI 指令的功能是恢复 PC 值，使 CPU 返回断点，这称为恢复断点。恢复现场和断点后，CPU 将继续执行原主程序，中断响应过程到此为止。

2）实现优先权排队

通常，系统中有多个中断源，当有多个中断源同时发出中断请求时，要求计算机能确定哪个中断更紧迫，以便首先响应。为此，计算机给每个中断源规定了优先级别，称为优先权。这样，当多个中断源同时发出中断请求时，优先权高的中断能先被响应，只有优先权高的中断处理结束后才能响应优先权低的中断。计算机按中断源优先权高低逐次响应的过程称优先权排队，这个过程可通过硬件电路来实现，亦可通过软件查询来实现。

3）实现中断嵌套

当 CPU 响应某一中断时，若有优先权高的中断源发出中断请求，则 CPU 能中

断正在进行的中断服务程序，并保留这个程序的断点（类似于子程序嵌套），响应高级中断，高级中断处理结束以后，再继续进行被中断的中断服务程序，这个过程称为中断嵌套。如果发出新的中断请求的中断源的优先权级别与正在处理的中断源同级或更低时，CPU 不会响应这个中断请求，直至正在处理的中断服务程序执行完以后才能去处理新的中断请求。

4.4.1.5 源分类

中断源是指能够引起中断的原因。

一台处理机可能有很多中断源，但按其性质和处理方法，大致可分为如下五类。

① 机器故障中断。

② 程序性中断。现行程序本身的异常事件引起的，可分为以下三种：一是程序性错误，例如指令或操作数的地址边界错，非法操作码和除数为零等；二是产生特殊的运算结果，例如定点溢出；三是程序出现某些预先确定要跟踪的事件，跟踪操作主要用于程序调试。有些机器把程序性中断称为“异常”，不称为中断。

③ 输入—输出设备中断。

④ 外中断。来自控制台中断开关、计时器、时钟或其他设备，这类中断的处理较简单，实时性强。

⑤ 调用管理程序。用户程序利用专用指令“调用管理程序”发中断请求，是用户程序和操作系统之间的联系桥梁。

4.4.1.6 优先权

几个中断请求可能同时出现，但中断系统只能按一定的次序来响应和处

理。可最先被响应的中断具有最高优先权，按优先级别顺序进行处理。优先权高低是由中断部件的中断排队线路确定的。

(1)中断级

当机器设置很多中断源时，为了简化设计，对中断源分组管理。具有相同中断优先权的中断源构成一个中断级。同一级中断使用同一个中断控制程序起点。

(2)中断屏蔽

对应于各中断级设置相应的屏蔽位。只有屏蔽位为 1 时,该中断级才能参加中断优先权排队。中断屏蔽位可由专用指令建立，因而可以灵活地调整中断优先权。有些机器针对某些中断源也设置屏蔽位，只有屏蔽位为 1 时，相应的中断源才起作用。

4.4.2 STM32 中断优先级概念

4.4.2.1 STM32（coetex-M3）有两个中断优先级概念，抢占式优先级和响应式优先级，也把响应式优先级称作“亚优先级”或“副优先级”，每个中断源都需要被指定属于哪一种优先级。

1、何为抢占式优先级（pre-emption priority）

高抢占式优先级的中断事件会打断当前的主程序/中断程序运行，俗称中断嵌套。

2、何为响应式优先级

在抢占式优先级相同的情况下，高相应的优先级首先被相应。

在抢占式优先级相同的情况下，如果有低级抢占式优先级的中断正在被

执行，要等到低级优先级的中断执行结束后才能得到相应（不能被嵌套）。

3、真阳判断中断被响应

首先是抢占式优先级，其次是响应式优先级。

抢占式优先级决定是否有中断嵌套。

4、STM32（coetex-M3）优先级冲突的处理

具有高抢占式优先级的中断可以在具有低抢占式优先级的中断处理中被响应，即中断嵌套，或者说高抢占式优先级的中断可以嵌套低抢占式优先级的中断。当两个中断源的抢占式优先级相同的情况下，这两个中断没有嵌套关系，当以个中断到来时，程序正在处理另一个中断，这个后来到中断要等到前一个中断处理结束后才能被处理。如果这两个中断同时到达，则中断控制器要根据要根据他们的优先级高级来决定先处理那个。如果他们的抢占式优先级和响应优先级都相同，则根据他们在中断表中的排列顺序来决定先处理那个中断。

5、STM32 对中断优先级的定义

下图是 STM32 中断优先级的拓扑说明，

STM32 中指定中断优先级的寄存器有 4 位，NVIC_PriorityGroup 函数，由这个函数的取值（0-4）决定决定寄存器分组状态。设置优先级分组，这 4 个寄存器位分组如下：

第 0 组，所有 4 位用于指定响应优先级；

第 1 组，最高 1 位用于指定抢占式优先级，余下 3 位用于指定响应优先级；

第 2 组，最高 2 位用于指定抢占式优先级，余下 2 位用于指定响应优先级；

第 3 组，最高 3 位用于指定抢占式优先级，余下 1 位用于指定响应优先级；

第 4 组，所有 4 位用于指定抢占式优先级；

下面说一下本次试验的目的。

4.4.3 实验目的

通过实验板键盘产生外部中断，在中断中控制 LED 发光二极管，当键盘每次按下时，LED 发光二极管状态取反。目的是通过键盘产生中断，中断处理程序控制 LED 发光二极管，当键盘按下时 LED 发光，当键盘再一次按下时 LED 熄灭，如此反复。

4.4.4 硬件设计

大黄蜂 STM32 开发板上有 4 个独立键盘和 3 个发光二极管，从左至右编号分别是 K1、K2、K3、K4、LED1、LED2、LED3。PCB 线路板上的标号分别是 K1、K2、K3、K4、LED1、LED2、LED3。与原理图对应的标号是(34.PC5-CO1、16.PC1-COL2、17.PC2-COL3、18.PC3-COL4)。原理图上标号的命名规则上节已经详细说明，这里就不再详细说明。

原理图与 PCB 线路板上已经都连接好了，大家可以参考电路图纸，不用自己搭接环境了，直接把程序下载到大黄蜂 STM32 开发板上运行就可以了。

4.4.4 软件设计

- **中断总结概述**

ARM coetex-M3 内核共支持 256 个中断，其中 16 个内部中断，240 个外

部中断和可编程的 256 级中断优先级设置。STM32 目前共支持中断 84 个（16 个内部+68 个外部中断），还有 16 级可编程的中断优先级设置，仅仅使用中断优先级设置 8bit 中的高 4 位。

STM32 可以支持 68 个中断通道，已经固定分配给相应外部设备，每个中断通道都具备自己的中断优先级控制字（8 位，但 STM32 只使用 4 位，高 4 位），没 4 个通道的 8 位中断优先级控制字构成一个 32 位的中断优先级寄存器，68 个通道的中断优先级的控制字至少构成 17 个 32 位的中断优先级控制寄存器。具体详细参考 STM32 数据手册。

- **外部中断对应的中断入口函数**

在我们使用的这款单片机中，PA0~PE0 公用 EXIT0 这个中断向量；PA1~PE1 公用 EXIT1 这个中断向量；PA2~PE2 公用 EXIT2 这个中断向量；PA3~PE3 公用 EXIT3 这个中断向量；PA4~PE4 公用 EXIT4 这个中断向量；PA5~9~PE5~9 公用 EXIT9~5 这个中断向量;PA10~15~PE10~15 公用 EXIT15~10 这个中断向量；

我们全部使用库函数编写程序，就不用考虑这中断优先级控制寄存器传送细节，只要把参数传递正确就可以。下面是程序的相关设计说明。

4.4.4.1 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
```

本节实验我们主要用到的库文件，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。

4.4.4.2 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。

4.4.4.3 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"

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。 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库函数的头文件。

4.4.4.4 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_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)
 {
      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
 }
```

外部中断实验是一个比较复杂的实验，要考虑全面了，才能实现成功。通过这个实验能初步掌握中断系统的使用方法。也可以掌握通过延时方法去除输入信号抖动。

4.4.5 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。

4.4.6 GPIO 引脚时钟使能

```
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//打开所有端口的复用功能
```

本节实验用到了 PB 和 PC 端口，所以要把 PB 和 PC 端口的时钟打开，并且打开端口的复用功能。

4.4.7 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 PC 端口产生中断，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);
```

4.4.8 STM32 中断处理函数 stm32f10x_it.c

在本次试验中，中断处理函数是最重要的函数，大家一定要掌握好，现在我们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，也要引入自定义的头文件“pbdata.h”。

```
#include "stm32f10x_it.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_rcc.h"
#include "misc.h"
#include "pbdata.h"//引入自定义头文件
```

下面是外部中断 9_5 处理程序。请大家注意软件滤波的实现方法，在这个例子中，采用延时 10ms 的滤波时间，如果 PC5 管脚状态没有改变，说明有真的中断进入。

```
/*******************************************************************************
* 名    称：void EXTI9_5_IRQHandler(void)
* 功    能：EXTI9-5 中断处理程序
* 入口参数：无
* 出口参数：无
* 说    明：
* 调用方法：无
*******************************************************************************/
void EXTI9_5_IRQHandler(void)
{
  if(EXTI_GetITStatus(EXTI_Line5)==SET)//检查指定（5 号线）线路是否有中断发生
  {
      EXTI_ClearITPendingBit(EXTI_Line5);//清楚中断
   delay_ms(10);//消除抖动
   if(GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_5)==Bit_RESET)//先读入 PC5 管脚状态，如果是低电平，有中断产生
   {
      if(GPIO_ReadOutputDataBit(GPIOB,GPIO_Pin_5)==Bit_RESET)//PB5 管脚状态取反后输出
      {
         //LED 熄灭
```

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

4.4.9 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);
void GPIO_Configuration(void);
void EXTI_Configuration(void);
void NVIC_Configuration(void);

int main(void)
{
   RCC_Configuration();  //系统时钟初始化
   GPIO_Configuration();//端口初始化
   EXTI_Configuration();//中断初始化
   NVIC_Configuration();

   while(1);
}

void RCC_Configuration(void)
{
    SystemInit();
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC,ENABLE);
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
}
```

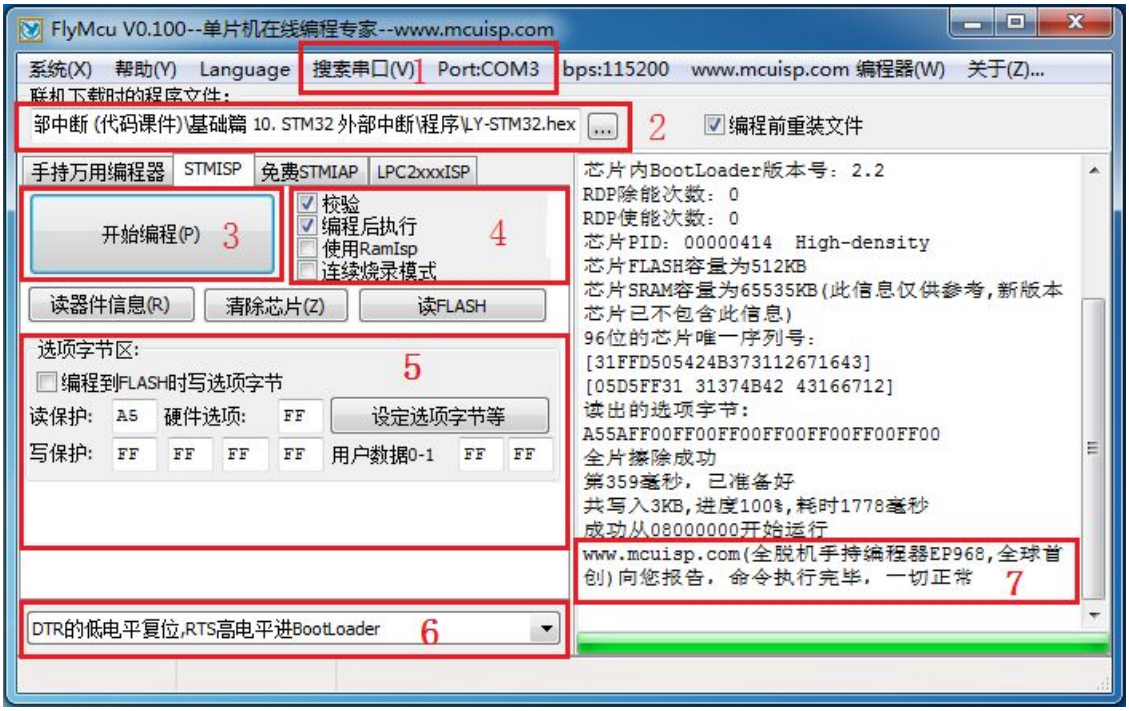
```
void GPIO_Configuration(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    //LED
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;
    GPIO_Init(GPIOB,&GPIO_InitStructure);

    //KEY
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IPU;
    GPIO_Init(GPIOC,&GPIO_InitStructure);
}
//中断配置函数
void EXTI_Configuration(void)
{
    EXTI_InitTypeDef EXTI_InitStructure;

    EXTI_ClearITPendingBit(EXTI_Line5);//清空 5 号线路挂起位
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource5); //选择 C 端
口 5 管脚用作外部中断线路

    EXTI_InitStructure.EXTI_Line=EXTI_Line5;//设置外部中断线 5
    EXTI_InitStructure.EXTI_Mode=EXTI_Mode_Interrupt;//设置成中断请求方式
    EXTI_InitStructure.EXTI_Trigger=EXTI_Trigger_Falling;//设置成下降沿触
发方式
    EXTI_InitStructure.EXTI_LineCmd=ENABLE;//设置中断使能

    EXTI_Init(&EXTI_InitStructure);//以上设置参数传递给单片机
}

void NVIC_Configuration(void)//设置外部中断优先级
{
    NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);// 设置优先级分组
    NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;//外部中断线 9-5 中断
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;//设置先占优
先级
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1; //设置从优先级
    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()、外部中断配置函数 EXTI_Configuration()、外部中断优先级设置函数 NVIC_Configuration()，接下来进入 while(1)循环体，在循环体中，就是等待中断的到来。如果中断到来，就转入中断处理函数，处理中断并相应的输出，使 LED 发光二极管随着每一次键盘按下点亮和熄灭。

在这个实验中大家要着重掌握 NVIC_Configuration()的使用，它涉及了很多参数，语句占用的也很多，并且在函数固件库中用了好几页的篇幅来说明他的功能。大家要注意。

4.4.10 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 10. STM32 外部中断\程序）

4.4.11 实验效果图

外部中断程序写入实验板后，可以看出每按下 K1 键一次，LED1 发光二极管的状态就改变一次。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。