第 5 章 STM32 中断输入

本章主要介绍 STM32 的中断输入，与 STC51 中断的基本原理、执行过程类似，读者需要理解中断源、中断分组、中断优先级及 STM32 对中断的管理，注意 STM32 中断与 STC51 中断的区别，STM32 标准库和 HAL 库编程的区别，Proteus 仿真与实物的区别。

5.1 STM32 中断输入概述

STM32 的中断与 STC51 单片机中断的过程、原理等大同小异。CPU 内部产生的中断称为异常；CPU 外部的片内外设产生的中断称为中断；Cortex 内核具有强大的异常响应系统，它把能够打断当前代码执行流程的事件分为异常(exception)和中断(interrupt)，并把它们用一个表管理起来，编号为 0～15 的称为内核异常，其余的称为中断。用户自己编写的程序，自己指定位置调用称为函数调用。

在启动文件 startup_stm32f10x_md.s 中，有相应芯片可用的全部中断。在编写中断服务函数时，可以从启动文件中定义的中断向量表查找中断服务函数名，这些中断服务程序函数名已经定义好，用户不能更改其名称。如图 5-1 所示，第 62 行～第 76 行是内部异常，第 76 行是 SyStick 中断名称，第 78 行后是片内外设的中断服务程序函数名。

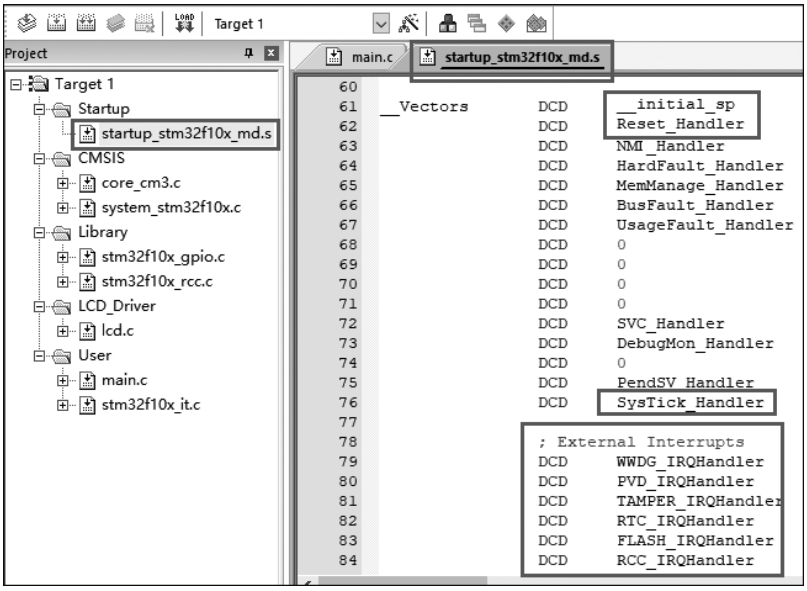


图 5-1 启动文件中的中断服务程序的函数名

STM32 的中断如此之多，配置起来并不容易，因此需要一个强大而方便的中断控制器 NVIC(nested vectored interrupt controller)。NVIC 是属于 Cortex 内核的器件，不可屏蔽中断(NMI)和外部中断都由它来处理，而 SYSTICK 不是由 NVIC 来控制的，如图 5-2 所示。

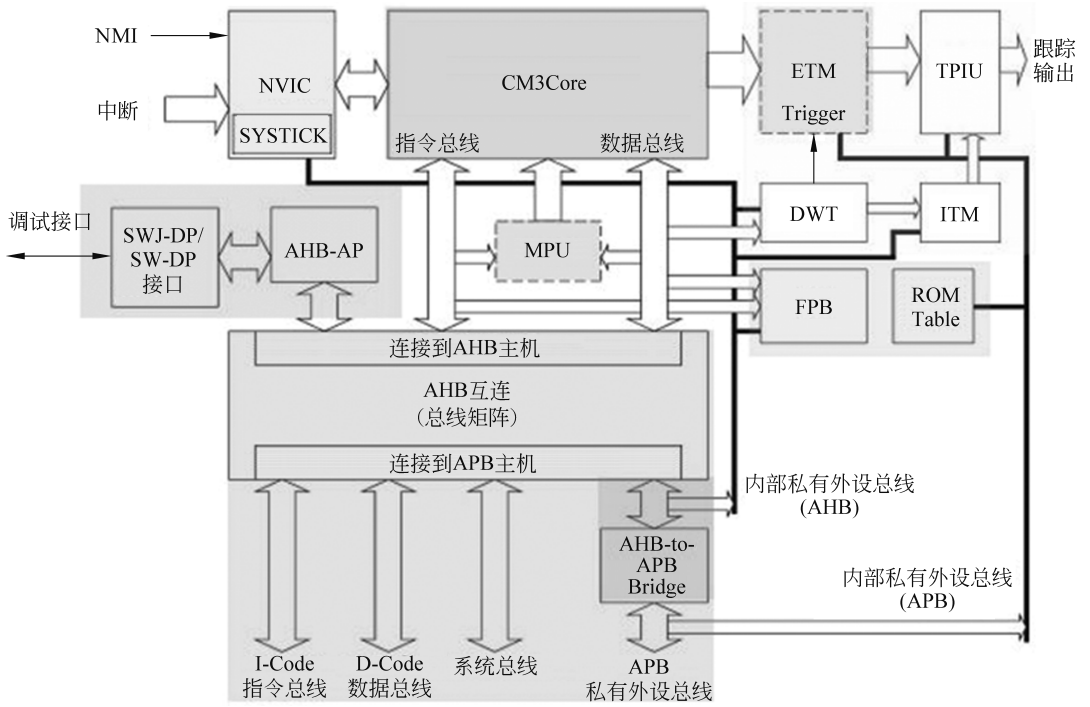


图 5-2　NVIC 在内核中的位置

5.1.1　STM32 中断优先级

在某时刻若有多个中断源同时提出中断申请，那么到底响应哪个中断？或者在执行某个中断服务程序的过程中，又有其他中断源提出中断申请，那么是否响应？编程人员需要提前设置好中断优先级，让优先级高的中断源优先响应。

STM32 的中断具有两个属性：一个为抢占属性；另一个为响应属性。其属性编号越小，表明它的优先级别越高。抢占是指打断其他中断的属性，即因为具有这个属性会出现嵌套中断(在执行中断服务函数 A 的过程中被中断 B 打断，执行完中断服务函数 B 再继续执行中断服务函数 A)，抢占属性由 NVIC_IRQChannelPreemptionPriority 的参数配置。而响应属性则应用在抢占属性相同的情况下，当两个中断向量的抢占优先级相同时，如果两个中断同时到达，则先处理响应优先级高的中断，响应属性由 NVIC_IRQChannelSubPriority 参数配置。例如，现在有 3 个中断向量，如表 5-1 所示。

若内核正在执行 C 的中断服务函数，则它能被抢占优先级更高的中断 A 打断。由于 B 和 C 的抢占优先级相同，因此 C 不能被 B 打断。但如果 B 和 C 中断是同时到达的，内核就会首先响应优先级别更高的 B 中断。

表 5-1　中断举例

中断	抢占优先级	响应优先级
A	0	0
B	1	0
C	1	1

5.1.2　STM32 中断分组

NVIC 只可以配置 16 种中断向量的优先级，也就是说，抢占优先级和响应优先级的数量由一个 4 位的数字来决定，把这个 4 位数字的位数分配成抢占优先级部分和响应优先级部分，即：抢占优先级位数＋响应优先级位数＝4。中断分组及优先级如表 5-2 所示，注意二进制位数与取值范围的关系，n 位二进制数的取值范围为 0～2^n-1。例如，3 位二进制数最大值为 $2^3-1=8-1=7$，即取值范围为 0～7。

表 5-2　中断分组及优先级

组号	含　义	抢占优先级范围	响应优先级范围
0	0 位抢占优先级，4 位响应优先级	无	0～15
1	1 位抢占优先级，3 位响应优先级	0～1	0～7
2	2 位抢占优先级，2 位响应优先级	0～3	0～3
3	3 位抢占优先级，1 位响应优先级	0～7	0～1
4	4 位抢占优先级，0 位响应优先级	0～15	无

采用库函数 NVIC_PriorityGroupConfig()可以配置中断分组，可输入的参数为 NVIC_PriorityGroup_0～NVIC_PriorityGroup_4。

例如：

```
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);     //设置中断分组 2
```

STM32 单片机的所有 I/O 端口都可以配置为 EXTI 中断模式，用来捕捉外部信号，可以配置为下降沿中断、上升沿中断和上升下降沿中断 3 种模式。它们以图 5-3 所示方式连接到 16 个外部中断/事件线上，STM32 的所有 GPIO 都引入到 EXTI 外部中断线上，使得所有的 GPIO 都能作为外部中断的输入源。观察 GPIO 与 EXTI 的连接方式可知，PA0～PG0 连接到 EXTI0、PA1～PG1 连接到 EXTI1……PA15～PG15 连接到 EXTI15。这里大

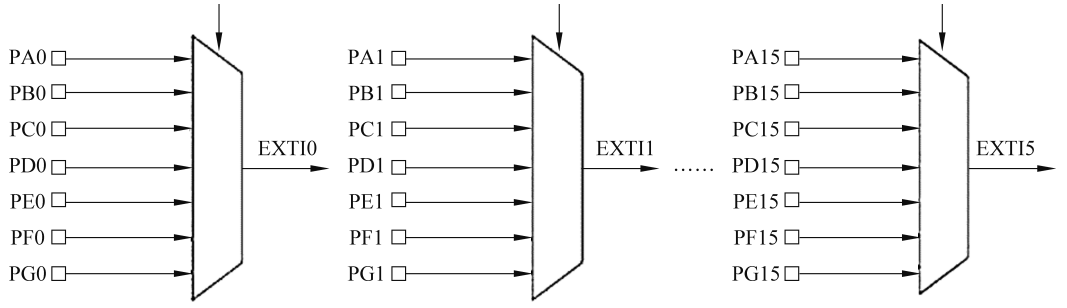


图 5-3　EXTI 与 GPIO 连接图

家要注意的是：PAx～PGx 端口的中断事件都连接到了 EXTIx，即同一时刻 EXTIx 只能响应一个端口的事件触发，不能够同一时间响应所有 GPIO 端口的事件，但可以分时复用。它可以配置为上升沿触发、下降沿触发或双边沿触发。EXTI 最普通的应用就是接上一个按键，设置为下降沿触发，用中断来检测按键。

5.2 STM32 中断输入固件库驱动实例及函数详解

5.2.1 外部中断驱动编程

在固件库的 Project→STM32F10x_StdPeriph_Examples→EXTI→EXTI_Config 文件夹下打开 main.c 文件，外部中断、GPIO 和中断初始化结构体的定义如图 5-4 所示。

外部中断配置代码如图 5-5 所示，开启 GPIO 及其复用功能时钟，再进行外部中断和中断管理初始化。调用 RCC_APB2PeriphClockCmd()时还输入了参数 RCC_APB2Periph_AFIO，表示开启 AFIO 的时钟。

```
main.c - 记事本
文件(F)  编辑(E)  格式(O)  查看(V)  帮助(H)
/* Private typedef -----------------------------------------
/* Private define ------------------------------------------
/* Private macro -------------------------------------------
/* Private variables ---------------------------------------
EXTI_InitTypeDef   EXTI_InitStructure;
GPIO_InitTypeDef   GPIO_InitStructure;
NVIC_InitTypeDef   NVIC_InitStructure;

/* Private function prototypes -----------------------------
void EXTI0_Config(void);
void EXTI9_5_Config(void);

/* Private functions ---------------------------------------
```

图 5-4　初始化结构体的定义

```
main.c - 记事本
文件(F)  编辑(E)  格式(O)  查看(V)  帮助(H)
void EXTI0_Config(void)
{
 /* Enable GPIOA clock */
 RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

 /* Configure PA.00 pin as input floating */
 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
 GPIO_Init(GPIOA, &GPIO_InitStructure);

 /* Enable AFIO clock */
 RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);

 /* Connect EXTI0 Line to PA.00 pin */
 GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0);

 /* Configure EXTI0 line */
 EXTI_InitStructure.EXTI_Line = EXTI_Line0;
 EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
 EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
 EXTI_InitStructure.EXTI_LineCmd = ENABLE;
 EXTI_Init(&EXTI_InitStructure);

 /* Enable and set EXTI0 Interrupt to the lowest priority */
 NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
 NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
 NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
 NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
 NVIC_Init(&NVIC_InitStructure);
```

图 5-5　外部中断配置

AFIO(alternate-function I/O)是指 GPIO 端口的复用功能。GPIO 除了用作普通的输入/输出(主功能)，还可以作为片上外设的复用输入/输出，如串口、ADC，这些就是复用功能。大多数 GPIO 都有一个默认复用功能，有的 GPIO 还有重映射功能。重映射功能是指把原来属于 A 引脚的默认复用功能转移到 B 引脚进行使用，前提是 B 引脚具有这个重映射功能。当把 GPIO 用作 EXTI 外部中断或使用重映射功能的时候，必须开启 AFIO 时钟，而在使用默认复用功能的时候，就不必开启 AFIO 时钟了。

EXTI 初始化配置可参考固件库函数，如图 5-6 所示，打开“STM32F10x 固件库中文解释 V2.0.pdf”并查找 EXTI_Init()函数。

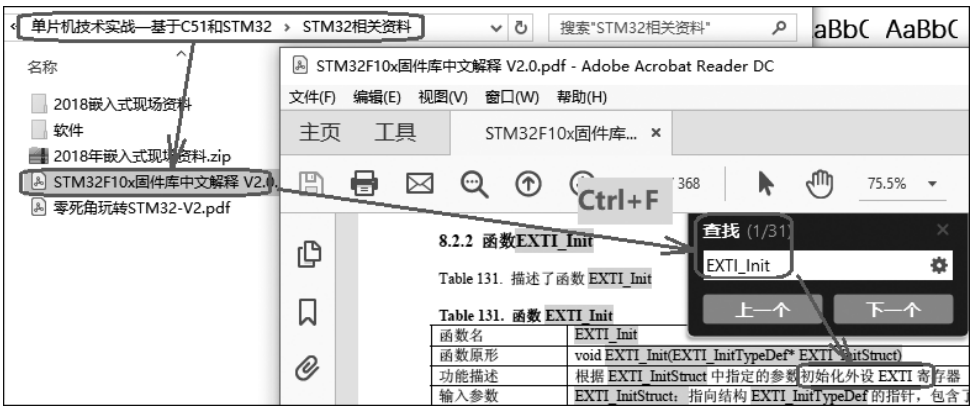


图 5-6　查找外部中断初始化函数

(1) .EXTI_Line=EXTI_Line0：给 EXTI_Line 成员赋值。选择 EXTI_Line0 线进行配置，因为按键的 PA0 连接到了 EXTI_Line0。中断线的线号与 Pin 的引脚号一致，如表 5-3 所示。例如，PB2、PC2、PD2、PE2 等外部中断线均为 Line2。

表 5-3　EXTI_Line 的值及描述

引脚编号	EXTI_Line	描　　述
Pin0	EXTI_Line0	外部中断线 0
Pin1	EXTI_Line1	外部中断线 1
Pin2	EXTI_Line2	外部中断线 2
⋮	⋮	⋮
Pin15	EXTI_Line15	外部中断线 15

(2) .EXTI_Mode=EXTI_Mode_Interrupt：给 EXTI_Mode 成员赋值，把 EXTI_Mode 的模式设置为中断模式(EXTI_Mode_Interrupt)。这个结构体成员也可以赋值为事件模式(EXTI_Mode_Event)，这个模式不会立刻触发中断，而只是在寄存器上把相应的事件标志位置 1，应用这个模式需要不停地查询相应的寄存器。

(3) .EXTI_Trigger=EXTI_Trigger_Falling：给 EXTI_Trigger 成员赋值，把触发方式(EXTI_Trigger)设置为下降沿触发(EXTI_Trigger_Falling)。

(4) .EXTI_LineCmd=ENABLE：给 EXTI_LineCmd 成员赋值，把 EXTI_LineCmd 设置为使能。

(5) 最后调用 EXTI_Init()把 EXTI 初始化结构体的参数写入寄存器。

采用与图 5-6 类似的方法查看固件库 NVIC_Init()函数，打开…\单片机技术实战—基于 C51 和 STM32\STM32 相关资料\STM32F10x 固件库中文解释 V2.0.pdf。

由于本实例工程简单，抢占优先级和响应优先级直接设置为最低级中断。填充完结构体，最后调用 NVIC_Init()函数来向寄存器写入参数。这里要注意的是，如果所用的 I/O 端口是 IO0～IO4，那么对应的中断向量是 EXTI0_IRQ*n*～EXTI4_IRQ*n*；如果所用的 I/O 端

口是 IO5～IO9 中的某一个，对应的中断向量只能是 EXTI9_5_IRQ*n*，即端口 5～9 共用一个中断服务程序；如果用的 I/O 端口是 IO10～IO15 中的一个，对应的中断向量也只能是 EXTI15_10_IRQ*n*。举例：如果 PE5 或者 PE6 作为 EXTI 中断端口，那么对应的中断向量都是 EXTI9_5_IRQ*n*，在同一时刻只能响应来自一个 I/O 端口的 EXTI 中断。

5.2.2 中断服务程序编程

所有的中断服务程序在 stm32f10x_it.c 中实现。“stm32f10x_it.c”文件是专门用来存放中断服务函数的(当然也可以放在其他文件中)。文件中默认只有几个关于系统异常的中断服务函数，而且都是空函数，用户在需要的时候可自行编写。那么中断服务函数名是不是可以自行定义呢？答案是不可以。中断服务函数的名称必须要与启动文件 startup_stm32f10x_md.s 中的中断向量表定义一致。

在固件库的 Project→STM32F10x_StdPeriph_Examples→EXTI→EXTI_Config 文件夹下，再打开 EXTI_Config 下的 stm32f10x_it.c，如图 5-7 所示。

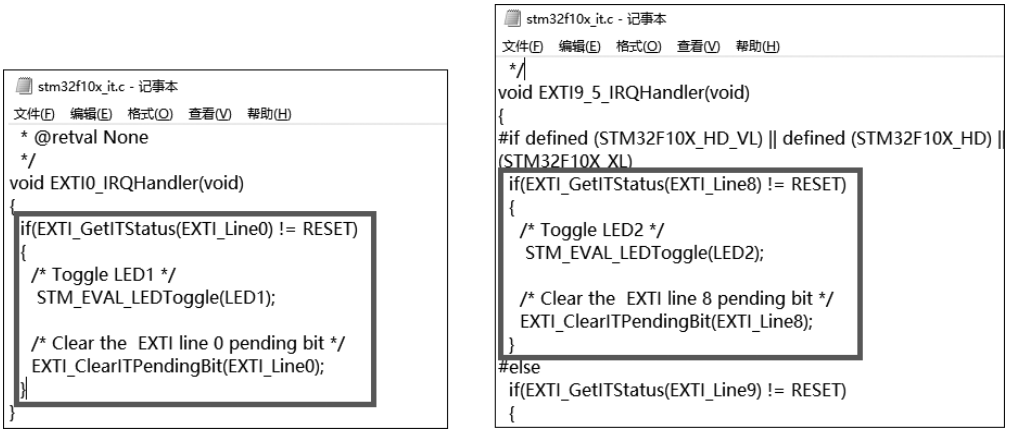


图 5-7 中断服务程序函数实例

EXTI0_IRQHandler 表示为 EXTI0 中断向量的服务函数名。于是，就可以在 stm32f10x_it.c 文件中加入名为 EXTI0_IRQHandler()的函数。

进入中断后，调用库函数 EXTI_GetITStatus()来重新检查是否产生了 EXTI_Line 中断；调用 EXTI_ClearITPendingBit()清除中断标志位再退出中断服务函数。这两个函数的解释见帮助文档。

中断服务程序比较简单，很容易读懂，但在写中断函数入口的时候要注意函数名的写法，函数名只有以下两种命名方法。

(1) EXTI0_IRQHandler；EXTI Line 0
EXTI1_IRQHandler；EXTI Line 1
EXTI2_IRQHandler；EXTI Line 2
EXTI3_IRQHandler；EXTI Line 3
EXTI4_IRQHandler；EXTI Line 4

(2) EXTI9_5_IRQHandler；EXTI Line 9..5
EXTI15_10_IRQHandler；EXTI Line 15..10

中断线在 5 之后的就不能像 0～4 那样只有单独一个函数名，都必须写成 EXTI9_5_IRQHandler 和 EXTI15 _ 10 _ IRQHandle、假如写成 EXTI5 _ IRQHandler、EXTI6 _ IRQHandler……EXTI15_IRQHandler，编译器是不会报错的，不过中断服务程序不能工作。所以如果不知道这样的区别，会浪费很多时间来查找错误。

5.3 STM32 中断输入应用实例：按键中断


外部中断
实例视频

5.3.1 基于标准库的竞赛板上实现

实现的功能要求：

① 屏幕初始化显示如下。

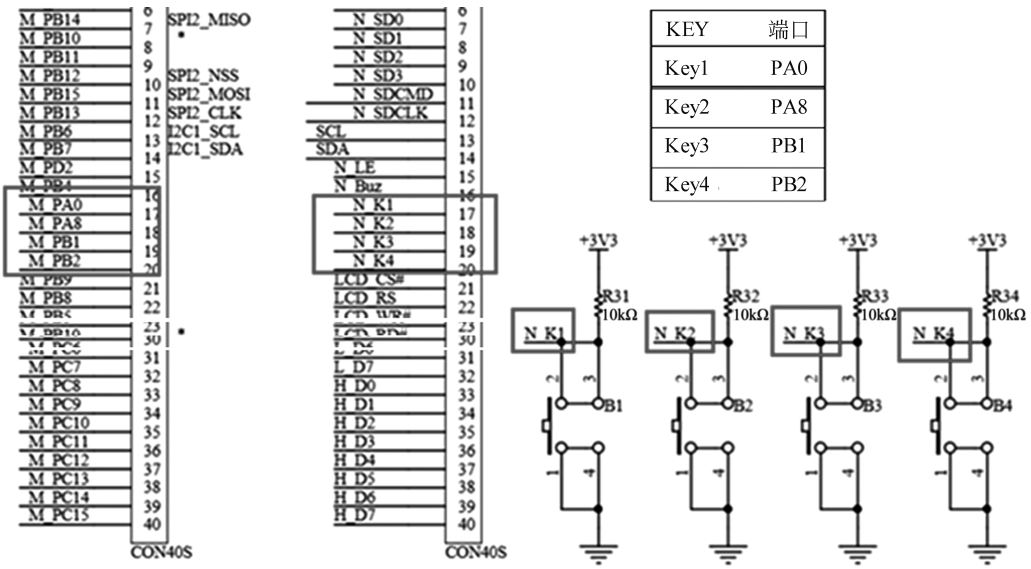
第 1 行："KEY TEST DEMO　　"。

第 3 行："Press the button..."。

② 当按键 B1～B4 中某一个被按下时，分别显示"ButtonN pressed..."（N 表示对应的按键号）。

1. Key 外部中断硬件连接原理

Key 外部中断硬件电路图如图 5-8 所示。当按键未按下时，STM32 端口引脚通过上拉电阻输入高电平（3.3V）；按下按键后，端口引脚接地（0V）。根据电路图，当按下按键时采用中断方式触发，必须开启 PA0、PA8、PB1、PB2 的中断。



KEY	端口
Key1	PA0
Key2	PA8
Key3	PB1
Key4	PB2

图 5-8　Key 外部中断硬件电路图

2. 代码实现

（1）main.c 文件。

```
/*********************************************************************************
* 程序说明：1.使用程序前，确认按键相关引脚已经通过跳线正确连接
*           2.B1-PA0   B2-PA8   B3-PB1   B4-PB2 *
```

```
*******************************************************************************/
#include "stm32f10x.h"
#include "lcd.h"

uint32_t TimingDelay = 0;
uint8_t EXTI_Status = 0;                          //全局变量,记录按键号

void Delay_Ms(uint32_t nTime);
void EXTI_Config(void);
int main(void)
{   SysTick_Config(SystemCoreClock/1000);   //每隔 1ms 中断一次
    EXTI_Config();
    //LCD 工作模式配置
    STM3210B_LCD_Init();
    LCD_Clear(White);
    LCD_SetTextColor(White);
    LCD_SetBackColor(Blue);
    LCD_ClearLine(Line0);
    LCD_ClearLine(Line1);
    LCD_ClearLine(Line2);
    LCD_ClearLine(Line3);
    LCD_ClearLine(Line4);
    LCD_DisplayStringLine(Line1,"   KEY EXTI DEMO    ");
    LCD_DisplayStringLine(Line3," Press the button...");
    LCD_SetTextColor(Blue);
    LCD_SetBackColor(White);
    while(1){
        switch(EXTI_Status)                   //根据按键号,显示不同按键并按下
        {    case 1:
                LCD_DisplayStringLine(Line7," Button1 pressed...");
                break;
            case 2:
                LCD_DisplayStringLine(Line7," Button2 pressed...");
                break;
            case 3:
                LCD_DisplayStringLine(Line7," Button3 pressed...");
                break;
            case 4:
                LCD_DisplayStringLine(Line7," Button4 pressed...");
                break;
        }
    }
}

void EXTI_Config(void) //可以单独新建一个 key.c 文件,将该配置函数配置到 key.c 文件中
{   EXTI_InitTypeDef EXTI_InitStructure;     //定义结构体类型变量
    GPIO_InitTypeDef GPIO_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
    /* Enable GPIOA clock */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);        //开启时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
```

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
//PA0-BUTTON1
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;          //浮空输入
GPIO_Init(GPIOA,&GPIO_InitStructure);                          //PA.0初始化
GPIO_EXTILineConfig(GPIO_PortSourceGPIOA,GPIO_PinSource0); //中断线配置
/* Configure EXTI0 line */
EXTI_InitStructure.EXTI_Line = EXTI_Line0;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStructure);                                //外部中断初始化
/* Enable and set EXTI0 Interrupt to the lowest priority */
NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;   //抢占优先级设置
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;          //响应优先级设置
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);              //外部中断线0中断管理初始化
//PA8-BUTTON2
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA,&GPIO_InitStructure);
GPIO_EXTILineConfig(GPIO_PortSourceGPIOA,GPIO_PinSource8);
/* Configure EXTI9_5 line */
EXTI_InitStructure.EXTI_Line = EXTI_Line8;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStructure);
/* Enable and set EXTI9_5 Interrupt to the lowest priority */
//外部中断5~中断9共用同一个中断服务程序
NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;             //注意此处为9_5
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOB,&GPIO_InitStructure);
GPIO_EXTILineConfig(GPIO_PortSourceGPIOB,GPIO_PinSource1);
/* Configure EXTI1 line */
EXTI_InitStructure.EXTI_Line = EXTI_Line1;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStructure);
/* Enable and set EXTI1 Interrupt to the lowest priority */
NVIC_InitStructure.NVIC_IRQChannel = EXTI1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
```

```
    NVIC_Init(&NVIC_InitStructure);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOB,&GPIO_InitStructure);
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOB,GPIO_PinSource2);
    /* Configure EXTI2 line */
    EXTI_InitStructure.EXTI_Line = EXTI_Line2;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);
    /* Enable and set EXTI2 Interrupt to the lowest priority */
    NVIC_InitStructure.NVIC_IRQChannel = EXTI2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
void Delay_Ms(uint32_t nTime)
{   TimingDelay = nTime;
    while(TimingDelay != 0);
}
```

(2) 外部中断服务程序编程。

在中断文件中增加外部全局变量 extern uint8_t EXTI_Status;,如图 5-9 所示。

图 5-9　增加外部全局变量

增加外部中断函数,代码分别如下。

```
void EXTI0_IRQHandler(void)
{   if(EXTI_GetITStatus(EXTI_Line0) != RESET)
    {   EXTI_Status = 1;
        EXTI_ClearITPendingBit(EXTI_Line0);
    }
}
void EXTI1_IRQHandler(void)
{   if(EXTI_GetITStatus(EXTI_Line1) != RESET)
    {   EXTI_Status = 3;
        EXTI_ClearITPendingBit(EXTI_Line1);
```