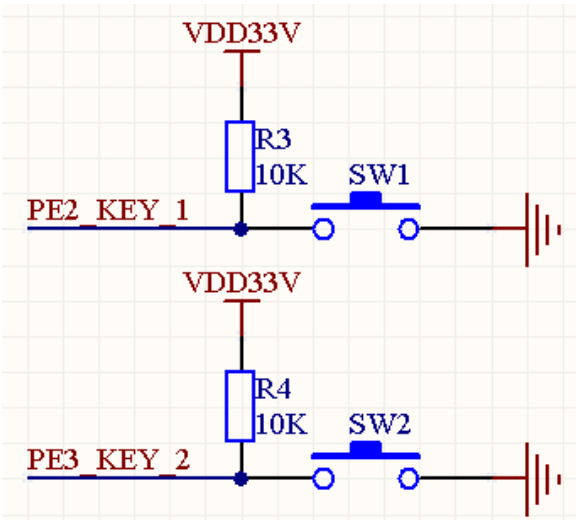
STM32 入门系列教程

初试 STM32 中断

Revision 0.01

(2010-04-23)

其它啥也不说，我们先看下芯达 STM32 外部中断所采用电路，非常简单，如下图所示：



该电路直接从 STM32 系列的 CPU 引脚直接引出两个 GPIO（PE2/PE3），外加上拉电阻后，使用一个轻触开关接地。很明显，按下开关时，PE2/PE3 引脚接地，否则为高电平。

学习 STM32 中断时，我们可以一边回想单片机中断系统一边学习。这里的思路就与单片机类似：当按下按键时，电平变动，使用上升或下降沿触发中断。对于单片机来说，很简单，开中断即可。但 STM32 却有所不同。我们使能配置 EXTI、NVIC 中断后，还需要注意 IO 口时钟的使能。笔者当初忽略了 AFIO 时钟，调试 2 天没有结果，重新查看 datasheet 时，发现 AFIO 时钟没有打开。下面简单列出外部中断的编程思路：

1、系统初始化，如系统时钟初始化，使之进入 72MHZ 主频；
2、GPIO 配置，务必注意打开 GPIO 时钟时，一定打开 AFIO 时钟。
3、EXTI 配置，在这里配置需要选择哪个引脚作为中断引脚。
4、NVIC 配置，这也是比单片机多出来的部分，我们必须把 NVIC 中对应的通道使能，并且设置优先级别。
5、使用 while(1)进行死循环，并在中断程序中写入中断发生时应如何处理。

好吧，开始 STM32 的外部中断的编程之旅吧，详细的例程代码，请参考光盘中的《芯达 STM32 配套例程》文件夹。本期例程使用的模板，是刚刚从 STM32 官网上下载的最新版本的 3.0 固件模板，其工程文件放在根目录下的 \Project\Template\RVMDK 目录中，点击工程文件即可打开。同样只要关注 main.c 文件即可。

步骤一 系统初始化，使用固件模板中自带的 SystemInit();函数即可。通过分析这个函数，我们会发现此函数把主频默认调整到 72MHZ。

步骤二 GPIO 配置，这以后均是需要自己编写的函数，例程中采用 GPIO_Config();函数，如下所示：

```
void GPIO_Config(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;


RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOE
                       | RCC_APB2Periph_AFIO, ENABLE);

/**
 *  LED1 -> PB8      ,   LED2 -> PB9 , LED3 -> PE0 , LED4 -> PE1
 *   Key1 -> PE2    ,   Key2 -> PE3
 */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 |GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 |GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOE, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2 |GPIO_Pin_3;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOE, &GPIO_InitStructure);
}
```

在这里函数里，我们首先使能了 GPIOB、GPIOE 的时钟，因为我们使用 PB8\PB9\PE0\PE1 作为用户指示灯来验证外部中断，使用 PE2\PE3 作为中断触发引脚。注意它们的配置中，同样是 GPIO 引脚，却需要把 PE2\PE3 配置为浮空输入模式，而其他配置为输出模式 —— 很好理解，因为 LED 指示灯是由 CPU 驱动点亮的，而中断引脚 PE2\PE3，是外界按键驱动的，相对于 CPU 来说，是“输入”。

步骤三　EXTI 的配置。此步相当于单片机的中断配置，使用的函数是 EXTI_Config();函数。

```
void EXTI_Config(void)
{
    EXTI_InitTypeDef EXTI_InitStructure;

    GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource2);  // 管 脚选择
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource3);
    EXTI_ClearITPendingBit(EXTI_Line3);
```

```
    EXTI_ClearITPendingBit(EXTI_Line2);

    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_InitStructure.EXTI_Line = EXTI_Line2 | EXTI_Line3;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);
}
```

该函数中，首先指明当前系统中使用哪个引脚作为触发外部中断的引脚，这个直接使用固件库中提供的 GPIO_EXTILineConfig();函数即可。我们指明了 PE2\PE3 引脚作为触发中断引脚。

然后使用 EXTI_ClearITPendingBit();函数清除中断标志位。关于中断标志位，大家应该都很熟悉。我们进入中断服务程序后，首先做的就是清除中断标志位，否则它会不断响应中断，不断进入中断函数。当然，我们这里做这个步骤，只是为了预防万一，您也可以删除此句试验下。

另外需要说明的是 EXTI_Line3，这个表示的是中断线 3。对于外部中断中的 GPIO，有 16 个中断线，分别是 0-15，刚好对应于每个 GPIO 端口的 0-15 引脚。

接下来设置外部中断结构体的成员，比如 EXTI_Mode_Interrupt 即为中断请求，还有一个是 EXTI_Mode_Event，事件请求。这里就有一个问题了，事件和中断到底啥关系呢？请参考如下语句（网络上查的）：

“事件：是表示检测有一某件触发事件发生了。
中断：有某个事件发生并产生中断，并跳转到对应的中断处理程序中。
事件可以触发中断，也可以不触发
中断有可能被更优先的中断屏蔽，事件不会

事件本质上就是一个触发信号,是用来触发特定的外设模块或核心本身(唤醒).
事件只是一个触发信号（脉冲），而中断则是一个固定的电平信号”

我们采用的触发方式是上升沿触发 EXTI_Trigger_Rising，使用的中断线是 EXTI_Line2 和 EXTI_Line3，实际上就是指明使用的是 GPIO 的第 2 引脚和第 3 引脚。最后不要忘了务必要使能它，呵呵。

步骤四 NVIC 配置。先解释下 NVIC，它的中文全称是“嵌套向量中断控制器。”我们进行 NVIC 配置时，采用的是 NVIC_Config();函数，下面列出该函数：

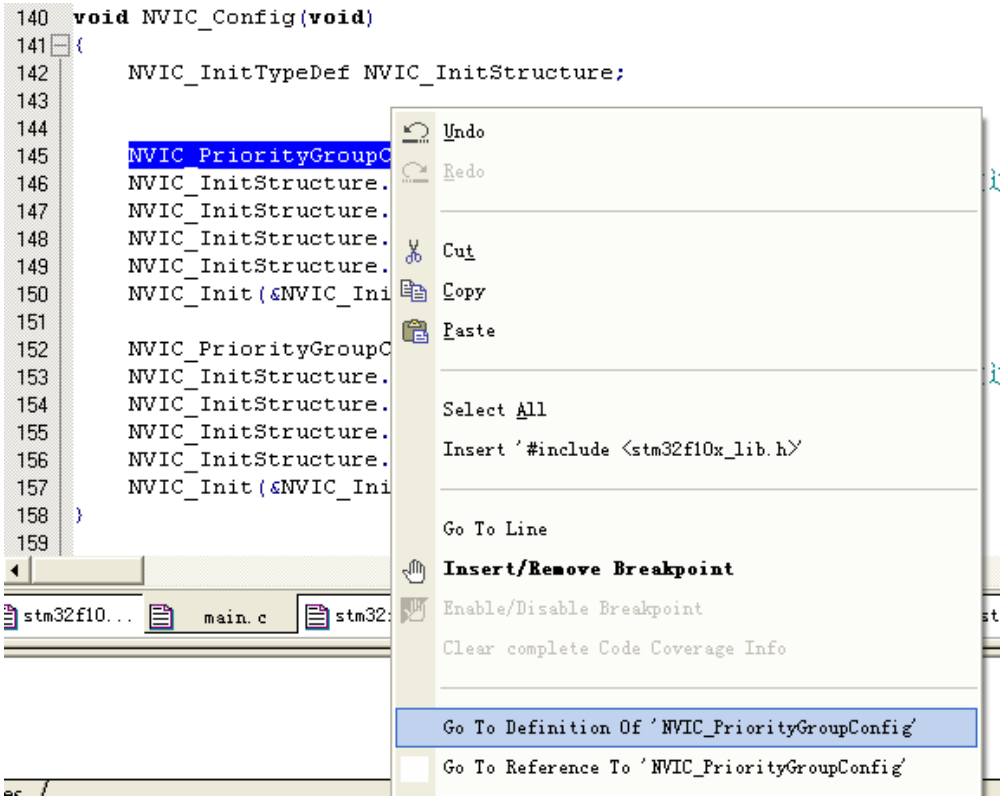
```
void NVIC_Config(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
    NVIC_InitStructure.NVIC_IRQChannel = EXTI2_IRQChannel;     //通道
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;//
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;    //
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
```

```
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
    NVIC_InitStructure.NVIC_IRQChannel = EXTI3_IRQChannel;     //通道
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;//
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;    //
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

该函数的编写分两个部分，一个部分是专门针对中断线 2 的，另一个部分针对中断线 3 的。实际上学习外部中断，只要其中一个即可。由于芯达 STM32 开发板上配置了两个按键，都可以用作外部中断的学习，因此我们这里列出了两个中断线。

整个函数比较简单，从函数名即可知道其含义。大家在查看这些函数的时候，请务必查看他们具体是如何实现的。比如 NVIC_PriorityGroupConfig();函数，不能只看这个函数是用于优先级组配置，还要稍微关注下它的实现。操作是，双击该函数名，选中它，然后右键，点击“Go To Definiton of ‘NVIC_PriorityGroupConfig’”，如下图所示，即可查看该函数的具体实现。



步骤五　中断服务子程序的编写。经过以上四个步骤，我们已经把中断配置完毕。剩下的工作，就是编写中断服务子程序，实现进入中断后会如何处理。我们在这里举了 PE2 上的开关按下后的动作：

```
void EXTI2_IRQHandler(void)
{
    if ( EXTI_GetITStatus(EXTI_Line2) != RESET ) {
```

```
            EXTI_ClearITPendingBit(EXTI_Line2);
            /*====LED34-ON=======*/
            GPIO_SetBits(GPIOB , GPIO_Pin_8);
            GPIO_SetBits(GPIOB , GPIO_Pin_9);
            GPIO_ResetBits(GPIOE , GPIO_Pin_0);
            GPIO_SetBits(GPIOE , GPIO_Pin_1);
            Delay(0xfffff);
            Delay(0xfffff);
            Delay(0xfffff);
        }
    }
```

EXTI_GetITStatus()函数的返回值如果不是 RESET，说明此时确实有中断发生了，因此，第一步，把中断标志位清除，以防它不断地进入中断。然后点亮 LED 指示灯。怎么点亮 LED 灯，这个就不详细展开解释，读者可根据自己的实际情况来编写程序。

值得说明的是，该函数应该写在 stm32f10x_it.c 文件中，并且中断服务子程序的函数名，必须严格按照 startup_stm32f10x_hd.s 文件中的中断向量表中，对应的向量名进行。如果担心写错函数名，可以直接打开 startup_stm32f10x_hd.s 文件，找到如下部分：

```
__Vectors       DCD     __initial_spTop              ; Top of Stack
                DCD     Reset_Handler                ; Reset Handler
                DCD     NMI_Handler                  ; NMI Handler
                DCD     HardFault_Handler            ; Hard Fault Handler
                DCD       MemManage_Handler                  ; MPU  Fault
Handler
                DCD     BusFault_Handler             ; Bus Fault Handler
                DCD     UsageFault_Handler           ; Usage Fault Handler
                DCD     0                            ; Reserved
                DCD     0                            ; Reserved
                DCD     0                            ; Reserved
                DCD     0                            ; Reserved
                DCD     SVC_Handler                  ; SVCall Handler
                DCD       DebugMon_Handler                 ; Debug  Monitor
Handler
                DCD     0                            ; Reserved
                DCD     PendSV_Handler               ; PendSV Handler
                DCD     SysTick_Handler              ; SysTick Handler

                ; External Interrupts
                DCD       WWDG_IRQHandler                  ; Window
Watchdog
                DCD     PVD_IRQHandler               ; PVD through EXTI
Line detect
                DCD     TAMPER_IRQHandler            ; Tamper
```

```
                DCD     RTC_IRQHandler              ; RTC
                DCD     FLASH_IRQHandler            ; Flash
                DCD     RCC_IRQHandler              ; RCC
                DCD     EXTI0_IRQHandler            ; EXTI Line 0
                DCD     EXTI1_IRQHandler            ; EXTI Line 1
                DCD     EXTI2_IRQHandler            ; EXTI Line 2
                DCD     EXTI3_IRQHandler            ; EXTI Line 3
                DCD     EXTI4_IRQHandler            ; EXTI Line 4
                DCD     DMA1_Channel1_IRQHandler    ; DMA1 Channel 1
                DCD     DMA1_Channel2_IRQHandler    ; DMA1 Channel 2
                DCD     DMA1_Channel3_IRQHandler    ; DMA1 Channel 3
                DCD     DMA1_Channel4_IRQHandler    ; DMA1 Channel 4
                DCD     DMA1_Channel5_IRQHandler    ; DMA1 Channel 5
                DCD     DMA1_Channel6_IRQHandler    ; DMA1 Channel 6
                DCD     DMA1_Channel7_IRQHandler    ; DMA1 Channel 7
                DCD     ADC1_2_IRQHandler           ; ADC1 & ADC2
                DCD       USB_HP_CAN1_TX_IRQHandler     ;  USB  High
Priority or CAN1 TX
                DCD       USB_LP_CAN1_RX0_IRQHandler  ;  USB  Low
Priority or CAN1 RX0
                DCD     CAN1_RX1_IRQHandler         ; CAN1 RX1
                DCD     CAN1_SCE_IRQHandler         ; CAN1 SCE
                DCD     EXTI9_5_IRQHandler          ; EXTI Line 9..5
                DCD     TIM1_BRK_IRQHandler         ; TIM1 Break
                DCD     TIM1_UP_IRQHandler          ; TIM1 Update
                DCD       TIM1_TRG_COM_IRQHandler       ;  TIM1  Trigger
and Commutation
                DCD       TIM1_CC_IRQHandler            ;  TIM1  Capture
Compare
                DCD     TIM2_IRQHandler             ; TIM2
                DCD     TIM3_IRQHandler             ; TIM3
                DCD     TIM4_IRQHandler             ; TIM4
                DCD     I2C1_EV_IRQHandler          ; I2C1 Event
                DCD     I2C1_ER_IRQHandler          ; I2C1 Error
                DCD     I2C2_EV_IRQHandler          ; I2C2 Event
                DCD     I2C2_ER_IRQHandler          ; I2C2 Error
                DCD     SPI1_IRQHandler             ; SPI1
                DCD     SPI2_IRQHandler             ; SPI2
                DCD     USART1_IRQHandler           ; USART1
                DCD     USART2_IRQHandler           ; USART2
                DCD     USART3_IRQHandler           ; USART3
                DCD     EXTI15_10_IRQHandler        ; EXTI Line 15..10
                DCD     RTCAlarm_IRQHandler         ; RTC Alarm through
EXTI Line
```

```
                DCD       USBWakeUp_IRQHandler          ; USB Wakeup
from suspend
                DCD       TIM8_BRK_IRQHandler           ; TIM8 Break
                DCD       TIM8_UP_IRQHandler            ; TIM8 Update
                DCD       TIM8_TRG_COM_IRQHandler       ; TIM8 Trigger
and Commutation
                DCD       TIM8_CC_IRQHandler            ; TIM8 Capture
Compare
                DCD       ADC3_IRQHandler               ; ADC3
                DCD       FSMC_IRQHandler               ; FSMC
                DCD       SDIO_IRQHandler               ; SDIO
                DCD       TIM5_IRQHandler               ; TIM5
                DCD       SPI3_IRQHandler               ; SPI3
                DCD       UART4_IRQHandler              ; UART4
                DCD       UART5_IRQHandler              ; UART5
                DCD       TIM6_IRQHandler               ; TIM6
                DCD       TIM7_IRQHandler               ; TIM7
                DCD       DMA2_Channel1_IRQHandler      ; DMA2 Channel1
                DCD       DMA2_Channel2_IRQHandler      ; DMA2 Channel2
                DCD       DMA2_Channel3_IRQHandler      ; DMA2 Channel3
                DCD       DMA2_Channel4_5_IRQHandler    ; DMA2 Channel4
& Channel5
__Vectors_End
```

我们现在使用的是中断线 2 的处理函数，因此是上面红色粗体所示的函数名：**EXTI2_IRQHandler**。

步骤六 编写 while 死循环程序，指明没有中断发生时，系统应该运行什么程序。我们编写了一个死循环：当没有中断发生时，不断地让指示灯 LED1 与 LED2 进行闪烁处理。按下按键后，关掉指示灯 LED1 与 LED2，同时点亮对应的 LED3 或者 LED4。

至此，STM32 的外部中断教程编写结束，感谢您阅读本教程。如果您对串口操作还有不明白的地方，请直接到我们的合作网站：ARM 技术交流网 www.arm79.com，进行讨论。我们将会尽快给您做出答复。

附：

福州芯达工作室简介

福州芯达工作室成立于 2009 年 9 月，我们专注于嵌入式产品的研发与推广，目前芯达产品涉及 ARM9 系列、STM32 系列。

芯达团队成员均硕士研究生毕业，具有一定研发实力。我们的愿景在于把福州芯达打造成国内一流的嵌入式品牌。或许我们现在做的还不够，但是我们真的努力在做，希望通过我们的努力，能够在您学习和使用芯达产品的过程中带来或多或少的帮助。

这是芯达为了配合 STM32 开发板而推出的入门系列教程。如果您在看了我们的教程后，理清了思路，我们都会倍感欣慰！让我们一起学习，共同进步，在征服嵌入式领域的道路上风雨同行！

官方网站：http://www.arm79.com/

官方淘宝：http://shop36353570.taobao.com/