STM32中断处理

内容提要

0 中断原理的回顾

1 STM32的中断和异常

2 嵌套向量中断控制器NVIC

3 NVIC库函数

4 EXTI外部中断

5 外设中断的库函数

6 EXTI应用示例：按键中断

0 中断原理回顾

数据传送方式
- 程序查询传送
- 中断传送
- DMA传送

- 为什么需要中断

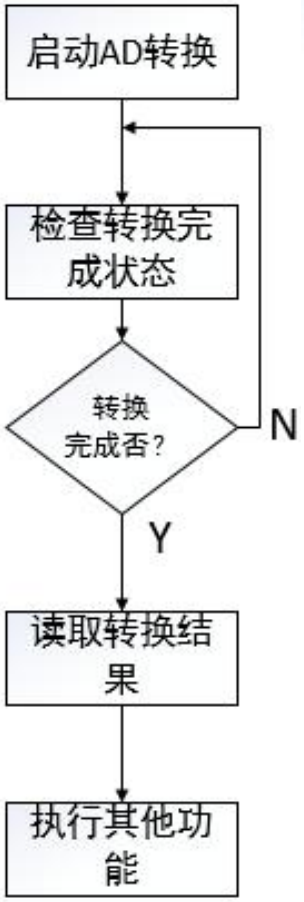
  便于处理异步事件，提高cpu利用率

- 中断来了之后，cpu需要做什么

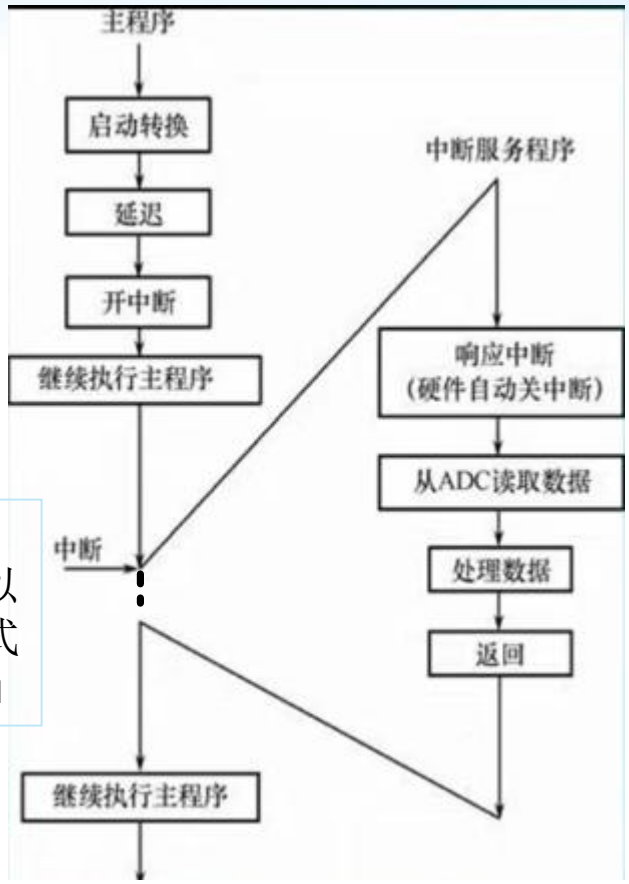
  保存断点等，并转向中断服务程序

- 中断处理完成后，怎么办

  返回断点处继续执行被中断打断的程序



查询方式读取AD转换结果



AD转换完成后以中断方式通知cpu

中断方式读取AD转换结果

1 STM32的中断和异常

- 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，而16 以上的则称为外部中断（外是相对内核而言），这个表就称为中断向量表。
- STM32 对这个表重新进行了编排，把编号从-3 至6的中断向量定义为系统异常，编号为负 的内核异常不能被设置优先级，如复位(Reset)、不可屏蔽中断(NMI)、硬错误(Hardfault)。从编号7 开始的为外部中断，这些中断的优先级都是可以自行设置的。

优先级为-3~6的系统异常

表55　其它STM32F10xxx产品(小容量、中容量和大容量)的向量表

位置	优先级	优先级类型	名称	说明	地址
	-	-	-	保留	0x0000_0000
	-3	固定	Reset	复位	0x0000_0004
	-2	固定	NMI	不可屏蔽中断 RCC时钟安全系统(CSS)联接到NMI向量	0x0000_0008
	-1	固定	硬件失效(HardFault)	所有类型的失效	0x0000_000C
	0	可设置	存储管理(MemManage)	存储器管理	0x0000_0010
	1	可设置	总线错误(BusFault)	预取指失败，存储器访问失败	0x0000_0014
	2	可设置	错误应用(UsageFault)	未定义的指令或非法状态	0x0000_0018
	-	-	-	保留	0x0000_001C ~0x0000_002B
	3	可设置	SVCall	通过SWI指令的系统服务调用	0x0000_002C
	4	可设置	调试监控(DebugMonitor)	调试监控器	0x0000_0030
	-	-	-	保留	0x0000_0034
	5	可设置	PendSV	可挂起的系统服务	0x0000_0038
	6	可设置	SysTick	系统嘀嗒定时器	0x0000_003C

STM32的外部中断

位置	优先级	优先级类型	名称	说明	地址
0	7	可设置	WWDG	窗口定时器中断	0x0000_0040
1	8	可设置	PVD	连到EXTI的电源电压检测(PVD)中断	0x0000_0044
2	9	可设置	TAMPER	侵入检测中断	0x0000_0048
3	10	可设置	RTC	实时时钟(RTC)全局中断	0x0000_004C
4	11	可设置	FLASH	闪存全局中断	0x0000_0050
5	12	可设置	RCC	复位和时钟控制(RCC)中断	0x0000_0054
6	13	可设置	EXTI0	EXTI线0中断	0x0000_0058
7	14	可设置	EXTI1	EXTI线1中断	0x0000_005C
8	15	可设置	EXTI2	EXTI线2中断	0x0000_0060
9	16	可设置	EXTI3	EXTI线3中断	0x0000_0064
10	17	可设置	EXTI4	EXTI线4中断	0x0000_0068
11	18	可设置	DMA1通道1	DMA1通道1全局中断	0x0000_006C
12	19	可设置	DMA1通道2	DMA1通道2全局中断	0x0000_0070
13	20	可设置	DMA1通道3	DMA1通道3全局中断	0x0000_0074
14	21	可设置	DMA1通道4	DMA1通道4全局中断	0x0000_0078
15	22	可设置	DMA1通道5	DMA1通道5全局中断	0x0000_007C
16	23	可设置	DMA1通道6	DMA1通道6全局中断	0x0000_0080
17	24	可设置	DMA1通道7	DMA1通道7全局中断	0x0000_0084

STM32的外部中断（优先级25—41）

位置	优先级	优先级类型	名称	说明	地址
18	25	可设置	ADC1_2	ADC1和ADC2的全局中断	0x0000_0088
19	26	可设置	USB_HP_CAN_TX	USB高优先级或CAN发送中断	0x0000_008C
20	27	可设置	USB_LP_CAN_RX0	USB低优先级或CAN接收0中断	0x0000_0090
21	28	可设置	CAN_RX1	CAN接收1中断	0x0000_0094
22	29	可设置	CAN_SCE	CAN SCE中断	0x0000_0098
23	30	可设置	EXTI9_5	EXTI线[9:5]中断	0x0000_009C
24	31	可设置	TIM1_BRK	TIM1刹车中断	0x0000_00A0
25	32	可设置	TIM1_UP	TIM1更新中断	0x0000_00A4
26	33	可设置	TIM1_TRG_COM	TIM1触发和通信中断	0x0000_00A8
27	34	可设置	TIM1_CC	TIM1捕获比较中断	0x0000_00AC
28	35	可设置	TIM2	TIM2全局中断	0x0000_00B0
29	36	可设置	TIM3	TIM3全局中断	0x0000_00B4
30	37	可设置	TIM4	TIM4全局中断	0x0000_00B8
31	38	可设置	I2C1_EV	I^2C1事件中断	0x0000_00BC
32	39	可设置	I2C1_ER	I^2C1错误中断	0x0000_00C0
33	40	可设置	I2C2_EV	I^2C2事件中断	0x0000_00C4
34	41	可设置	I2C2_ER	I^2C2错误中断	0x0000_00C8

STM32的外部中断（优先级42--55）

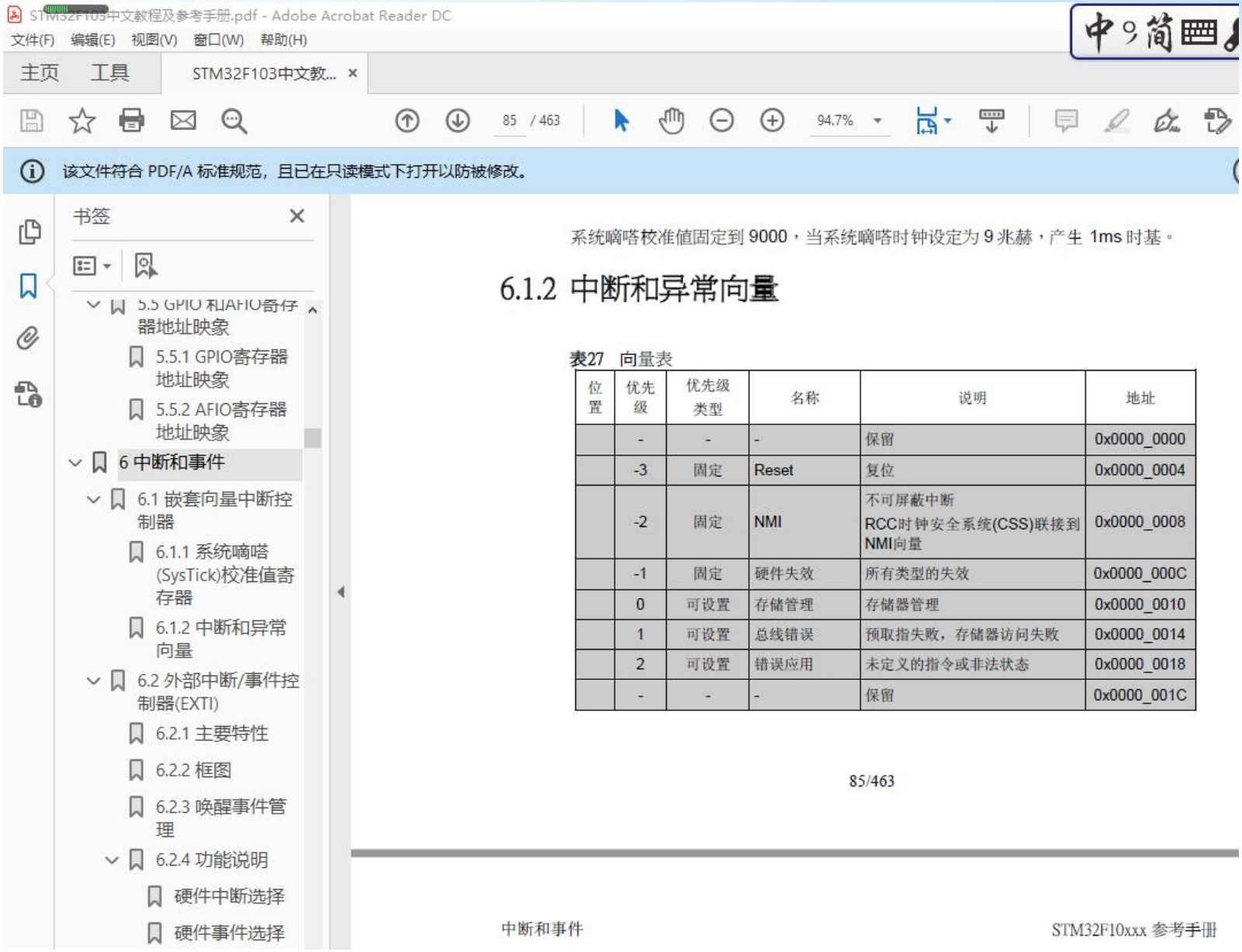
位置	优先级	优先级类型	名称	说明	地址
35	42	可设置	SPI1	SPI1全局中断	0x0000_00CC
36	43	可设置	SPI2	SPI2全局中断	0x0000_00D0
37	44	可设置	USART1	USART1全局中断	0x0000_00D4
38	45	可设置	USART2	USART2全局中断	0x0000_00D8
39	46	可设置	USART3	USART3全局中断	0x0000_00DC
40	47	可设置	EXTI15_10	EXTI线[15:10]中断	0x0000_00E0
41	48	可设置	RTCAlarm	连到EXTI的RTC闹钟中断	0x0000_00E4
42	49	可设置	USB唤醒	连到EXTI的从USB待机唤醒中断	0x0000_00E8
43	50	可设置	TIM8_BRK	TIM8刹车中断	0x0000_00EC
44	51	可设置	TIM8_UP	TIM8更新中断	0x0000_00F0
45	52	可设置	TIM8_TRG_COM	TIM8触发和通信中断	0x0000_00F4
46	53	可设置	TIM8_CC	TIM8捕获比较中断	0x0000_00F8
47	54	可设置	ADC3	ADC3全局中断	0x0000_00FC
48	55	可设置	FSMC	FSMC全局中断	0x0000_0100

STM32的外部中断（优先级56—66）

位置	优先级	优先级类型	名称	说明	地址
49	56	可设置	SDIO	SDIO全局中断	0x0000_0104
50	57	可设置	TIM5	TIM5全局中断	0x0000_0108
51	58	可设置	SPI3	SPI3全局中断	0x0000_010C
52	59	可设置	UART4	UART4全局中断	0x0000_0110
53	60	可设置	UART5	UART5全局中断	0x0000_0114
54	61	可设置	TIM6	TIM6全局中断	0x0000_0118
55	62	可设置	TIM7	TIM7全局中断	0x0000_011C
56	63	可设置	DMA2通道1	DMA2通道1全局中断	0x0000_0120
57	64	可设置	DMA2通道2	DMA2通道2全局中断	0x0000_0124
58	65	可设置	DMA2通道3	DMA2通道3全局中断	0x0000_0128
59	66	可设置	DMA2通道4_5	DMA2通道4和DMA2通道5全局中断	0x0000_012C

- 上述表格可以从《STM32中文参考手册》找到，但一般建议从启动文件（如startup_stm32f10x_hd.s）中查找
- 不同型号STM32微控制器的向量表略有区别
  - 启动文件包含了其支持的所有中断
  - 并给出了中断服务函数名

《STM32F103中文教程及参考手册》中关于中断的描述



系统嘀嗒校准值固定到 9000，当系统嘀嗒时钟设定为 9 兆赫，产生 1ms 时基。

6.1.2 中断和异常向量

表27 向量表

位置	优先级	优先级类型	名称	说明	地址
	-	-	-	保留	0x0000_0000
	-3	固定	Reset	复位	0x0000_0004
	-2	固定	NMI	不可屏蔽中断 RCC时钟安全系统(CSS)联接到NMI向量	0x0000_0008
	-1	固定	硬件失效	所有类型的失效	0x0000_000C
	0	可设置	存储管理	存储器管理	0x0000_0010
	1	可设置	总线错误	预取指失败，存储器访问失败	0x0000_0014
	2	可设置	错误应用	未定义的指令或非法状态	0x0000_0018
	-	-	-	保留	0x0000_001C

85/463

中断和事件 STM32F10xxx 参考手册

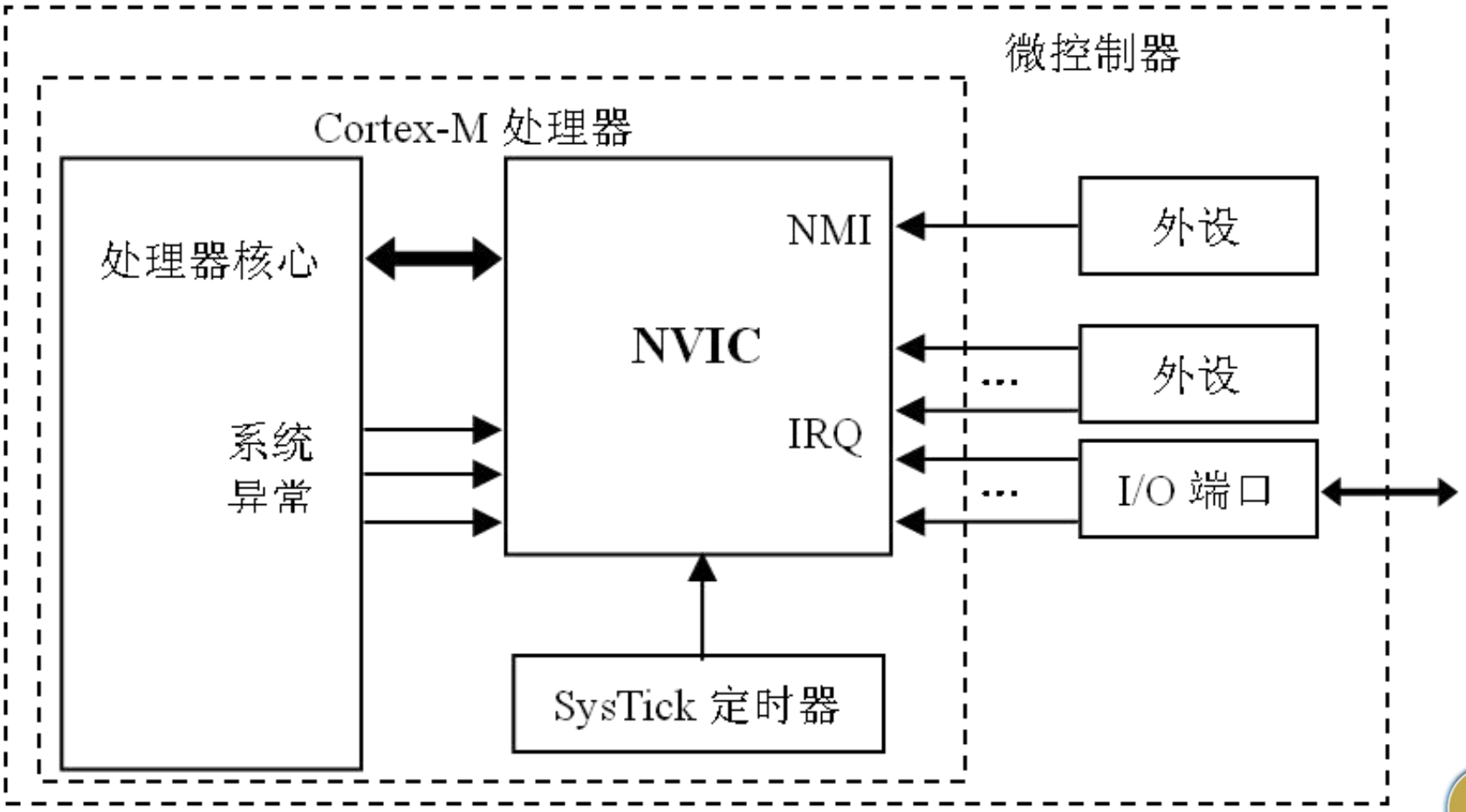
stm32f10x启动代码中关于中断向量的描述

misc.c | main.c | exti.c | led.c | startup_stm32f10x_hd.s

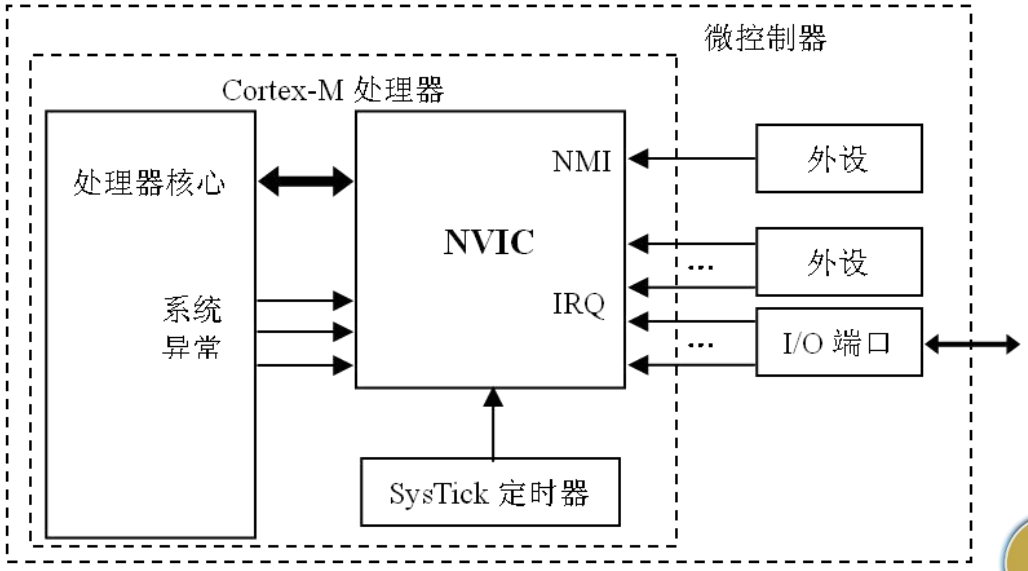
```
55
56  ; Vector Table Mapped to Address 0 at Reset
57                  AREA    RESET, DATA, READONLY
58                  EXPORT  __Vectors
59                  EXPORT  __Vectors_End
60                  EXPORT  __Vectors_Size
61
62  __Vectors       DCD     __initial_sp              ; Top of Stack
63                  DCD     Reset_Handler             ; Reset Handler
64                  DCD     NMI_Handler               ; NMI Handler
65                  DCD     HardFault_Handler         ; Hard Fault Handler
66                  DCD     MemManage_Handler         ; MPU Fault Handler
67                  DCD     BusFault_Handler          ; Bus Fault Handler
68                  DCD     UsageFault_Handler        ; Usage Fault Handler
69                  DCD     0                         ; Reserved
70                  DCD     0                         ; Reserved
71                  DCD     0                         ; Reserved
72                  DCD     0                         ; Reserved
73                  DCD     SVC_Handler               ; SVCall Handler
74                  DCD     DebugMon_Handler          ; Debug Monitor Handler
75                  DCD     0                         ; Reserved
76                  DCD     PendSV_Handler            ; PendSV Handler
77                  DCD     SysTick_Handler           ; SysTick Handler
78
79                  ; External Interrupts
80                  DCD     WWDG_IRQHandler           ; Window Watchdog
81                  DCD     PVD_IRQHandler            ; PVD through EXTI Line detect
82                  DCD     TAMPER_IRQHandler         ; Tamper
83                  DCD     RTC_IRQHandler            ; RTC
84                  DCD     FLASH_IRQHandler          ; Flash
85                  DCD     RCC_IRQHandler            ; RCC
86                  DCD     EXTI0_IRQHandler          ; EXTI Line 0
87                  DCD     EXTI1_IRQHandler          ; EXTI Line 1
88                  DCD     EXTI2_IRQHandler          ; EXTI Line 2
89                  DCD     EXTI3_IRQHandler          ; EXTI Line 3
90                  DCD     EXTI4_IRQHandler          ; EXTI Line 4
91                  DCD     DMA1_Channel1_IRQHandler  ; DMA1 Channel 1
92                  DCD     DMA1 Channel2 IRQHandler  ; DMA1 Channel 2
```

2嵌套向量中断控制器NVIC

- NVIC（Nested Vectored Interrupt Controller内嵌向量中断控制器）是属于Cortex内核的器件，中断都由它来管理。



- 中断请求IRQ（Intrrupt Request）
  - 通常来自片上外设或经I/O端口的外部中断输入
- 非屏蔽中断NMI（Non-Maskable Interrupt）请求
  - 可来自看门狗定时器或供电电压下降探测电路
- 系统时钟(滴答)SysTick，可以产生周期性中断请求
  - 可用于各种定时控制



抢占优先级和响应优先级

- STM32的中断向量具有两个属性，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，是指打断其它中断的属性，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)。
- 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。

- 例如，现在有三个中断向量

中断向量	抢占优先级	响应优先级
A	0	0
B	1	0
C	1	1

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，所以C不能被B打断。但如果B和C中断是同时到达的，内核就会首先响应响应优先级别更高的B中断。

NVIC的优先级抢占

- 在配置优先级的时候，还要注意一个很重要的问题，中断种类的数量。NVIC只可以配置最多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。

NVIC优先级配置

- STM32使用4位优先级寄存器、支持16种中断向量的优先级
- 4位分成抢占（组）优先级和响应（子）优先级
- 有5组分配形式（下页表）

STM32的5组优先级配置

组号	组合方式	抢占优先级	响应优先级
0	0-4组合	无	4位都配置响应优先级 16种中断向量具有不同的响应优先级
1	1-3组合	1位配置抢占优先级 2种抢占优先级(0、1级)	3位配置响应优先级 8种响应优先级（0~7）
2	2-2组合	2位配置抢占优先级 4种抢占优先级(0~3级)	2位配置响应优先级 4种响应优先级（0~3）
3	3-1组合	3位配置抢占优先级 8种抢占优先级(0~7级)	1位配置响应优先级 2种响应优先级（0、1）
4	4-0组合	4位都配置抢占优先级 16种抢占优先级(0~15级)	无

3 NVIC库函数

使用NVIC之前一定要配置NVIC，STM32库在misc.c中提供了NVIC配置函数，使用中断时，一定要在项目中添加misc.c文件。

函数名	函数功能
NVIC_PriorityGroupConfig	配置优先权组合方式
NVIC_Init	NVIC初始化
NVIC_SetVectorTable	设置向量表位置和偏移
NVIC_SystemLPConfig	为系统进入低功耗模式选择条件
SysTick_CLKSourceConfig	配置系统时钟SysTick时钟源

NVIC优先级分组函数

NVIC优先级分组配置函数NVIC_PriorityGroupConfig，

- 其函数申明如下：

```
void NVIC_PriorityGroupConfig( uint32_t
                    NVIC_PriorityGroup );
```

这个函数的作用是对中断的优先级进行分组，这个函数在系统中只能被调用一次，一旦分组确定就最好不要更改。

- 参数NVIC_PriorityGroup的5种取指，分别对应抢占优先级+响应优先级的五种分配组合

NVIC_PriorityGroup_0：0-4组合
NVIC_PriorityGroup_1：1-3组合
NVIC_PriorityGroup_2：2-2组合
NVIC_PriorityGroup_3：3-1组合
NVIC_PriorityGroup_4：4-0组合

- 比如：NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);

就确定了为“ 2 位抢占优先级， 2 位响应优先级”。

NVIC初始化函数 NVIC_Init ，其函数申明为

- void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct)

其中：

NVIC_InitTypeDef 是一个结构体，其结构体的成员如下：

```
typedef struct{
uint8_t NVIC_IRQChannel;
uint8_t NVIC_IRQChannelPreemptionPriority;
uint8_t NVIC_IRQChannelSubPriority;
FunctionalState NVIC_IRQChannelCmd;
} NVIC_InitTypeDef;
```

NVIC结构体成员（NVIC初始化时要用到）

- NVIC_InitTypeDef结构

```
typedef struct
{
   uint8_t  NVIC_IRQChannel;
   uint8_t  NVIC_IRQChannelPreemptionPriority;
   uint8_t  NVIC_IRQChannelSubPriority;
   FunctionalState  NVIC_IRQChannelCmd;
} NVIC_InitTypeDef;
```

注：

NVIC_IRQChannel：定义初始化的是哪个中断，可以在 stm32f10x.h 中找到每个中断对应的名字。例如 USART1_IRQn 。

NVIC_IRQChannelPreemptionPriority ：定义这个中断的抢占优先级别。

NVIC_IRQChannelSubPriority：定义这个中断的子优先级别。

NVIC_IRQChannelCmd：该中断是否使能。

比如：要使能串口1 的中断，同时设置抢占优先级为 1 ，子优先级为 2 ，初始化的写法是：

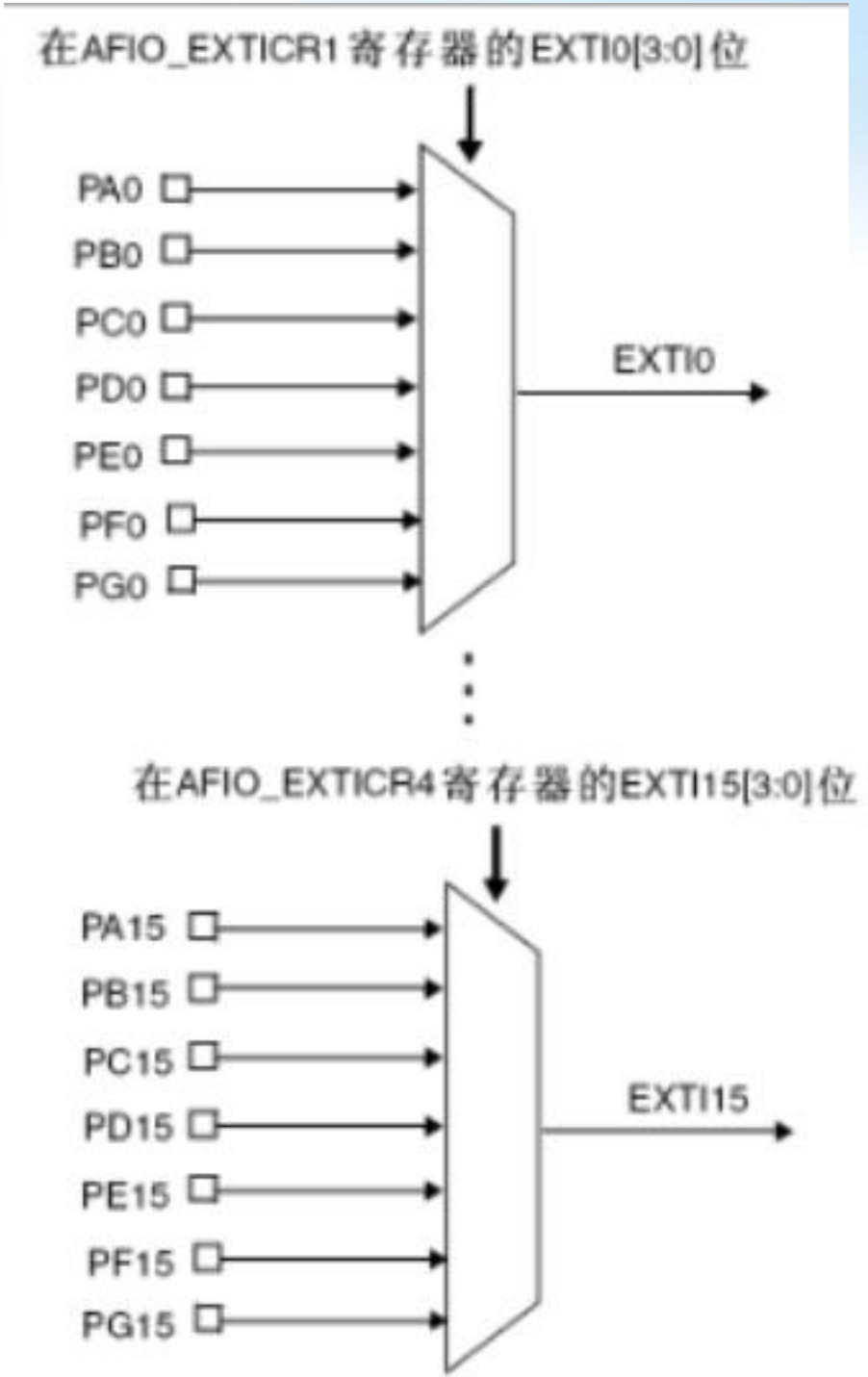
```
NVIC_InitTypeDef     NVIC_InitStructure;
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;//串口 1 中断
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=1 ;//  抢占优先级为 1
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 2;//  子优先级位 2
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;                  //IRQ 通道使能
NVIC_Init(&NVIC_InitStructure);       //根据上面指定的参数初始化 NVIC 寄存器
```

总结一下中断优先级设置的步骤：

- 1. 先设置中断分组 。确定组号，也就是确定抢占优先级和子优先级的分配位数。调用函数为 NVIC_PriorityGroupConfig
- 2. 再设置所用到的中断的中断优先级别。对每个中断调用函数NVIC_Init()

4 、EXTI（External Interrupt）外部中断

- STM32所有I/O端口都可以配置为EXTI外部中断模式，用来捕捉外部信号
  - 用来检测外设中断请求，可以配置为下降沿触发、上升沿触发或上升下降沿触发
- 所有GPIO引脚都可以作为中断源（IO功能强大的表现之一）
- 众多GPIO引脚通过MUX连接到16个EXTI外部中断线上
  - PAx～PGx引脚连接到EXTIx
  - EXTI9～EXTI5共用一个中断向量、被称为EXTI9_5
  - EXTI15～EXTI10共用一个中断向量、被称为EXTI15_10
  - 共用中断向量的中断通道不能嵌套。



GPIO的EXTI连接

- PAx～PGx引脚连接到EXTIx
- 同一个时刻EXTIx只能响应一个端口引脚x的中断事件

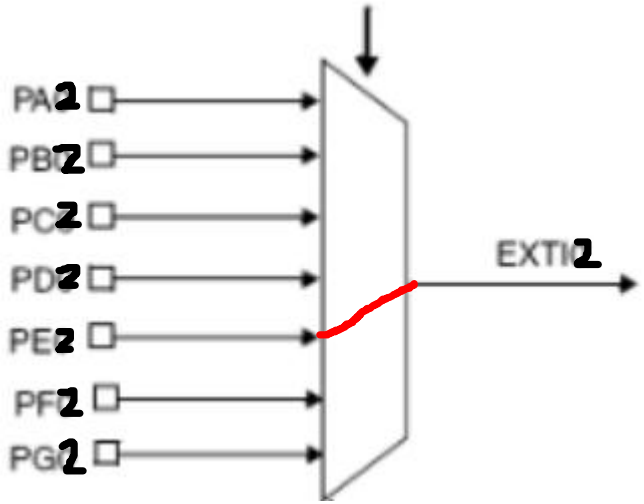
在库函数中，配置GPIO引脚与中断线的映射关系靠函数GPIO_EXTILineConfig() 来实现：

- void GPIO_EXTILineConfig(uint8_t GPIO_PortSourc e, uint8_t GPIO_PinSource);

例如：

- GPIO_EXTILineConfig(GPIO_PortSourceGPIOE,GPIO_PinSource2);

将 GPIOE.2 与 EXTI2 中断线连接起来



EXTI库函数

STM32库在stm32f10x_exti.c中提供了8个EXTI函数

函数名	函数功能
EXTI_Init	EXTI初始化
EXTI_DeInit	EXTI解除初始化
EXTI_StructInit	填充EXTI初始化结构成员
EXTI_GetITStatus	获取中断状态
EXTI_ClearITPendingBit	中断挂起位清除
EXTI_GetFlagStatus	获取标志状态
EXTI_ClearFlag	挂起标志清除
EXTI_GenerateSWIInterrupt	产生软件中断

EXTI中断的初始化是通过函数 EXTI_Init 实现的，其定义为：

```
void EXTI_Init(EXTI_InitTypeDef* EXTI_InitStruct);
```

结构体 EXTI_InitTypeDef 的成员变量：

```
typedef struct{
uint32_t EXTI_Line;
EXTIMode_TypeDef EXTI_Mode;
EXTITrigger_TypeDef EXTI_Trigger;
FunctionalState EXTI_LineCmd;
}EXTI_InitTypeDef;
```

注：

EXTI_Line：中断线的标号，取值范围为EXTI_Line0~EXTI_Line15

EXTI_Mode：中断模式，可选值为 中断 EXTI_Mode_Interrupt 和 事件 EXTI_Mode_Event 。

EXTI_Trigger：触发方式，可以是下降沿触发 EXTI_Trigger_Falling/上升沿触发 EXTI_Trigger_Rising /任意电平 （上升沿和下降沿 触发 EXTI_Trigger_Rising_Falling）

EXTI_LineCmd：允许或禁止

例如：下面的写法设置中断线EXTI4上的中断为下降沿触发。

```
EXTI_InitTypeDef   EXTI_InitStructure;
EXTI_InitStructure.EXTI_Line=EXTI_Line4;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStructure);        //根据 EXTI_InitStruct 中指定的
//参数初始化外设 EXTI 寄存器
```

- 在编写中断服务函数的时候会经常使用到两个函数：

1、ITStatus EXTI_GetITStatus(uint32_t EXTI_Line);

判断某个中断线上的中断是否发生 （标志位是否置位），这个函数一般使用在中断服务函数的开头判断中断是否发生。

2、void EXTI_ClearITPendingBit(uint32_t EXTI_Line);

用来清除某个中断线上的中断标志位，这个函数一般应用在中断服务函数结束之前，清除中断标志位以防止中断阻塞。

- 常见的写法如下：

```
void EXTI3_IRQHandler(void)
{
  if(EXTI_GetITStatus(EXTI_Line3)!=RESET)//判断某个线上的中断是否发生
    {
     中断逻辑…
     EXTI_ClearITPendingBit(EXTI_Line3);  //清除 LINE 上的中断标志位
    }
}
```

固件库还提供了两个函数用来判断外部中断状态以及清除外部 状态标志位的函数：
EXTI_GetFlagStatus 和 EXTI_ClearFlag，
他们的作用和前面两个函数的作用类似。

```
/* Get the status of EXTI line 8 */
FlagStatus EXTIStatus;
EXTIStatus = EXTI_GetFlagStatus(EXTI_Line8);
...
/* Clear the EXTI line 2 pending flag */
EXTI_ClearFlag(EXTI_Line2);
```

注： EXTI_GetITStatus 函数中会先判断这种中断是否使能，使能了才去判断中断标志位，而EXTI_GetFlagStatus 直接用来判断状态标志位。

总结

使用 IO 口外部中断EXTI的一般步骤：

1、初始化 IO 口为输入

2、开 启 AFIO 时钟

3、设置 IO 口与中断线的映射关系

4、初始化线上中断， 设置触发条件 等

5、配置中断分组（ NVIC ），并使能中断

6、编写中断服务函数

6 芯片外设的中断相关函数

函数名	函数功能
PPP_ITConfig	设置外设PPP的某个（些）中断请求是允许或禁止
PPP_GetITStatus	获取外设PPP某个中断状态
PPP_ClearITPendingBit	清除外设PPP某个挂起的中断标志
PPP_GetFlagStatus	获取标志状态：检测某个外设标志是否置位
PPP_ClearFlag	挂起标志清除：清除某个外设的挂起标志

注：PPP指具体的外设，如ADC、DMA、I2C、RCC、RTC、TIM、USART等。

5 EXTI应用示例：按键中断

【例】根据按键状态控制LED灯亮灭

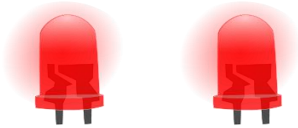
- 硬件连接：

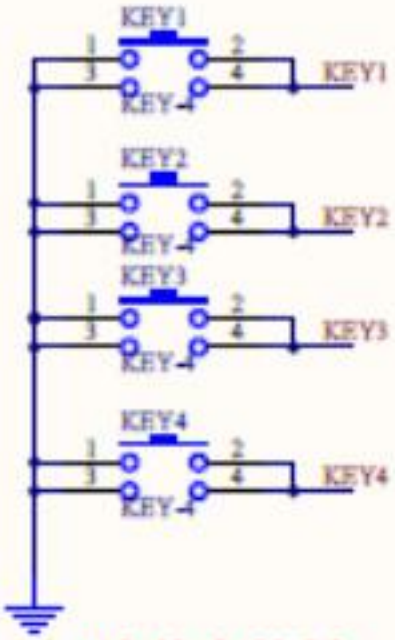
PC3接LED1

PE5接按键Key1

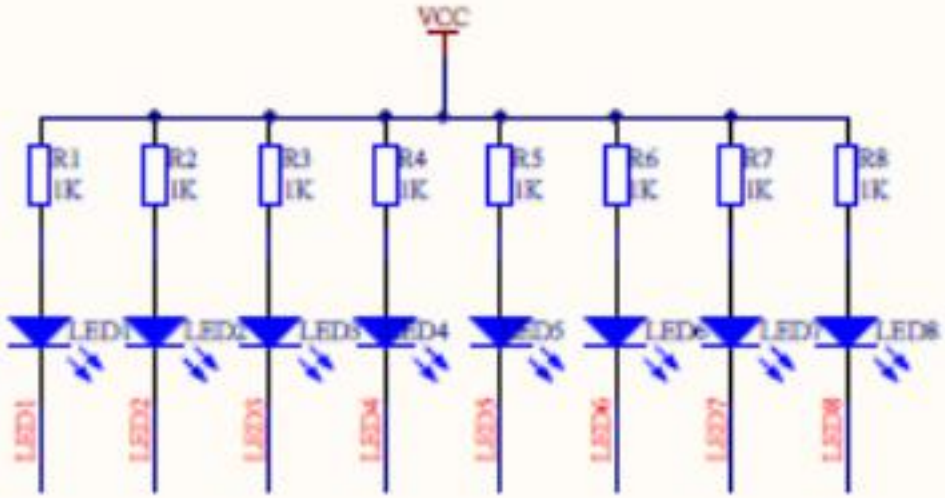
- 程序的功能

PE5配置为EXTI中断模式，key1按下时，进入EXTI中断处理，将led1状态取反





4路独立按键



8路流水灯

Key1通过PE5配置为上拉输入，无按键端口为高电平，按键按下则端口为低电平；

PC3控制共阳接法的LED1，端口输出1则灯灭，端口输出0则灯亮

配置工程环境

- 此程序用到了片上外设GPIO、RCC，还有中断

故工程中需要加入：

stm32f10x_gpio.c

stm32f10x_rcc.c

stm32f10x_exit.c

misc.c

- 相应的，stm32f10x_conf.h中对应的头文件应解除注释

```
26 /* Includes ------------------
27 /* Uncomment/Comment the line be
28 //#include "stm32f10x_adc.h"
29 //#include "stm32f10x_bkp.h"
30 //#include "stm32f10x_can.h"
31 //#include "stm32f10x_cec.h"
32 //#include "stm32f10x_crc.h"
33 //#include "stm32f10x_dac.h"
34 //#include "stm32f10x_dbgmcu.h"
35 //#include "stm32f10x_dma.h"
36 #include "stm32f10x_exti.h"
37 //#include "stm32f10x_flash.h"
38 //#include "stm32f10x_fsmc.h"
39 #include "stm32f10x_gpio.h"
40 //#include "stm32f10x_i2c.h"
41 //#include "stm32f10x_iwdg.h"
42 //#include "stm32f10x_pwr.h"
43 #include "stm32f10x_rcc.h"
44 //#include "stm32f10x_rtc.h"
45 //#include "stm32f10x_sdio.h"
46 //#include "stm32f10x_spi.h"
47 //#include "stm32f10x_tim.h"
48 //#include "stm32f10x_usart.h"
49 //#include "stm32f10x_wwdg.h"
50 #include "misc.h" /* High level
```

主程序流程

```
int main(void)
{ LED_GPIO_Config(); // LED初始化
  LED1(ON);//led1初始状态为点亮状态
  EXTI_PE5_Config(); //外部中断初始化(含KEY初始化)
  while (1) { } // 循环等待中断
}
```

LED_GPIO_Config函数

```
18 /*
19  * 函数名: LED_GPIO_Config
20  * 描述   : 配置LED用到的I/O口
21  * 输入   : 无
22  * 输出   : 无
23  */
24 void LED_GPIO_Config(void)
25 {
26   GPIO_InitTypeDef GPIO_InitStructure;
27   RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOC, ENABLE);
28
29     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 ;
30     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
31     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
32     GPIO_Init(GPIOC, &GPIO_InitStructure);
33
34   GPIO_SetBits(GPIOC, GPIO_Pin_3 );  // turn off all led
35 }
```

- PC3控制LED1（共阳接法）

EXTI_PE5_Config函数

```
40 /*
41  * 函数名: EXTI_PE5_Config
42  * 描述  : 配置 PE5 为线中断口，并设置中断优先级
43  * 输入  : 无
44  * 输出  : 无
45  * 调用  : 外部调用
46  */
47 void EXTI_PE5_Config(void)
48 {
49   GPIO_InitTypeDef GPIO_InitStructure;
50   EXTI_InitTypeDef EXTI_InitStructure;
51
52   /* config the extiline(PE5) clock and AFIO clock */
53   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE | RCC_APB2Periph_AFIO,ENABLE);
54
55   /* EXTI line gpio config(PE5) */
56   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
57   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;  // 上拉输入
58   GPIO_Init(GPIOE, &GPIO_InitStructure);
59
60   /* EXTI line(PE5) mode config */
61   GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource5);
62   EXTI_InitStructure.EXTI_Line = EXTI_Line5;
63   EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
64   EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling; //下降沿中断
65   EXTI_InitStructure.EXTI_LineCmd = ENABLE;
66   EXTI_Init(&EXTI_InitStructure);
67
68   /* config the NVIC(PE5) */
69   NVIC_Configuration();
70 }
```

- PE5接Key1

启动GPIO的复用功能（AFIO）时钟

- STM32的I/O引脚，不仅能够作为通用I/O端口GPIO还可以作为片上外设（如串口、ADC等）的I/O引脚，这称为复用I/O端口AFIO
- 大多数GPIO都有默认的复用功能，部分GPIO还有重映射功能,即把原来是A引脚的默认复用功能，映射到B引脚使用
- 当把GPIO用作EXTI外部中断或者使用重映射功能时，必须在设置GPIO引脚之前启动AFIO时钟

```
/* config the extiline(PE5) clock and AFIO clock */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE | RCC_APB2Periph_AFIO,ENABLE);
```

选择作为EXTI中断的GPIO引脚：如PE5

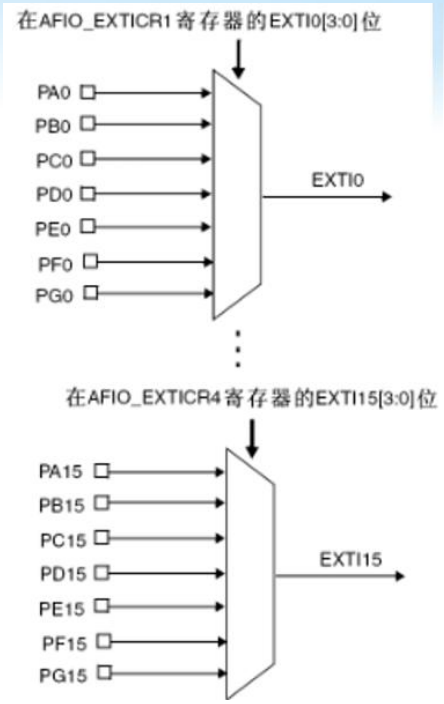
```
void GPIO_EXTILineConfig（
uint8_t GPIO_PortSource,
uint8_t GPIO_PinSource ）
```

- GPIO_PortSource选择作为EXTI线的GPIO端口

  形如GPIO_PortSourceGPIOx（x是A～G）

- GPIO_PinSource指明配置的EXTI线（GPIO引脚）

  形如GPIO_PinSourcex（x是0～15）




```
/* EXTI line(PE5) mode config */
GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource5);
```

EXTI初始化配置

- 调用stm32f10x_exti.c中的EXTI初始化配置

  void EXTI_Init(EXTI_InitTypeDef * EXTI_InitStruct)

  - 参数EXTI_InitStruct是指向EXTI_InitTypeDef结构的指针

```
typedef struct
{
 uint32_t EXTI_Line;                /* 指明EXTI线 */
 EXTIMode_TypeDef EXTI_Mode;        /* 指明工作模式 */
 EXTITrigger_TypeDef EXTI_Trigger; /* 指明有效边沿 */
 FunctionalState EXTI_LineCmd; /* ENABLE或DISABLE */
}EXTI_InitTypeDef;
```

用作EXTI来源的PE5的配置

```
55    /* EXTI line(PE5) mode config */
56    GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource5);
57
58    /* EXTI line gpio config(PE5) */
59    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
60    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;  // 上拉输入
61    GPIO_Init(GPIOE, &GPIO_InitStructure);
62
63    EXTI_InitStructure.EXTI_Line = EXTI_Line5;
64    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
65    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling; //下降沿中断
66    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
67    EXTI_Init(&EXTI_InitStructure);
```

- 注意：上拉输入与下降沿触发相对应

NVIC初始化配置

```
25  static void NVIC_Configuration(void)
26  {
27    NVIC_InitTypeDef NVIC_InitStructure;
28
29    /* Configure one bit for preemption priority */
30    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
31
32    /* 配置P[A|B|C|D|E]5为中断源 */
33    NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
34    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
35    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
36    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
37    NVIC_Init(&NVIC_InitStructure);
38  }
```

- 优先级分组设为1-3组合
- Exit5—exti9线共用一个中断向量exti9_5_IRQn(查阅stm32f10x.h可知)
- 配置抢占优先级和响应优先级，此例中简单地设置为最高优先级。
- 最后打开该中断通道

在stm32f10x_it.c中编写中断服务程序

- 中断服务函数名必须与启动代码中定义的相同

```
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
```

- KEY1（PE5）的中断服务函数

```
139  /* I/O线中断，中断线为PE5 */
140  void EXTI9_5_IRQHandler(void)
141  {
142    if(EXTI_GetITStatus(EXTI_Line5) != RESET) //确保是否产生了EXTI Line中断
143    {
144      // LED1 取反
145      GPIO_WriteBit(GPIOC, GPIO_Pin_3,
146        (BitAction)((1-GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_3))));
147      EXTI_ClearITPendingBit(EXTI_Line5);     //清除中断标志位
148    }
149  }
```

中断状态获取与清除函数

- 中断状态获取函数，用于确认发生了中断

  ITStatus EXTI_GetITStatus (uint32_t EXTI_Line)

  - 参数EXTI_Line是EXTI线编号：
    EXTI_Linex（x是0～19）
  - 返回值ITStatus是EXTI线的状态：
    触发（SET）、没有触发（RESET）

- 中断状态清除函数，用于清除EXTI线挂起的位

  void EXTI_ClearITPendingBit (uint32_t EXTI_Line)

挂起（Pending）是暂停中断处理
挂起状态表明中断还没有处理结束

总结使用 IO 口外部中断的一般步骤

- 复位后，所有中断都被禁止，并赋予优先级0
- 使用任何中断，都需要设置中断，具体（从外向内）步骤：
  - 1）初始化 IO 口为输入。
  - 2）开启 AFIO 时钟
  - 3）设置 IO 口与中断线的映射关系。
  - 4）初始化线上中断，设置触发条件等。
  - 5）配置NVIC中断组，并使能中断。
  - 6）编写中断服务函数。