IO引脚复用和映射

参考资料：

- 探索者STM32F4开发板：

  《STM32F4开发指南-库函数版本》-4.4 IO引脚复用和映射

- STM32F4xx官方资料：

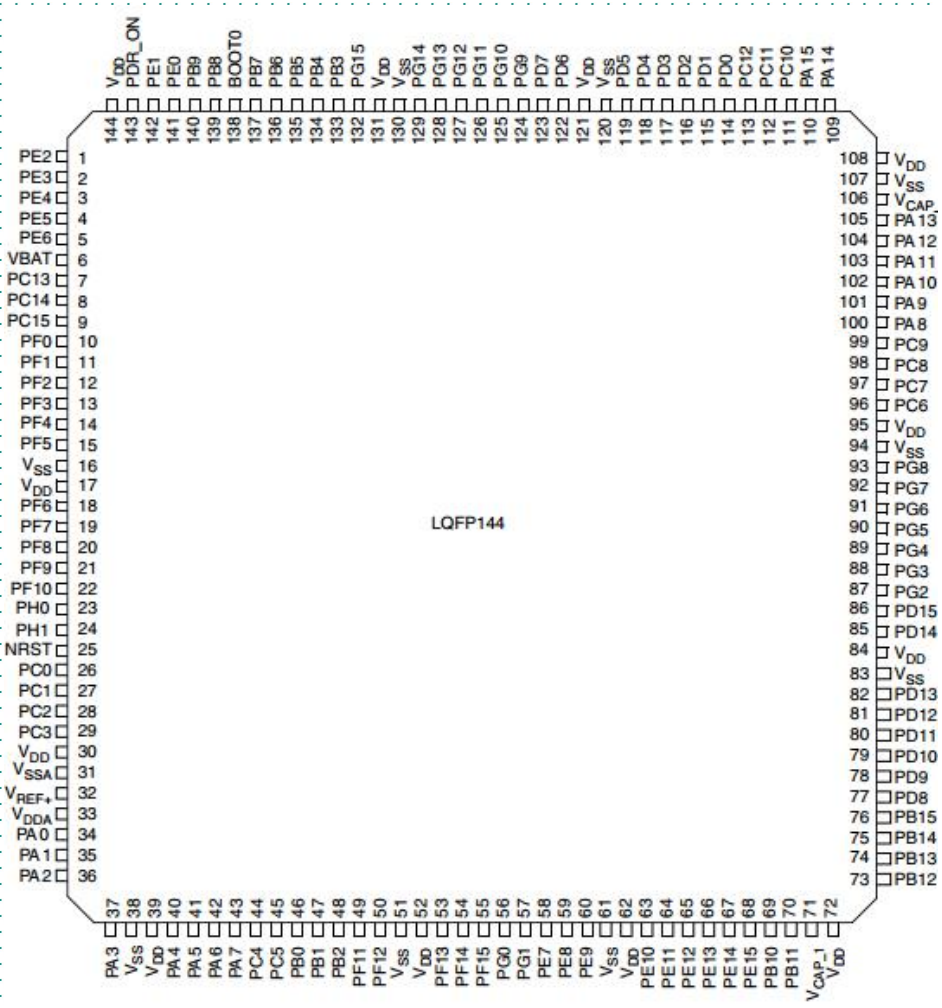
  《STM32F4xx中文参考手册》-第7章通用IO

✓ 1. 端口复用

◆ 什么是端口复用？

STM32有很多的内置外设，这些外设的外部引脚都是与GPIO复用的。也就是说，一个GPIO如果可以复用为内置外设的功能引脚，那么当这个GPIO作为内置外设使用的时候，就叫做复用。

✓ 1. 端口复用


LQFP144
PE2 1
PE3 2
PE4 3
PE5 4
PE6 5
VBAT 6
PC13 7
PC14 8
PC15 9
PF0 10
PF1 11
PF2 12
PF3 13
PF4 14
PF5 15
VSS 16
VDD 17
PF6 18
PF7 19
PF8 20
PF9 21
PF10 22
PH0 23
PH1 24
NRST 25
PC0 26
PC1 27
PC2 28
PC3 29
VDD 30
VSSA 31
VREF+ 32
VDDA 33
PA0 34
PA1 35
PA2 36
PA3 37
VSS 38
VDD 39
PA4 40
PA5 41
PA6 42
PA7 43
PC4 44
PC5 45
PB0 46
PB1 47
PB2 48
PF11 49
PF12 50
VSS 51
VDD 52
PF13 53
PF14 54
PF15 55
PG0 56
PG1 57
PE7 58
PE8 59
PE9 60
VSS 61
VDD 62
PE10 63
PE11 64
PE12 65
PE13 66
PE14 67
PE15 68
PB10 69
PB11 70
VCAP_1 71
VDD 72
73 PB12
74 PB13
75 PB14
76 PB15
77 PD8
78 PD9
79 PD10
80 PD11
81 PD12
82 PD13
83 VSS
84 VDD
85 PD14
86 PD15
87 PG2
88 PG3
89 PG4
90 PG5
91 PG6
92 PG7
93 PG8
94 VSS
95 VDD
96 PC6
97 PC7
98 PC8
99 PC9
100 PA8
101 PA9
102 PA10
103 PA11
104 PA12
105 PA13
106 VCAP_
107 VSS
108 VDD
109 PA14
110 PA15
111 PC10
112 PC11
113 PC12
114 PD0
115 PD1
116 PD2
117 PD3
118 PD4
119 PD5
120 VSS
121 VDD
122 PD6
123 PD7
124 PG9
125 PG10
126 PG11
127 PG12
128 PG13
129 PG14
130 VSS
131 VDD
132 PG15
133 PB3
134 PB4
135 PB5
136 PB6
137 PB7
138 BOOT0
139 PB8
140 PB9
141 PE0
142 PE1
143 PDR_ON
144 VDD

✓ 1. 端口复用

◆ 例如串口1 的发送接收引脚是**PA9,PA10**，当我们把**PA9,PA10**不用作**GPIO**，而用做复用功能串口**1**的发送接收引脚的时候，叫端口复用。

Pin number					Pin name (function after reset)(1)	Pin type	I / O structure	Notes	Alternate functions	Additional functions
LQFP64	LQFP100	LQFP144	UFBGA176	LQFP176						
41	67	100	F15	119	PA8	I/O	FT		MCO1 / USART1_CK/ TIM1_CH1/ I2C3_SCL/ OTG_FS_SOF/ EVENTOUT	
42	68	101	E15	120	PA9	I/O	FT		USART1_TX/ TIM1_CH2 / I2C3_SMBA / DCMI_D0/ EVENTOUT	OTG_FS_VBUS
43	69	102	D15	121	PA10	I/O	FT		USART1_RX/ TIM1_CH3/ OTG_FS_ID/DCMI_D1/ EVENTOUT	

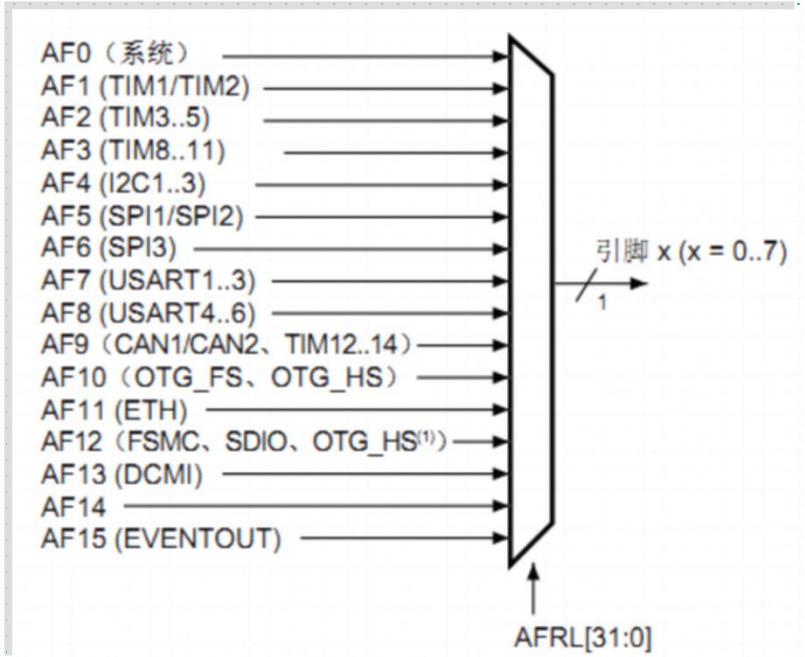
✓ 1. 端口复用

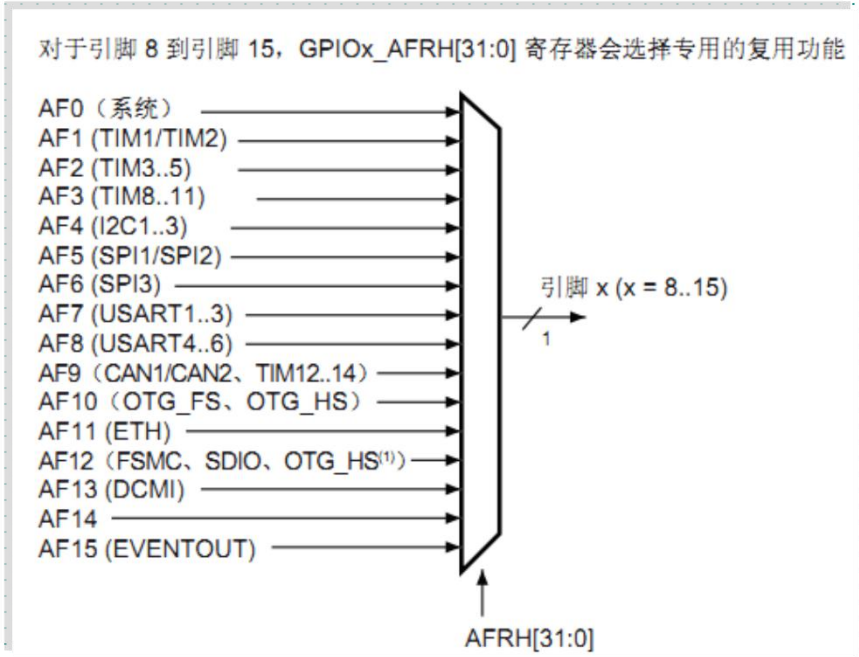
◆ STM32F4的端口复用映射原理

- ✓ STM32F4系列微控制器IO引脚通过一个复用器连接到内置外设或模块。该复用器一次只允许一个外设的复用功能（AF）连接到对应的IO口。这样可以确保共用同一个IO引脚的外设之间不会发生冲突。
- ✓ 每个IO引脚都有一个复用器，该复用器采用16路复用功能输入（AF0到AF15），可通过GPIOx_AFRL(针对引脚0-7)和GPIOx_AFRH（针对引脚8-15）寄存器对这些输入进行配置，每四位控制一路复用。

✓ 1. 端口复用

■ 端口复用映射示意图





✓ 1. 端口复用

■AFRL寄存器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:0 **AFRLy**：端口 x 位 y 的复用功能选择 (Alternate function selection for port x bit y) (y = 0..7)
这些位通过软件写入，用于配置复用功能 I/O。

AFRLy 选择：

0000：AF0
0001：AF1
0010：AF2
0011：AF3
0100：AF4
0101：AF5
0110：AF6
0111：AF7
1000：AF8
1001：AF9
1010：AF10
1011：AF11
1100：AF12
1101：AF13
1110：AF14
1111：AF15

✓ 1. 端口复用

■ 复用功能映射配置

1. **系统功能**

   将 I/O 连接到 AF0，然后根据所用功能进行配置：

   - JTAG/SWD：在各器件复位后，会将这些引脚指定为专用引脚，可供片上调试模块立即使用（不受 GPIO 控制器控制）。
   - RTC_REFIN：此引脚应配置为输入浮空模式。
   - MCO1 和 MCO2：这些引脚必须配置为复用功能模式。

2. **GPIO**

   在 GPIOx_MODER 寄存器中将所需 I/O 配置为输出或输入。

✓ 1. 端口复用

3. 外设复用功能

对于 ADC 和 DAC，在 GPIOx_MODER 寄存器中将所需 I/O 配置为模拟通道。

对于其它外设：

- 在 GPIOx_MODER 寄存器中将所需 I/O 配置为复用功能
- 通过 GPIOx_OTYPER、GPIOx_PUPDR 和 GPIOx_OSPEEDER 寄存器，分别选择类型、上拉/下拉以及输出速度
- 在 GPIOx_AFRL 或 GPIOx_AFRH 寄存器中，将 I/O 连接到所需 AFx

✓ 1. 端口复用配置过程

◆端口复用为复用功能配置过程

-以PA9,PA10配置为串口1为例

①GPIO端口时钟使能。

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE);

②复用外设时钟使能。

比如你要将端口PA9,PA10复用为串口，所以要使能串口时钟。

RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);

③端口模式配置为复用功能。 GPIO_Init（）函数。

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;//复用功能

✓ 1. 端口复用配置过程

④配置GPIOx_AFRL或者GPIOx_AFRH寄存器，将IO连接到所需的AFx。

```
/*PA9连接AF7，复用为USART1_TX */
GPIO_PinAFConfig(GPIOA,GPIO_PinSource9,GPIO_AF_USART1);
/* PA10连接AF7,复用为USART1_RX*/
GPIO_PinAFConfig(GPIOA,GPIO_PinSource10,GPIO_AF_USART1);
```

✓ 1. 端口复用配置过程

◆PA9,PA10复用为串口1的配置过程

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE); //使能GPIOA时钟 ①
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);//使能USART1时钟 ②

//USART1端口配置③
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10; //GPIOA9与GPIOA10
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;//复用功能
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;          //速度50MHz
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; //推挽复用输出
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP; //上拉
GPIO_Init(GPIOA,&GPIO_InitStructure); //初始化PA9，PA10

//串口1对应引脚复用映射 ④
GPIO_PinAFConfig(GPIOA,GPIO_PinSource9,GPIO_AF_USART1); //GPIOA9复用为USART1
GPIO_PinAFConfig(GPIOA,GPIO_PinSource10,GPIO_AF_USART1); //GPIOA10复用为USART1
```

目录

✓ 1. NVIC中断优先级分组

■ 参考资料：

● 探索者STM32F4开发板：

《STM32F4开发指南-库函数版本》-4.5小节 中断优先级分组管理

□ STM32F4xx官方资料：

《STM32F4xx中文参考手册》-第10章 中断和事件

✓ 1. NVIC中断优先级分组

- CM4内核支持256个中断，其中包含了16个内核中断和240个外部中断，并且具有256级的可编程中断设置。
- STM32F4并没有使用CM4内核的全部东西，而是只用了它的一部分。
  -STM32F40xx/STM32F41xx总共有92个中断。
  -STM32F42xx/STM32F43xx则总共有96个中断
- STM32F40xx/STM32F41xx的92个中断里面，包括10个内核中断和82个可屏蔽中断，具有16级可编程的中断优先级，而我们常用的就是这82个可屏蔽中断。

✓ 1. NVIC中断优先级分组

《STM32F4xx中文参考手册》P234 表45和46

■ **STM32F405xx/STM32F407XX向量表**

10个内核中断

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) 连接到 NMI 向量。	0x0000 0008
	-1	固定	HardFault	所有类型的错误	0x0000 000C
	0	可设置	MemManage	存储器管理	0x0000 0010
	1	可设置	BusFault	预取指失败，存储器访问失败	0x0000 0014
	2	可设置	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 指令调用的系统服务	0x0000 002C
	4	可设置	Debug Monitor	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

82个可屏蔽中断

0	7	可设置	WWDG	窗口看门狗中断	0x0000 0040
1	8	可设置	PVD	连接到 EXTI 线的可编程电压检测 (PVD) 中断	0x0000 0044
2	9	可设置	TAMP_STAMP	连接到 EXTI 线的入侵和时间戳中断	0x0000 0048
3	10	可设置	RTC_WKUP	连接到 EXTI 线的 RTC 唤醒中断	0x0000 004C
4	11	可设置	FLASH	Flash 全局中断	0x0000 0050
5	12	可设置	RCC	RCC 全局中断	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_Stream0	DMA1 流 0 全局中断	0x0000 006C

▪ ▪ ▪ ▪ ▪ ▪

75	82	可设置	OTG_HS_EP1_IN	USB On The Go HS 端点 1 输入全局中断	0x0000 016C
76	83	可设置	OTG_HS_WKUP	连接到 EXTI 线的 USB On The Go HS 唤醒中断	0x0000 0170
77	84	可设置	OTG_HS	USB On The Go HS 全局中断	0x0000 0174
78	85	可设置	DCMI	DCMI 全局中断	0x0000 0178
79	86	可设置	CRYP	CRYP 加密全局中断	0x0000 017C
80	87	可设置	HASH_RNG	哈希和随机数发生器全局中断	0x0000 0180
81	88	可设置	FPU	FPU 全局中断	0x0000 0184

✓ 1. NVIC中断优先级分组

几十个中断，怎么管理？

✓ 1. NVIC中断优先级分组

◆ 中断管理方法：

首先，对**STM32**中断进行分组，组**0~4**。同时，对每个中断设置一个抢占优先级和一个响应优先级值。

分组配置是在寄存器SCB->AIRCR中配置：

组	AIRCR[10：8]	IP bit[7：4]分配情况	分配结果
0	111	0：4	0位抢占优先级，4位响应优先级
1	110	1：3	1位抢占优先级，3位响应优先级
2	101	2：2	2位抢占优先级，2位响应优先级
3	100	3：1	3位抢占优先级，1位响应优先级
4	011	4：0	4位抢占优先级，0位响应优先级

✓ 1. NVIC中断优先级分组

◆ 抢占优先级 & 响应优先级区别：

- 高优先级的抢占优先级是可以打断正在进行的低抢占优先级中断的。
- 抢占优先级相同的中断，高响应优先级不可以打断低响应优先级的中断。
- 抢占优先级相同的中断，当两个中断同时发生的情况下，哪个响应优先级高，哪个先执行。
- 如果两个中断的抢占优先级和响应优先级都是一样的话，则看哪个中断先发生就先执行；

✓ 1. NVIC中断优先级分组

◆ 举例：

- 假定设置中断优先级组为2，然后设置中断3(RTC中断)的抢占优先级为2，响应优先级为1。中断6（外部中断0）的抢占优先级为3，响应优先级为0。中断7（外部中断1）的抢占优先级为2，响应优先级为0。

那么这3个中断的优先级顺序为：中断7>中断3>中断6。

✓ 1. NVIC中断优先级分组

◆ 特别说明：

一般情况下，系统代码执行过程中，只设置一次中断优先级分组，比如分组2，设置好分组之后一般不会再改变分组。随意改变分组会导致中断管理混乱，程序出现意想不到的执行结果。

✓ 1. NVIC中断优先级分组

◆ 中断优先级分组函数：

void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup);

```
void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup)
{
  assert_param(IS_NVIC_PRIORITY_GROUP(NVIC_PriorityGroup));
  SCB->AIRCR = AIRCR_VECTKEY_MASK | NVIC_PriorityGroup;
}
```

```
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
```

✓ 2. 中断优先级设置

分组设置好之后，怎么设置单个中断的抢占优先级和响应优先级？

✓ 2. 中断优先级设置

◆ 中断设置相关寄存器

__IO uint8_t IP[240]; //中断优先级控制的寄存器组

__IO uint32_t ISER[8]; //中断使能寄存器组
__IO uint32_t ICER[8]; //中断失能寄存器组
__IO uint32_t ISPR[8]; //中断挂起寄存器组
__IO uint32_t ICPR[8]; //中断解挂寄存器组
__IO uint32_t IABR[8]; //中断激活标志位寄存器组

✓ 2. 中断优先级设置

◆ MDK中NVIC寄存器结构体

```
typedef struct
{
  __IO uint32_t ISER[8];
       uint32_t RESERVED0[24];
  __IO uint32_t ICER[8];
       uint32_t RSERVED1[24];
  __IO uint32_t ISPR[8];
       uint32_t RESERVED2[24];
  __IO uint32_t ICPR[8];
       uint32_t RESERVED3[24];
  __IO uint32_t IABR[8];
       uint32_t RESERVED4[56];
  __IO uint8_t  IP[240];
       uint32_t RESERVED5[644];
  __O  uint32_t STIR;
}  NVIC_Type;
```

✓ 2. 中断优先级设置

◆对于每个中断怎么设置优先级？

中断优先级控制的寄存器组：IP[240]
全称是：Interrupt Priority Registers

240个8位寄存器，每个中断使用一个寄存器来确定优先级。
STM32F40x系列一共82个可屏蔽中断，使用IP[81]~IP[0]。

每个IP寄存器的高4位用来设置抢占和响应优先级（根据分组），低4位没有用到。

void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct);

✓ 2. 中断优先级设置

◆ 中断使能寄存器组：ISER[8]

作用：用来使能中断

32位寄存器，每个位控制一个中断的使能。STM32F40x只有82个可屏蔽中断，所以只使用了其中的ISER[0]~ISER[2]。

ISER[0]的bit0~bit31分别对应中断0~31。ISER[1]的bit0~27对应中断32~63；ISER[2]的bit0~17对应中断64~81；

void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct);

✓ 2. 中断优先级设置

◆ 中断失能寄存器组：ICER[8]

作用：用来失能中断

32位寄存器，每个位控制一个中断的失能。STM32F40x只有82个可屏蔽中断，所以只使用了其中的ICER[0]和ICER[1]。

ICER[0]的bit0~bit31分别对应中断0~31。ICER[1]的bit0~27对应中断32~63； ICER[3]的bit0~17对应中断64~82；

配置方法跟ISER一样。

void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct);

✓ 2. 中断优先级设置

- 中断挂起控制寄存器组：ISPR[8]

  作用：用来挂起中断

- 中断解挂控制寄存器组：ICPR[8]

  作用：用来解挂中断

```
static __INLINE void NVIC_SetPendingIRQ(IRQn_Type IRQn);
static __INLINE uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn);
static __INLINE void NVIC_ClearPendingIRQ(IRQn_Type IRQn)
```

✓ 2. 中断优先级设置

- 中断激活标志位寄存器组：IABR [8]

  作用：只读，通过它可以知道当前在执行的中断是哪一个

  如果对应位为1，说明该中断正在执行。

  static __INLINE uint32_t NVIC_GetActive(IRQn_Type IRQn)

✓ 2. 中断优先级设置

◆ 中断参数初始化函数

void NVIC_Init(NVIC_InitTypeDef NVIC_InitStruct);*

```
typedef struct
{
  uint8_t NVIC_IRQChannel; //设置中断通道
  uint8_t NVIC_IRQChannelPreemptionPriority;//设置响应优先级
  uint8_t NVIC_IRQChannelSubPriority; //设置抢占优先级
  FunctionalState NVIC_IRQChannelCmd; //使能/使能
} NVIC_InitTypeDef;
```

```
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
```

✓ 3. NVIC总结

◆ 中断优先级设置步骤

①系统运行后先设置中断优先级分组。调用函数：

void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup);

整个系统执行过程中，只设置一次中断分组。

②针对每个中断，设置对应的抢占优先级和响应优先级：

void NVIC_Init(NVIC_InitTypeDef NVIC_InitStruct);*

③ 如果需要挂起/解挂，查看中断当前激活状态，分别调用相关函数即可。