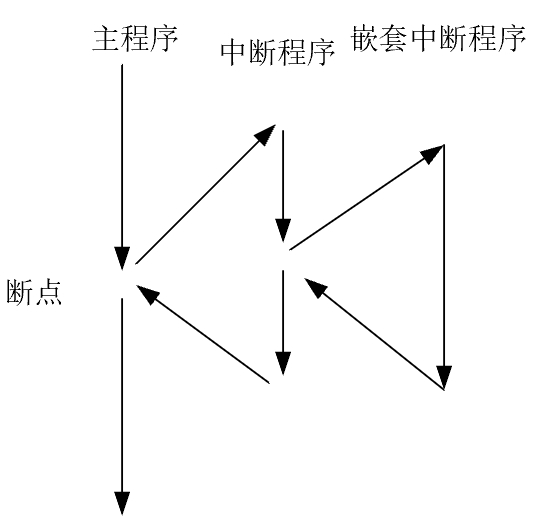
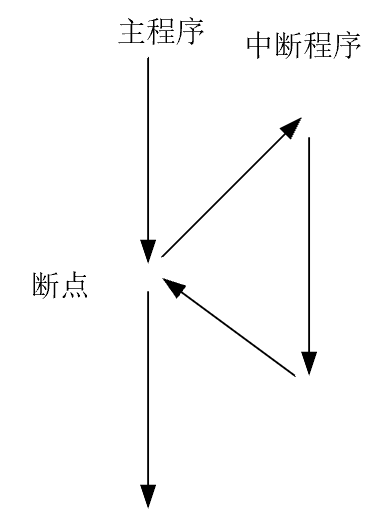
中断：在主程序运行过程中，出现了特定的中断触发条件（中断源），使得CPU暂停当前正在运行的程序，转而去处理中断程序，处理完成后又返回原来被暂停的位置继续运行

中断优先级：当有多个中断源同时申请中断时，CPU会根据中断源的轻重缓急进行裁决，优先响应更加紧急的中断源

中断嵌套：当一个中断程序正在运行时，又有新的更高优先级的中断源申请中断，CPU再次暂停当前中断程序，转而去处理新的中断程序，处理完成后依次进行返回

中断执行流程：



**STM32中断**

68个可屏蔽中断通道，包含EXTI,TIM,ADC,USART,SPI,12C,RTC等多个外设。

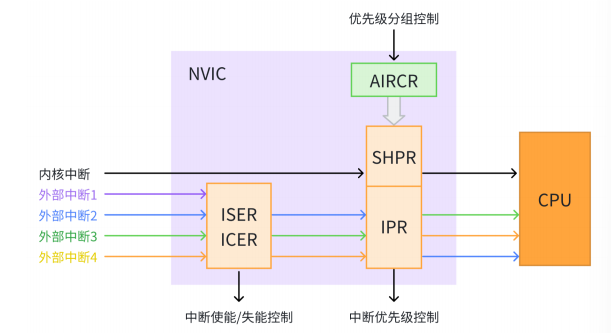
使用NVIC统一管理中断，每个中断通道都拥有16个可编程的优先等级，可对优先级进行分组，进一步设置抢占优先级和响应优先级。

NVIC，即Nested Vectored Interrupt Controller（嵌套向量中断控制器），是STM32中的中断控制器。它负责管理和协调处理器的中断请求，是STM32中处理异步事件的重要机制。

NVIC提供了灵活、高效、可扩展的中断处理机制，支持多级优先级、多向中断、嵌套向量中断等特性。当一个中断请求到达时，NVIC会确定其优先级并决定是否应该中断当前执行的程序，以便及时响应和处理该中断请求。这种设计有助于提高系统的响应速度和可靠性，特别是在需要处理大量中断请求的实时应用程序中。

NVIC 支持：256个中断（16内核+240外部），支持：256个优先级，允许裁剪。

NVIC工作原理



NVIC寄存器



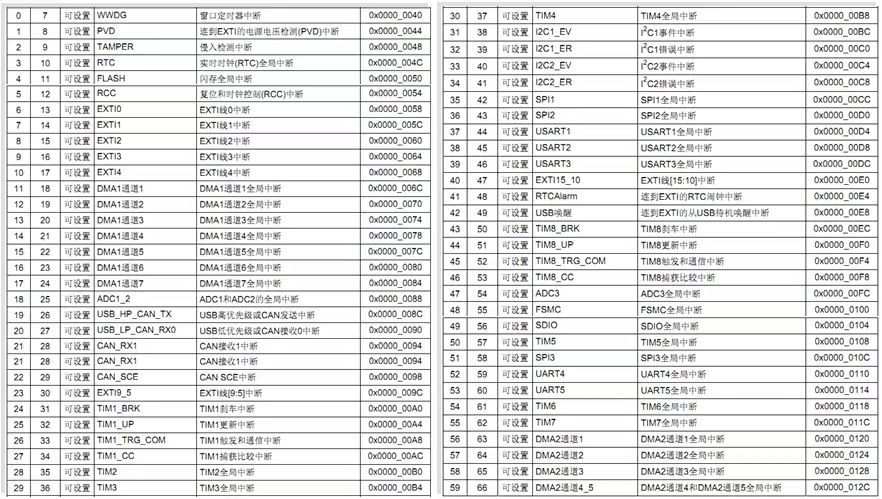
中断向量表

STM32的中断向量表是一个存储中断处理函数地址的数组，位于Flash区的起始位置。每个数组元素对应一个中断源，其地址指向相应的中断服务程序。当中断发生时，处理器会根据中断号查找向量表，然后跳转到对应的中断服务程序执行。

中断向量表的主要作用是解决中断函数地址不固定与中断必须跳转到固定地方执行程序之间的矛盾。由于编译器每次编译都会为中断函数随机分配地址，但硬件要求中断必须跳转到固定的位置，因此，中断向量表就作为这样一个固定的地址列表，其中包含了中断函数的地址以及跳转到这些地址的程序。当中断发生时，处理器会跳转到这个固定的中断向量表，然后根据其中的信息跳转到相应的中断处理函数从而执行中断。

STM32F10xxx产品的向量表





中断优先级基本概念

NVIC可以管理多个中断请求，并按优先级处理它们。在STM32中，中断优先级被划分为抢占式优先级和响应 优先级，可以根据具体的应用需求进行配置。不同的优先级分组方式会影响中断的响应和处理顺序。

抢占优先级

如果一个中断的抢占优先级高于当前正在执行的中断，那么它可以打断当前中断，优先得到执行。数值越小，优先级越高。

响应优先级

如果两个中断同时到达，且它们的抢占优先级相同，那么响应优先级高的中断将首先得到响应。数值越小，优先级越高。

自然优先级

自然优先级是由硬件固定并预先设定的，用户无法更改。当抢占优先级和响应优先级都相同时，自然优先级将决定哪个中断先得到处理。

优先级执行顺序

当多个中断同时发生时，执行顺序首先由抢占优先级决定。如果抢占优先级相同，则进一步由响应优先级决。如果响应优先级也相同，则最终由自然优先级决定。

在中断嵌套的情况下，高抢占优先级的中断可以打断低抢占优先级的中断，但高响应优先级的中断不能打断低响应优先级的中断（当它们具有相同的抢占优先时）。

优先级分组

优先级寄存器IPR有8位，但实际只使用到高4位，用于决定抢占优先级、响应优先级的等级。具体这4位如何切割由AIRCR寄存器控制。



抢占优先级和响应优先级均相同的按照中断号来排队，数值小的优先排队（中断号为向量表中优先级那一列）。

**EXTI（外部中断）**

EXTI 是 External Interrupt 的缩写，表示外部中断事件控制器。EXTI 可以监测指定 GPIO 口的电平信号变化，并在检测到指定条件时，向内核的中断控制器 NVIC 发出中断申请。NVIC 在裁决后，如果满足条件，会中断CPU的主程序，使 CPU 转而执行EXTI 对应的中断服务程序。

支持的触发方式：上升沿、下降沿、双边沿或软件触发。

支持的IO口：所有的GPIO口，但相同的Pin不能同时触发中断。例如，PA0 和 PB0 不能同时被配置为中断源。

通道数：16个GPIO\_Pin，外加PVD输出，RTC闹钟、USB唤醒、以太网唤醒。

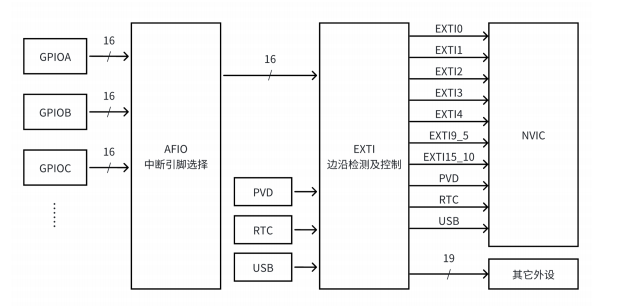
触发响应方式：中断响应/事件响应。

中断/事件

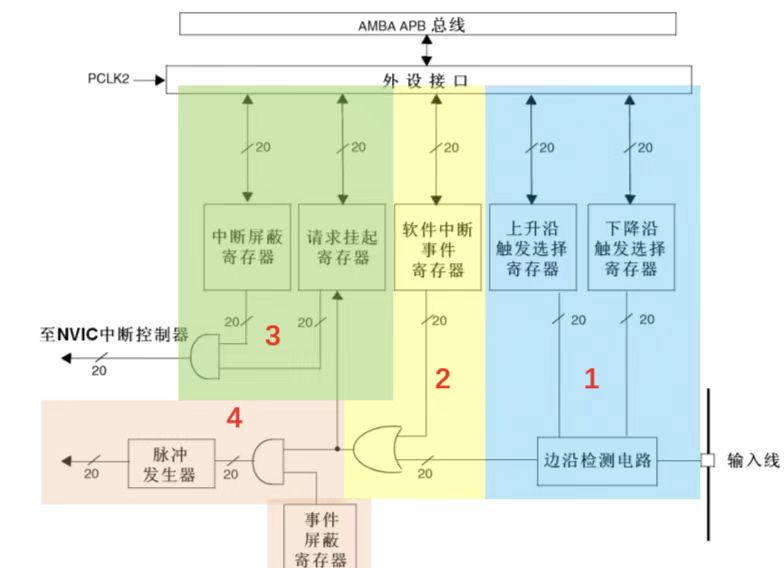
中断会打断CPU当前正在执行的程序，转而去执行中断服务程序，待中断服务程序执行完毕后，CPU会返回到原来的程序执行点继续执行。

事件只是简单地表示某个动作或状态的变化，而不会打断CPU当前正在执行的程序。当事件发生时，它会根据配置来决定是否触发相应的中断。如果开放了对应的中断屏蔽位，事件就可以触发相应的中断，否则事件只会作为一个信号存在，不会被CPU处理。

EXTI基本结构



EXTI基本框图



其中按照控制功能划分，共分为4个部分；有两条主线，一条是由输入线到 NVIC 中断控制器（123），一条是由输入线到脉冲发生器（124）。

输入线：是线路的信息输入端，它可以通过配置寄存器设置为任何一个 GPIO 口，或者是一些外设的事件。输入线一般都是存在电平变化的信号。

1. 边沿检测电路：上升沿触发选择寄存器和下降沿触发选择寄存器。边沿检测电路以输入线作为信号输入端，如果检测到有边沿跳变就输出有效信号‘1’，就输出有效信号‘1’到标号2部分电路，否则输入无效信号‘0’。边沿跳变的标准在于对两个触发选择寄存器的设置。

2、或门电路：它的两个信号输入端分别是软件中断事件寄存器和边沿检测电路的输入信号。或门电路只要输入端有信号‘1’，就会输出‘1’，所以就会输出 ‘1’到标号3电路和标号4电路。

3、与门电路(标号3):两个信号输入端分别是中断屏蔽寄存器和标号2电路信号。如果中断屏蔽寄存器设置为 0 时，不管从标号2电路输出的信号特性如何，最终标号3电路输出的信号都是 0；假如中断屏蔽寄存器设置为 1 时，最终标号3电路输出的信号才由标号2电路输出信号决定，这样子就可以简单控制 中断屏蔽寄存器 来实现中断的目的。标号4电路输出 ‘1’就会把请求挂起寄存器(EXTI\_PR)对应位置 1。

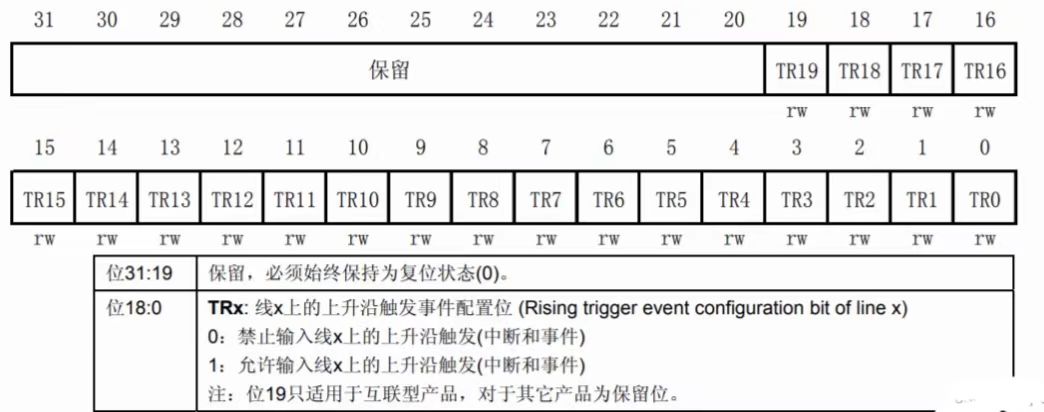
1. 与门电路(标号4)：输入端来自标号2电路以及来自于事件屏蔽寄存器。可以简单的控制事件屏蔽寄存器来实现是否产生事件的目的。标号4电路输出有效信号 1 就会使脉冲发生器电路产生一个脉冲，而无效信号就不会使其产生脉冲信号。脉冲信号产生可以给其他外设电路使用，例如定时器，模拟数字转换器等，这样的脉冲信号一般用来触发 TIM 或者 ADC 开始转换。

产生中断线路目的使把输入信号输入到 NVIC，进一步运行中断服务函数，实现功能。

### EXTI相关寄存器

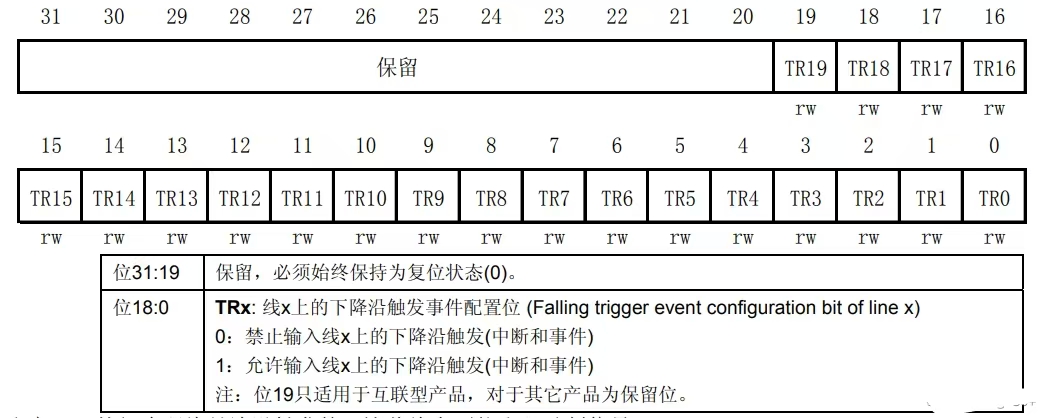
在上述工作原理图中，所涉及的寄存器共有7个，但是对于外部中断来说，我们只需要掌握其中四个寄存器：上升沿触发选择寄存器(EXTIL\_RTSR)、下降沿触发选择寄存器(EXTIL\_FTSR)、挂起寄存器(EXTI\_PR)与中断屏蔽寄存器(EXTI\_IMR)。

1. 上升沿触发选择寄存器（EXTI\_RTSR)



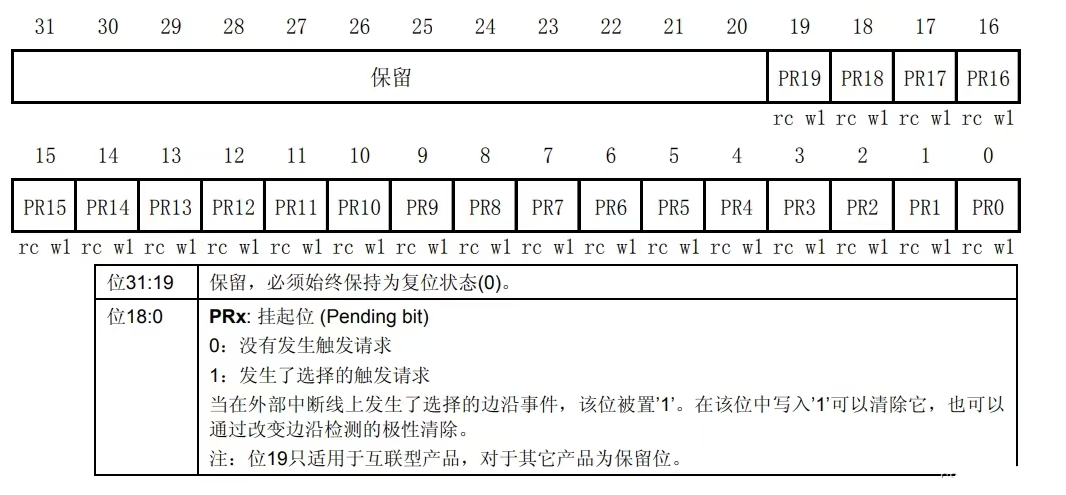
该寄存器主要用于控制输入线进来的输入信号，上升沿时是否在边沿检测电路被检测出，20位共控制20条EXTI线。

1. 下降沿触发选择寄存器（EXTI\_FTSR)



该寄存器主要用于控制输入线进来的输入信号，下降沿时是否在边沿检测电路被检测出，20位共控制20条EXTI线。

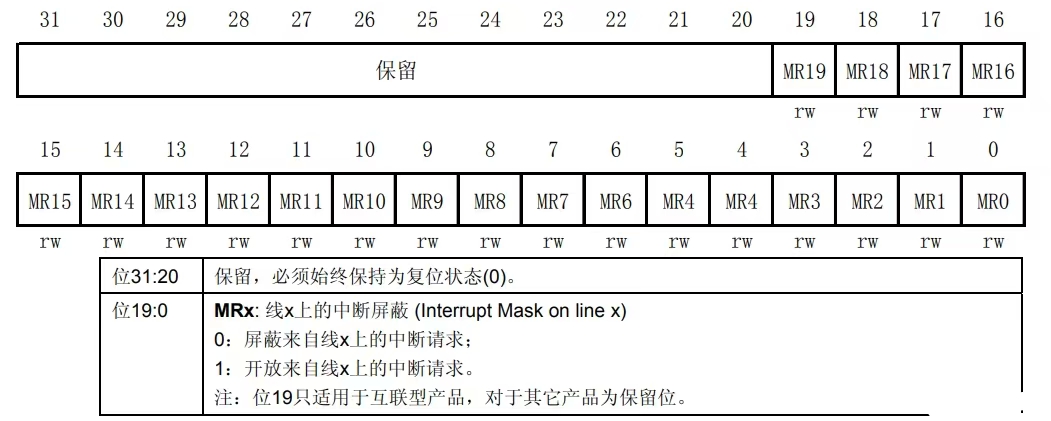
1. 挂起寄存器（EXTI\_PR)



该寄存器的作用主要有两个：

1. 检测外部中断线上是否发生了选择的边沿事件，如果发生了，该位置1，并将信号传递给与门电路，进而进入NVIC中；
2. 在该位手动（软件）写入1，可以清除之前中断信号1，主要作用是进入中断后，清除中断位，防止多次进入中断。

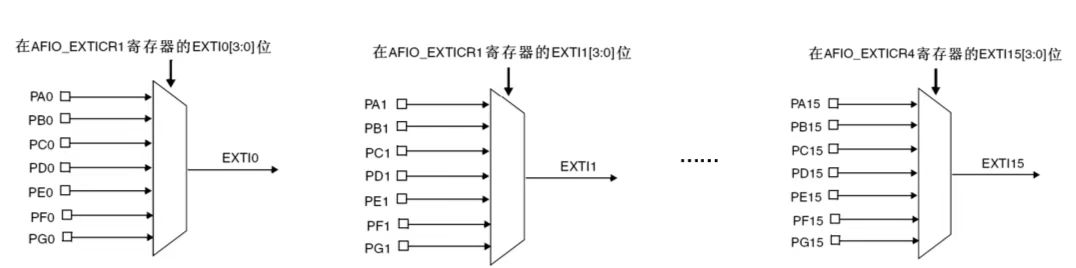
4.中断屏蔽寄存器（EXTI\_IMR)



该寄存器的主要作用只有一个，就是是否允许来自中断线上的中断信号进入NVIC中断控制器。

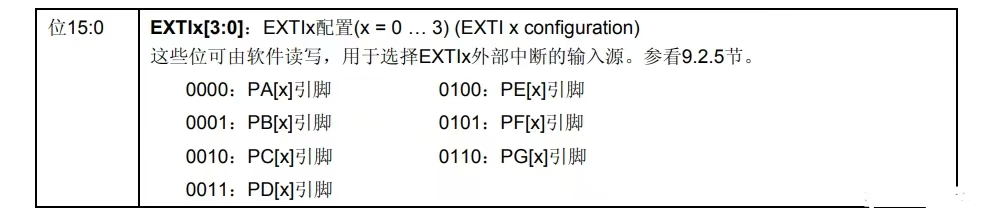
EXIT线与IO映射关系

EXTI 线 0~15：对应外部 IO 口的输入中断。但是STM32F1 供给 IO 口使用的中断线只有 16 个，但是 STM32F1 的 IO 口 却远远不止 16 个，所以 STM32 把 GPIO 管脚 GPIOx.0~GPIOx.15(x=A,B,C,D,E,F,G)分别对应中断线 0~15。这样子每个中断线对应了最多 9 个 IO 口，以线 0 为例：它对应了GPIOA.0、GPIOB.0、 GPIOC.0、GPIOD.0、GPIOE.0、GPIOF.0 和 GPIOG.0。而中断线每次只能连接到 1 个 IO 口上， 这样就需要通过配置决定对应的中断线配置到哪个 GPIO 上了



GPIO 和中断线映射关系是在寄存器 AFIO\_EXTICR1 ~ AFIO\_EXTICR4 中配置的

外部中断配置寄存器1(AFIO\_EXTICR1)



AFIO\_EXTICR1 寄存器配置 EXTI0 到 EXTI3 线，包含的外部中断的引脚包括 PAx 到 PGx， x=0 到 3。

AFIO\_EXTICR2 寄存器配置 EXTI4 到 EXTI7 线；

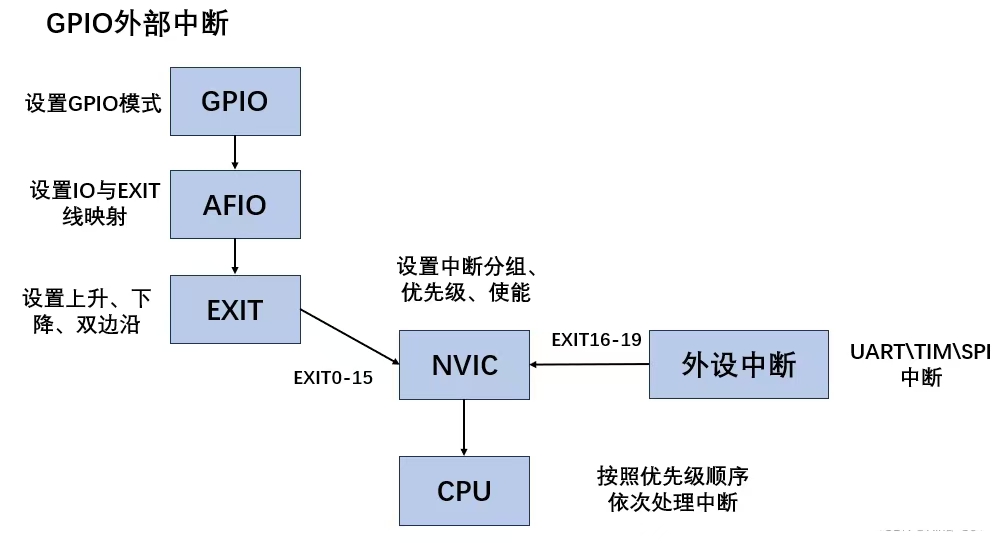
AFIO\_EXTICR3 寄存器配置 EXTI8 到 EXTI11线；

AFIO\_EXTICR4 寄存器配置 EXTI12到 EXTI15线；

特别注意：配置AFIO寄存器之前需要使能AFIO的时钟；

中断的使用步骤

GPIO1外部中断



上述中断步骤共有两条线路，一条为GPIO外部中断的工作流程，即EXIT线（0-15）的工作流程，另外一条为其他外设中断的工作流程，即EXIT线（16-19）的工作流程。

EXIT的配置步骤

1、使能GPIO时钟；

2、设置GPIO的工作模式；（上拉、下拉、浮空输入）

3、使能AFIO时钟；（设置AFIO时钟，开启相关寄存器）；

4、设置GPIO与EXIT映射关系；（选择IO对应EXIT输入线，AFIO\_EXICR寄存器）；

5、设置EXIT屏蔽、上升沿、下降沿；（设置EXIT对应通道的屏蔽和上升沿、下降沿触发，EMR、RTSR、FTSR寄存器）；

6、设置NVIC，分3步（优先级分组、优先级以及使能）；

7、设置中断服务函数（包括清除中断标志）；

STM32 EXIT的HAL库设置步骤：

1、使能GPIO时钟--使用\_HAL\_RCC\_GPIOx\_CLK\_ENANLE

1. GPIO/AFIO/EXIT设置--使用\_HAL\_GPIO\_Init

这些步骤 HAL 库全部封装在 HAL\_GPIO\_Init 函数里面

3、设置中断分组--使用HAL\_NVIC\_SetpriorityGrouping

4、设置中断优先级--使用HAL\_NVIC\_Setpriority

5、使能中断--使用HAL\_NVIC\_EnableRQ

6、设置中断服务函数--EXIT\_TRQrlander

STM32仅有EXIT0-4、EXIT9\_5、EXIT15\_10 7个外部中断服务函数；

中断线 0-4，每个中断线对应一个中断函数，中断线 5-9 共用中断函数EXTI9\_5\_IRQHandler， 中断线 10-15 共用中断函数 EXTI15\_10\_IRQHandler。一般情况下，我们可以把中断控制逻辑直 接编写在中断服务函数中，但是 HAL 库把中断处理过程进行了简单封装。

通用外设驱动

一般外设的配置驱动模型为：

1. 初始化分为：

时钟设置（包括时钟源与开启时钟）；

参数设置；

IO设置（可选）；

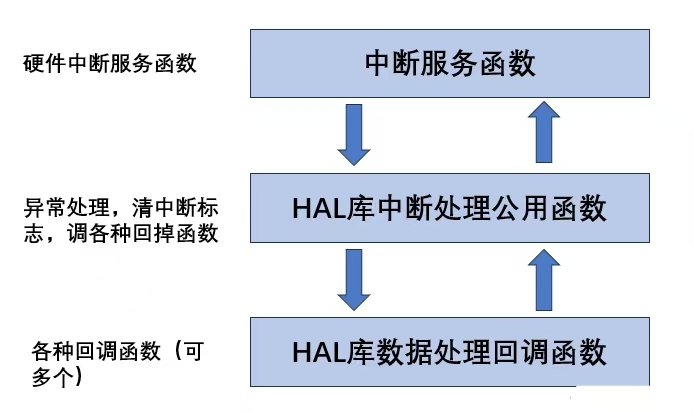
中断设置（可选）；

2、读函数：从外设读取数据（可选）；

3、写函数：往外设写入数据（可选）；

4、中断服务函数（可选）；

中断回调机制（HAL库）



**定时器（TIM)中断**

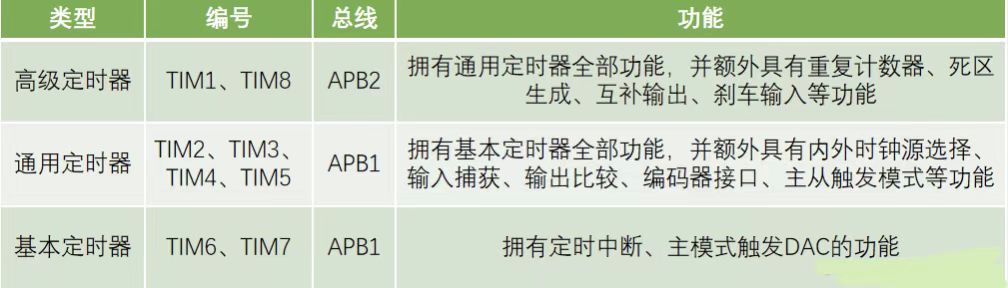
1，什么是定时器（TIM）

定时器就是能够准确规定一段时间的硬件电路。在单片机中，定时器可以对输入的时钟进行计数，并在计数值达到设定值时触发中断。也就是TIM，全名Timer。

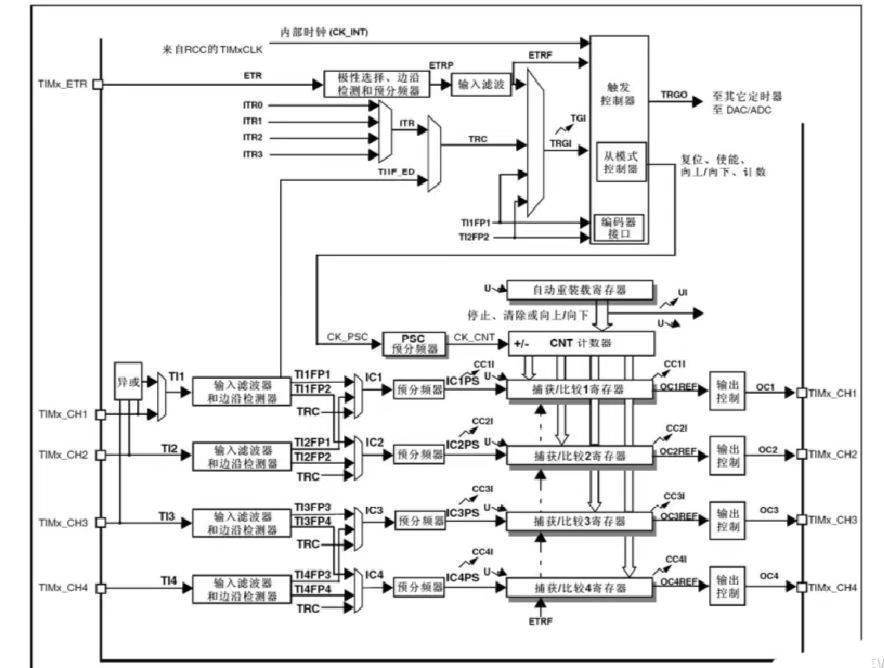
在单片机中是16位计数器、预分频器、自动重装寄存器的时基单元，在72MHz计数时钟下可以实现最大59.65s的定时。

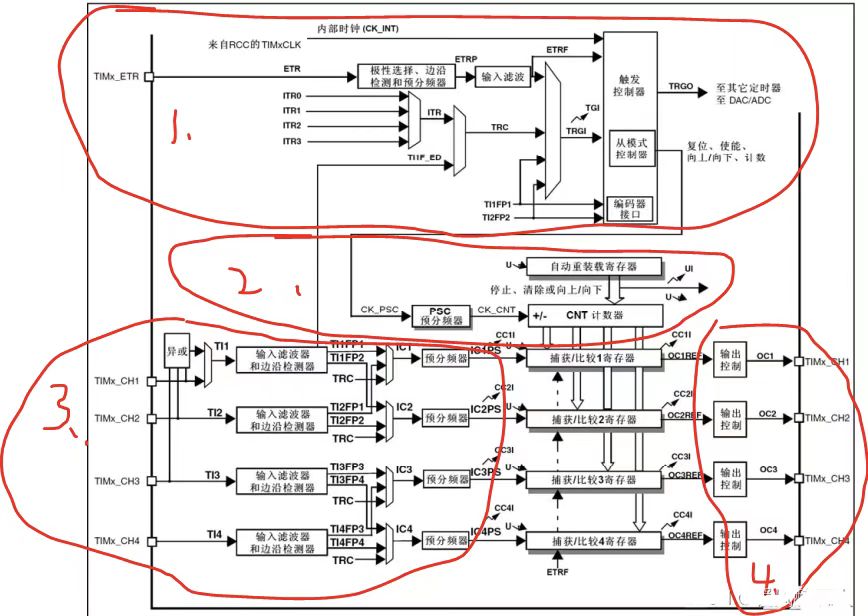
2.分类

根据复杂度和应用场景分为了高级定时器（一般用不到、通用定时器、基本定时器三种类型。



通用定时器框图



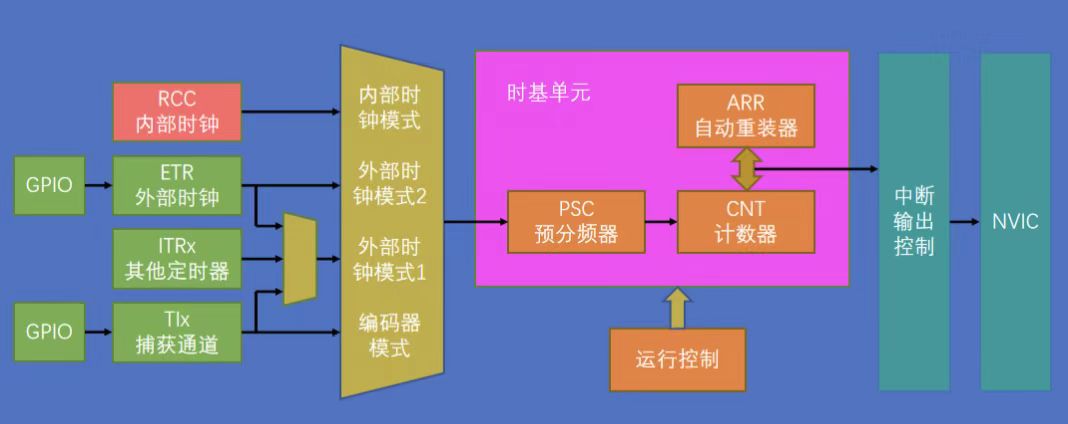


序号1是时钟发生器，序号2是时基单元，序号3是输入捕获，序号4是输出比较，通用定时器大致分为这4个部分。

在时钟发生器中，由内部时钟或外部TIMx\_ETR等经过一系列处理产生时钟源，送入时基单元；在时基单元中经过预分频产生一个时钟，再由CNT进行计数（向上或向下），计数到自动重装载值时可以触发相应的中断；在输入捕获部分中，对TIMx\_CHx中的信号进行捕获，再进行滤波等操作，再通过捕获比较寄存器中捕获到两次信号后的计数器的值，就可以得到脉冲宽度等；在输出比较的部分中，可以在捕获比较寄存器中设置一个数值，用计数器中的值与其比较，当高于该数值时，输出高/低电平，低于该数值时，输出低/高电平。

3，定时中断的基本结构

定时器中断主要应用到通用定时器的时钟发生器和时基单元两个部分。



3.时钟选择：

内部时钟(CK\_INT);

外部时钟模式1：外部输入脚(TIx)；

外部时钟模式2：外部触发输入(ETR)；

内部触发输入(ITRx):使用一个定时器作为另一个定时器的预分频器，例如可以配置一个定时器Timer1而作为一个定时器Timer2的预分频器；

5.计数器模式：

（1）通用定时器可以向上计数、向下计数、向上向下双向计数模式：

①向上计数模式：

计数器从0计数到自动加载值(TIMx\_ARR),然后重新从0开始计数并且产生一个计数器溢出事件；

②向下计数模式：

计数器从自动装入的值(TIMx\_ARR)开始向下计数到0，然后从自动装入的值重新开始，并产生一个计数器向下溢出事件；

③中央对齐模式(向上/向下计数)：

计数器从0开始计数到自动装入的值-1，产生一个计数器溢出事件，然后向下计数到1并且产生一个计数器溢出事件，然后0再从0开始重新计数。

6.定时器中断相关寄存器：

计数器当前值寄存器：TIMx\_CNT;

预分频寄存器：TIMx\_PSC；

自动重装载寄存器：TIMx\_ARR；

控制寄存器1：TIMx\_CR1；

DMA中断使能寄存器：TIMx\_DIER；

7.常用库函数：

定时器参数初始化：

void TIM\_TimeBaseInit（TIM\_TypeDef\* TIMx,TIM\_TimeBaseInitTypeDef\* TIM\_TimeBaseInitStruct）;

定时器使能：

void TIM\_Cmd(TIM\_TypeDef\* TIMx,FunctionalState NewState);

定时器中断使能函数：

void TIM\_ITConfig(TIM\_TypeDef\* TIMx,uint16\_t TIM\_IT,FunctionalState NewState);

状态标志位获取和清除：

 ①FlagStatus TIM\_GetFlagStatus(TIM\_TypeDef\* TIMx,uint16\_t TIM\_FLAG);

 ②void TIM\_ClearFlag(TIM\_TypeDef\* TIMx,uint16\_t TIM\_FLAG);

③ITStatus TIM\_GetITStatus(TIM\_TypeDef\* TIMx,uint16\_t TIM\_IT);

④void TIM\_ClearITPendingBit(TIM\_TypeDef\* TIMx,uint16\_t TIM\_IT);

8.定时器中断实现步骤：

使能定时器时钟：RCC\_APB1PeriphClockCmd();

初始化定时器，配置ARR，PSC寄存器：TIM\_TimeBaseInit();

开启定时器中断，配置NVIC：void TIM\_ITConfig(); NVIC\_Init();

使能定时器：TIM\_Cmd();

编写中断服务函数：TIMx\_IRQHandler();