CORETEX M4 中断系统简明使用指南

版本 A/0 – 2023 年 4 月

概述

中断是几乎所有的处理器都会提供的一种机制。本质上中断是处理器或者外部触发了一些事件，导致处理器必须先暂时放下目前的工作，处理这一突发事件，处理完后再继续原先的工作。一个典型的中断处理流程如下：

- 特定事件发生，外设向处理器发起一个中断请求；
- 处理器挂起当前正在执行的任务；
- 处理器跳转到中断服务函数，处理中断事务；
- 中断服务函数结束，恢复执行原来的任务。

在 ARM 系列的内核中通常把中断看作是一种特殊的异常，以后我们将不再区分两者，如不加说明，则强调的都是它们对主程序所体现出来的“中断”性质，与我们以前学单片机时所讲的概念是相同的，如果非得做一个区分，则中断与异常的区别在于，那中断对于 Cortex 核来说都是“意外突发事件”——也就是说，该请求信号来自核的外面，来自各种偏上外设和扩展的外设，对核来说是“异步”的；而异常则是因为核的活动产生——在执行指令或访问存储器时产生，因此对核来说是“同步”的。

目录

图片列表

表格列表

版本历史

版本	日期	作者	状态	变更
A/0	2023 年 4 月 8 日	CanChai	Released	首次发布。

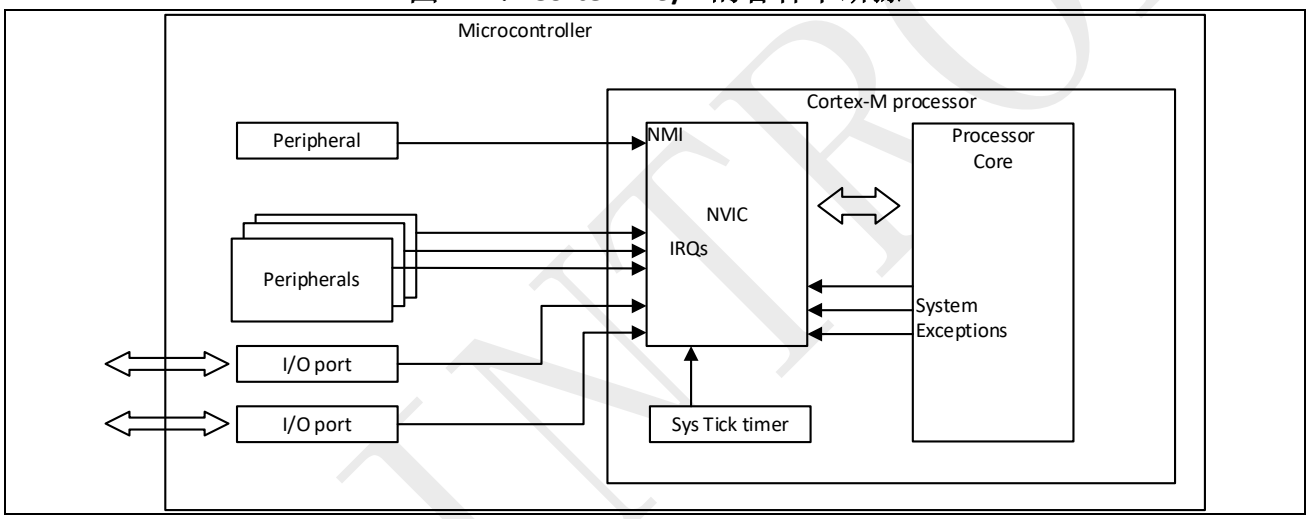
术语或缩写

术语或缩写	描述
SCB	系统控制模块
CSC	系统控制空间
NVIC	中断控制器
AIRCR	应用程序中断及复位控制寄存器

1 中断源

在 ARM 内核中，有一个专门的中断控制器 NVIC(Nested Vectored Interrupt Controller)管理异常和中断。NVIC 可以处理一系列的中断请求(Interrupt Requests，IRQs)和不可屏蔽中断(None-Maskable Interrupt，NMI)。通常 IRQs 是由芯片外设或者外部设备通过用 IO 端口触发，它们可以通过软件被屏蔽；而 NMI 通常是一些与系统正常运行与否相关的中断源，例如：看门狗计时、低电压检测等，这类中断的触发通常标识着一些异常的工作状态，所以不能被屏蔽。此外，在处理器内部还有一个被称为 SysTick 的时钟，可以定时的产生一个中断，它常被用于操作系统的计时。图 1-1：Cortex-M3/4 的各种中断源中显示了 Cortex-M4 的各种中断源，除了上述的三种之外，处理器内核本身也会产生中断，这些中断通常被称为系统中断，一般用于反应系统的各种异常状态，或者是为了支持操作系统而由软件触发的中断。

图 1-1：Cortex-M3/4 的各种中断源



每个中断都有一个中断编号，其中 1 到 15 被处理器内核拿来触发系统异常中断，16 及以上的编号则由各种系统外设使用。Cortex-M4 的 NVIC 可以支持 240 个中断，而实际的芯片厂商并不会把 240 个中断都用上，他们会根据实际设计分配这些中断编号。表 1-1：中断清单中列举了 M4 的各种中断类型。需要注意的是，在设备驱动库中，这些编号的设置有所不同，外设中断的编号是从 0 开始的，而系统中断的编号则是负数，如表 1-1 所示如所示：

表 1-1：中断清单

中断编号	设备驱动库中断编号	中断类型	优先级	描述
1		Reset	-3（最高）	系统复位
2	-14	NMI	-2	不可屏蔽中断
3	-13	HardFault	-1	硬件错误。Cortex-M4 有三种类型的 Fault 异常：总线异常(Bus Fault)、内存管理异常(Memory Management Fault)、和 Usage Fault。它们中任何一种异常发生时都会产生 Hard Fault，当相应的异常被屏蔽时就会进入 HardFault 的中断服务函数中。
4	-12	MemManage	可配置	内存管理异常(Memory Management Fault)：访问了内存管理单元(MPU)定义的不合法区域，或者不合法的内存访问，例如从非代码区域获取指令。
5	-11	BusFault	可配置	总线异常(Bus Fault)
6	-10	UsageFault	可配置	错误的使用方式(Usage Fault)，由非法的指令或者非法的系统状态转换触发。
7-10				保留
11	-5	SVC	可配置	Supervisor Call via SVC instruction
12	-4	Debug	可配置	Debug 监视器
13				保留
14	-2	PendSV	可配置	Pendable request for System Service
15	-1	SysTick	可配置	System Tick 计时中断
16-255	0-239	IRQs	可配置	各种外设中断

系统产生中断时，中断状态寄存器 IPSR 将会记录产生中断的编号。处理器会根据 IPSR 的编号，从中断向量表中查找相应的中断服务函数，Reset 是一种特殊的中断。当系统复位后就会执行 Thread 模式下复位处理函数，此时的 IPSR 的中断编号为 0。

2 中断优先级

在 Cortex-M 系列处理器中，是否要处理一个中断请求，或者是什么时候处理，由中断中断优先级决定。Cortex-M4 支持中断嵌套，高优先级的中断可以抢占低优先级的中断；即在执行低优先级的中断服务函数时，产生了高优先级的中断，处理器会转而处理高优先级的中断请求，结束后继续执行低优先级的服务函数。

表 1-1：中断清单中列出了各种中断的优先级，其中 Reset、NMI 和 HardFault 的优先级是固定的，被表示为了负数，这是为了体现它们具有比其它任何中断都高的优先级。其余的中断优先级都是可以配置的，取值区间是[0,255]，取值越小具有的优先级就越高。虽然内核设计可以支持 256 个优先级，但实际可用的优先级个数是由芯片厂商的实现决定，比如 Spintrol 第三代产品就只支持 16 个优先级。这主要是因为，过高的优先级会使得芯片设计过于复杂，而且功耗也会因此而增加。而实际应用中也没有对大量可配置优先级的需求，所以芯片制造商就会做一些简化。

中断优先级由一个 8 位寄存器控制，如果要实现完整的 256 个中断优先级，那么这 8 位都会被用作优先级控制，但通常芯片厂商会通过保留寄存器的低位来简化优先级的实现，例如，对于一个支持 16 个优先级的 Spintrol 芯片，它的高 4 位被拿来做优先级控制，而低 4 位则保留。

在 SCB（System Control Block, SCB）中的应用程序中断及复位控制寄存器（AIRCR）提供了中断的优先级分组控制，进一步的把优先级寄存器的 8 位分为了 Group Priority 和 Sub-Priority 两个部分。如下表 2-1：中断优先级分组示例示例中，高 4 位[7:4]用作优先级控制，AIRCR 则把这 4 位分为了 x 和 y 两部分；x 为高位，被称为 Group Priority；y 为低位，被称为 Sub-Priority。一共有 5 种分组形式，其中 Group Priority 在一些参考手册中也成为 pre-empt priority(抢占优先级)，它决定了抢占行为：当系统正在响应某异常 L 时，如果来了抢占优先级更高的 H，则 H 可以抢占 L。子优先级则处理“内务”：当抢占优先级相同的异常不止一个悬起时，就最先响应子优先级最高的异常。需要注意的是，子优先级至少 1 位，因此抢占优先级最多 7 位，也就是说，最多只有 128 级的抢占优先级。

表 2-1：中断优先级分组示例

PRIGROUP [2:0]	Interrupt priority level value， PRI_N[7:4]			Number of	
	Binary point	Group priority bits	Subpriority bits	Group priorities	subpriority
0b0xx	0bxxxx	[7:4]	None	16	None
0b100	0bxxx.y	[7:5]	4	8	2
0b101	0bxx.yy	[7:6]	[5:4]	4	4
0b110	0bx.yyy	[7]	[6:4]	2	8
0b111	0byyyy	None	[7:4]	None	16

在 PRI_n 中用 x 表示 group priority 的位，y 表示 subpriority 的位。

3 中断请求及控制状态

在 Cortex-M4 中每个中断都可以被开启（enable）或者关闭（disable），还专门有寄存器标记各个中断是否被请求并且等待被执行（pending），如果正在处理某个中断还会将其标记为 Active 的状态。为了支持对各个中断状态的管理，Cortex-M4 提供了一个中断控制器 NVIC，它是一个可编程的模块，对应的寄存器映射到了系统控制空间（System Control Space, SCS），详情可以参考 Cortex-M4 的内存系统说明文档。NVIC 管理着异常和中断的配置、优先级、屏蔽设置，具有支持中断嵌套，基于向量查找服务函数入口的特点。

如图 3-1：中断状态的时序简图所示，当有中断请求时，NVIC 会在其寄存器中标识，一旦标识成功即使中断请求信号消失了，也会等待被处理，这一状态就是 Pending。中断被处理需要满足三个条件：

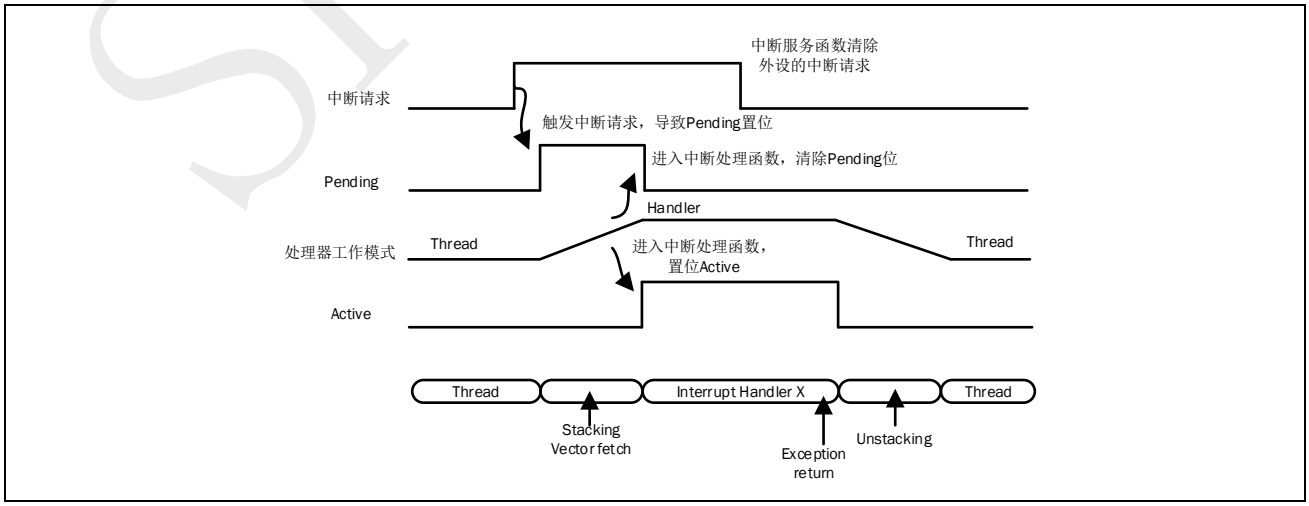
- a.触发中断请求信号，进入 Pending 状态；
- b.中断被使能了（enable）；
- c.当前系统没有正在处理更高或者同等优先级的中断；

对于一个 Pending 状态下的中断，如果不满足后两个条件，那么 Pending 状态会一直保持到条件满足或者被用户取消为止。

中断 Active 之前，处理器需要先进行压栈（Stacking）操作保存当前正在处理的任务，同时通过查表（Vector Fetching）的机制找到中断服务函数入口，并控制 PC 寄存器跳转到服务函数。进入服务函数后，NVIC 会清除 Pending 标识位同时置位 Active。当中断服务函数处理完成后返回时，需要进行出栈（Unstacking）操作恢复之前的处理任务。

Cortex-M4 支持脉冲式和电平式两种中断形式，而 Spintrol 第三代产品都是脉冲式中断。图 3-1：中断状态的时序简图中所示的就是一个电平式的例子，在该例子中中断请求信号会由外设一直保留，需要在中断服务函数中手动地通过对外设的寄存器访问以清除中断请求，否则将会重复进入该中断。而脉冲式中断则是指中断请求信号不会一致保留，但要求至少维持一个时钟周期。如果能够保证中断服务函数退出时，请求信号一定消除了，那么这种中断形式可以不手动清除中断请求。

图 3-1：中断状态的时序简图



4 中断向量表

当中断产生时，处理器需要了解中断服务函数的入口，在 Cortex-M4 中采用了一种向量的机制予以实现。在内存空间的起始部分存放了一个线性表，其中记录了每一个中断的服务函数入口地址，这一向量表可以从启动文件中找到相应的配置。

一般系统复位时，起始地址 0x00 处就是系统的向量表。而通常起始地址所对应的物理存储是 ROM，是不能修改的，但在一些特殊的情况下，我们还是希望修改向量表的位置，甚至每个中断的服务函数都可以变动。为了满足这一需求，NVIC 提供了一个向量表偏移寄存器(Vector Table Offset Register, VTOR)来配置向量表的起始地址。

当中断产生时，系统会根据中断向量编号从向量表中查找服务函数的入口。例如，NMI 的中断编号为 2，那么当 VTOR=0x10000000 时，NMI 的服务函数入口地址就保存在 2*0x04+0x10000000=0x10000008。

5 涉及中断的寄存器

- 中断控制器 NVIC 用来控制可屏蔽中断。
- Cortex-M4 还提供了一些异常/中断掩码寄存器来对不同类型的中断进行屏蔽。
- 在 SCB(System Control Block)中还有一个中断控制和状态寄存器 ICSR 记录了系统的中断状态和 NMI 中断。
- 在 SCB 中的配置和控制寄存器 CCR 控制着进入中断的方式，也用于使能一些 trap 异常。
- 在 SCB 中的系统异常优先级寄存器 SHPR 用于设置 fault, SVCall, SysTick, PendSV 等系统异常的优先级。