快速链接:

.

👉👉👉 ARMv8/ARMv9架构入门到精通-[目录] 👈👈👈

- 付费专栏-付费课程 **【购买须知】:**
- 联系方式-加入交流群 ----**联系方式-加入交流群**
- 个人博客笔记导读目录(全部)

引流关键词:armv8, armv9, gic,gicv2,gicv3,异常, 中断，irq,fiq,serror,sync，同步异常，异步异常，向量表，向量表基地址，VBAR，vbar_el3，中断嵌套，中断级联，Linux Kernel，optee,ATF,TF-A,optee,hypervisor, SPM

目录

14 中断路由(信号流)的总结

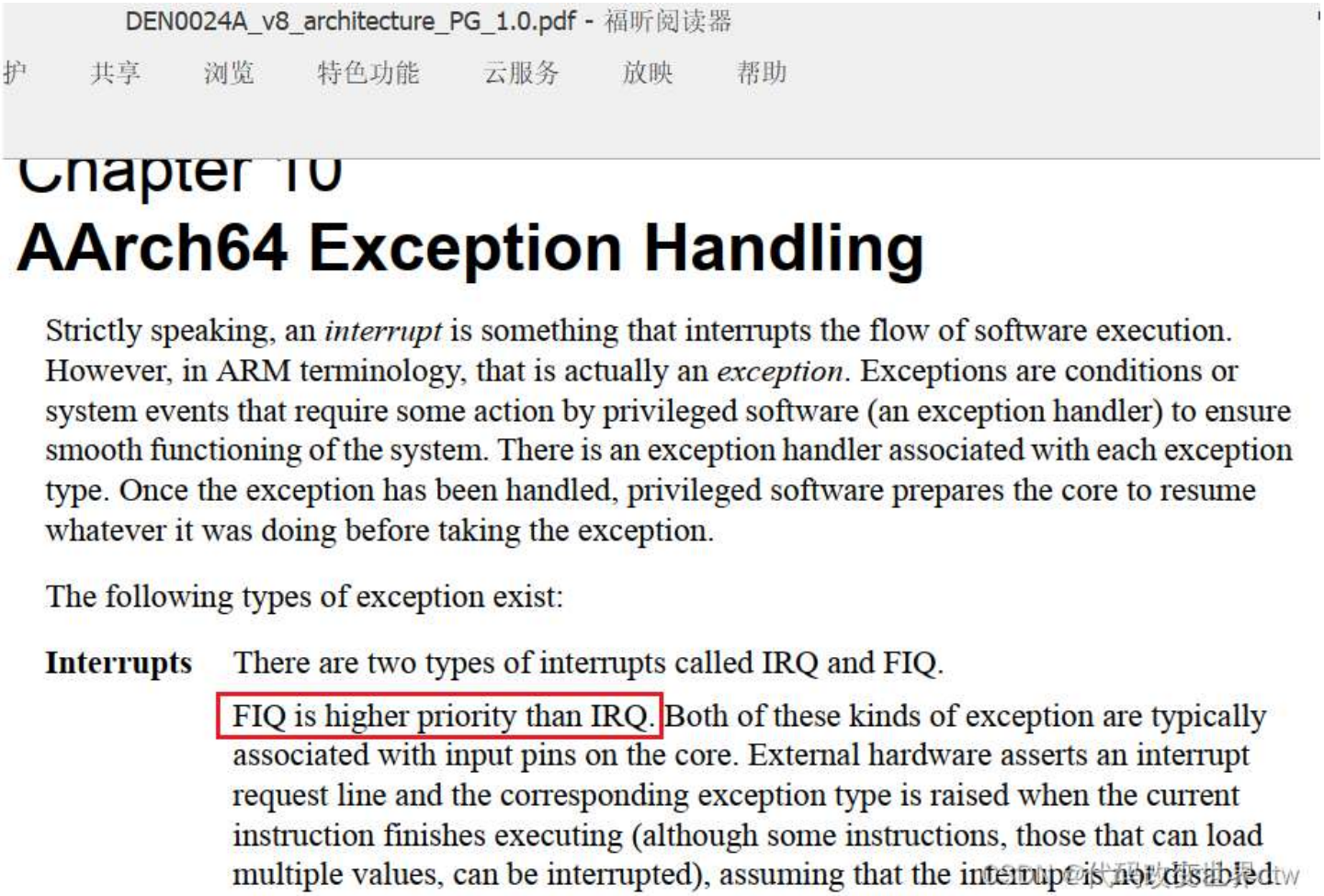
1 中断的定义

有人说，中断就包含IRQ和FIQ，其实这是不准确的，准确的说法应该是：产生到aarch64的异步异常(包括IRQ, FIQ, SError) 可看作中断。

官方文档原话： `In the Armv8-A architecture, asynchronous exceptions that are taken to AArch64 state are also known as interrupts.`

2 FIQ和IRQ

有人说FIQ是快速中断, FIQ比IRQ具有较高的优先级，而且他还能提出ARM官方文档来证明他说的正确性：


DEN0024A_v8_architecture_PG_1.0.pdf - 福昕阅读器

护 共享 浏览 特色功能 云服务 放映 帮助

Chapter 10

AArch64 Exception Handling

Strictly speaking, an *interrupt* is something that interrupts the flow of software execution. However, in ARM terminology, that is actually an *exception*. Exceptions are conditions or system events that require some action by privileged software (an exception handler) to ensure smooth functioning of the system. There is an exception handler associated with each exception type. Once the exception has been handled, privileged software prepares the core to resume whatever it was doing before taking the exception.

The following types of exception exist:

Interrupts There are two types of interrupts called IRQ and FIQ.

FIQ is higher priority than IRQ. Both of these kinds of exception are typically associated with input pins on the core. External hardware asserts an interrupt request line and the corresponding exception type is raised when the current instruction finishes executing (although some instructions, those that can load multiple values, can be interrupted), assuming that the interrupt is not disabled.

其实，这也是错误的!

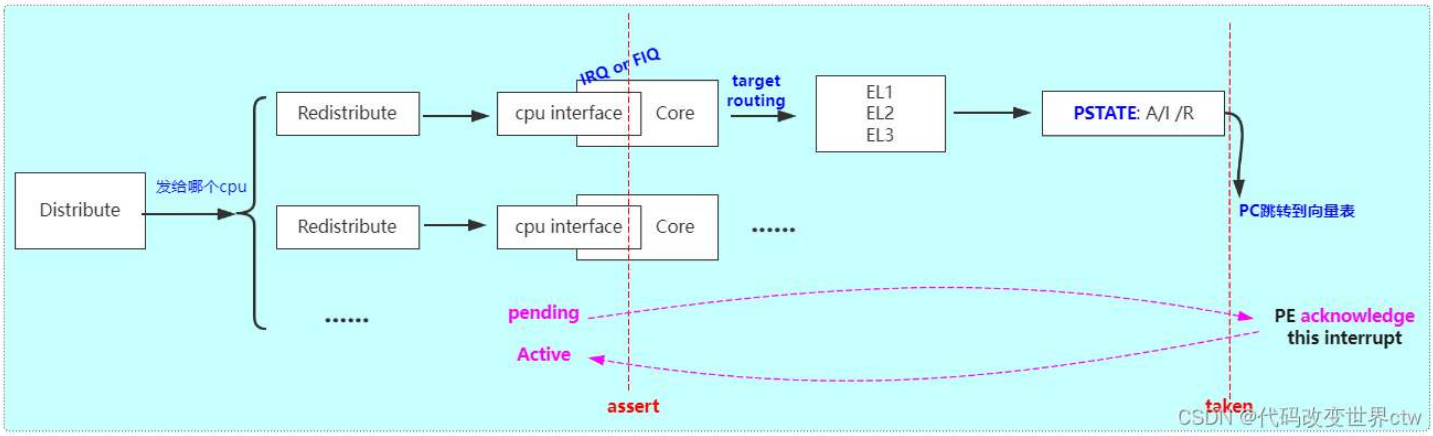
正确的说法是FIQ和IRQ具有同样的优先级(默认的情况下，我们只讨论armv8-aarch64和armv9)

4.3. IRQ and FIQ

The Armv8-A architecture has two exception types, IRQ and FIQ, that are intended to be used to generate peripheral interrupts. In other versions of the Arm architecture, FIQ is used as a higher priority fast interrupt. This is different from Armv8-A, in which FIQ has the same priority as IRQ.

IRQ and FIQ have independent routing controls and are often used to implement Secure and Non-secure interrupts, as discussed in the Generic Interrupt Controller guide.

3 中断术语的介绍

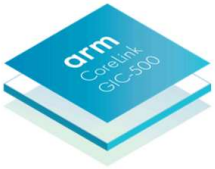


SPIs(Share Periphral Interrupts)中断进来之后，由inactive状态变成pending，此时中断标记为IRQ/FIQ，这是也就是中断assert了，然后该中断会根据HCR/SCR等的配置进行路由（路由到哪个Exception Level等），这个过程也就target，也可以叫做routing。路由之后，在部分场景下还会再检查PSTATE的MASK位，接下来就是PE acknowledge了，此时也就是中断被taken了。 PE acknowledge后，cpu interface会将该中断置为Active.

4 gic中断控制器的介绍

（注意：本文重点介绍armv8/armv9异常中断，不会展开介绍gic，这里只是带一下简单概念）
gic中断控制器有众多版本，gicv2/gicv3/gicv4是gic的架构，gic500/gic600是具体的gic IP。
而在armv8/armv9中，基本都是使用的gicv3/gicv4。 如下图所示，每一个core都定义了，它说使用的gic架构。

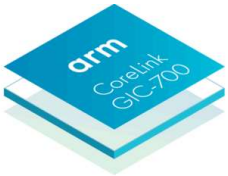
Cortex-A15, Cortex-A7



Cortex-A73, Cortex-A72, Cortex-A57, Cortex-A53

Cortex-A76, Cortex-A55

Cortex-A78AE, Cortex-A76AE, Cortex-A76, Cortex-A55

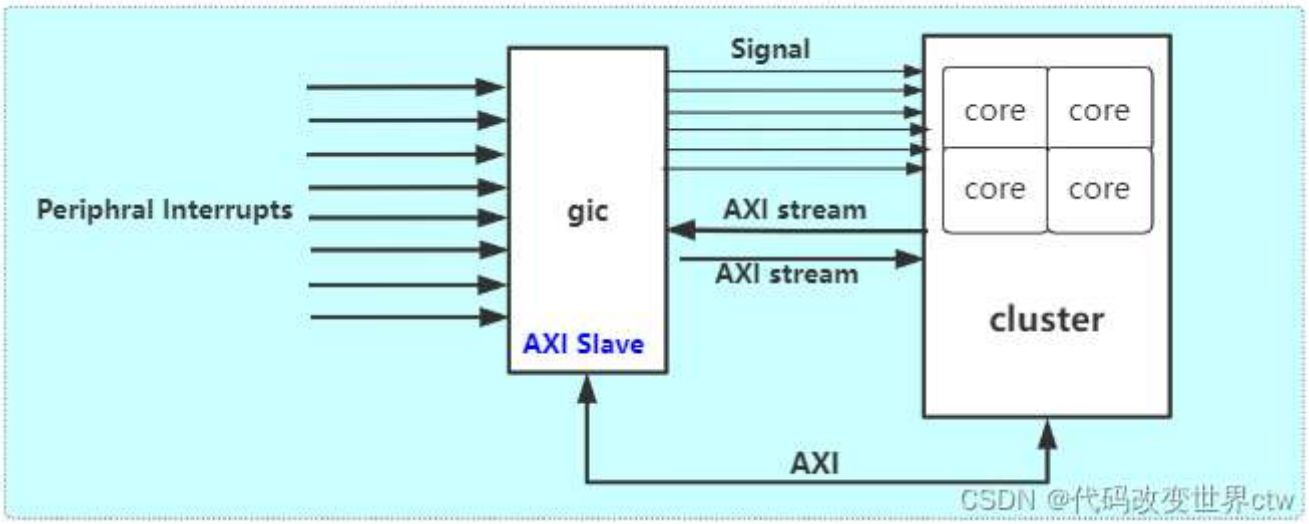


enhancing virtualization capabilities

Feature	Cortex-A32	Cortex-A34	Cortex-A35	Cortex-A53	Cortex-A55	Cortex-A57‡	Cortex-A65	CortexA65AE	Cortex-A72	Cortex-A73	Cortex-A75	Cortex-A76	Cortex-A76AE	Cortex-A77	Cortex-A78	Cortex-A78AE
Architecture	Armv8-A (AArch32 only)	Armv8-A (AArch64 only)	Armv8-A	Armv8-A	Armv8.2-A	Armv8-A	Armv8.2-A (AArch64 only)	Armv8.2-A (AArch64 only)	Armv8-A	Armv8-A	Armv8.2-A	Armv8.2-A (AArch32 at EL0 only)	Armv8.2-A (AArch32 at EL0 only)	Armv8.2-A (AArch32 at EL0 only)	Armv8.2-A (AArch32 at EL0 only)	Armv8.2-A (AArch32 at EL0 only)
Interrupt Controller	External GICv3	External GICv3	External GICv3	External GICv3	External GICv4	External GICv3	External GICv4	External GICv4	External GICv3	External GICv3	External GICv3	External GICv4	External GICv4	External GICv4	External GICv4	External GICv4

Feature	Cortex-A5	Cortex-A7	Cortex-A9‡	Cortex-A15‡	Cortex-A17‡
Architecture	Armv7-A	Armv7-A	Armv7-A	Armv7-A	Armv7-A
Interrupt Controller	Optional Integrated GIC v1 (MP only) Integrated GIC v1 (MP only)	Optional Integrated GIC v2	Internal Integrated GIC v1 (MP only)	Optional Integrated GIC v2	Optional Integrated GIC v2

gic中断控制器与core连接的硬件框图

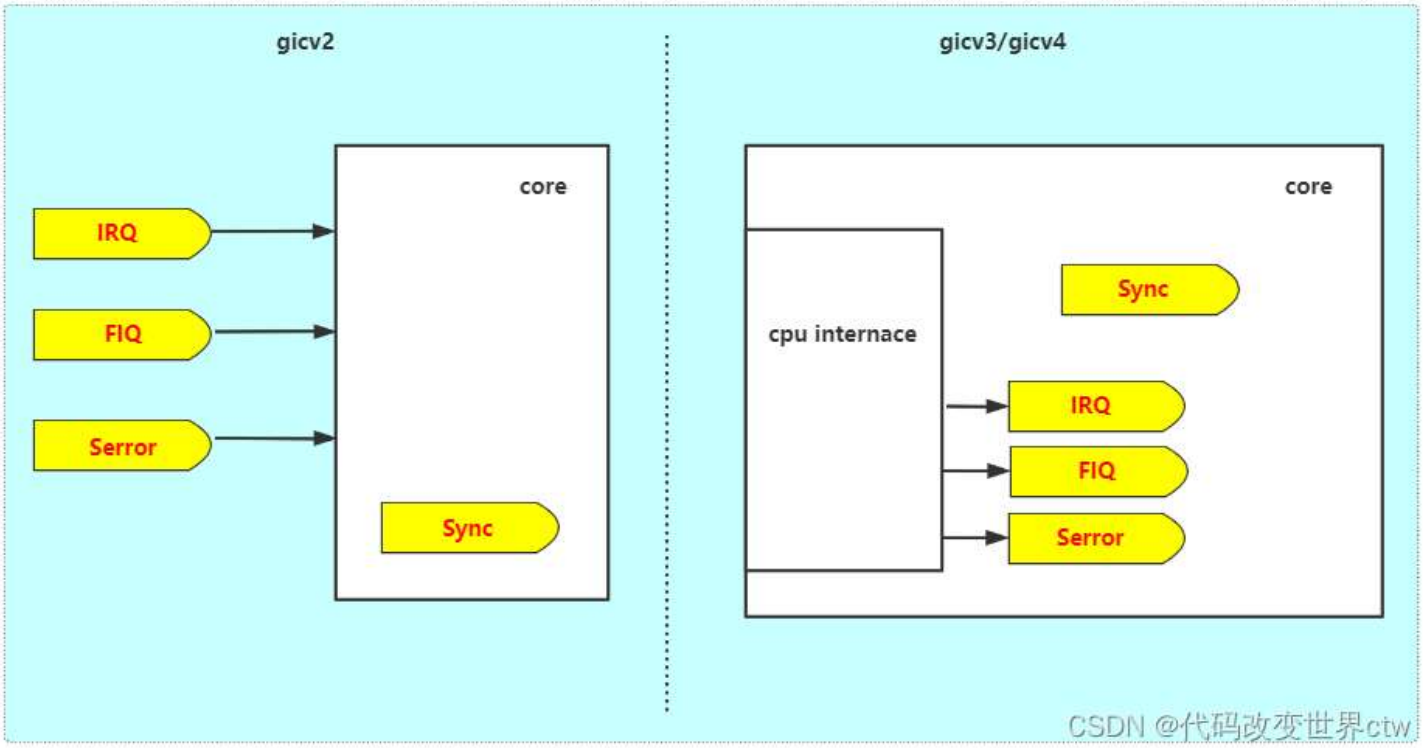


5 Core中的中断控制器接口的介绍

在gicv2中，gic中断控制器将中断信号(irq,fiq,serror)直接以signal的方式发送给core。

而在gicv3中，gic的组件发生变化，将gic的cpu internface接口做到了core中，在这种情况下，gic将中

断信号通过AXI Stream发送给core(cluster)，然后cpu interface再继续发送irq/fiq/serror信息。



6 同步异常和异步异常的概念

6.1、同步异常和异步异常的定义

具备以下3个行为的称之为同步异常：

• The exception is generated as a result of direct execution or attempted execution of an instruction.

• The return address presented to the exception handler is guaranteed to indicate the instruction that caused the exception.

• The exception is precise

其实就是说：

- 异常是由执行或尝试执行指令产生的
- 产生异常的那个位置是确定的，即每次执行到“那个指令处”就会产生
- 异常是precise的

具备以下3个行为的称之为异步异常：：

• The exception is not generated as a result of direct execution or attempted execution of the instruction stream.

• The return address presented to the exception handler is not guaranteed to indicate the instruction that caused the exception.

• The exception is imprecise.

其实就是说：

- 异常不是由执行或尝试执行指令产生的
- 产生异常的那个位置不是确定的，即不知道执行到哪里，就产生了异常
- 异常是imprecise的

那么precise 和 imprecise 又是什么意思呢？？

An exception is described as *precise* when the exception handler receives the PE state and memory system state that is consistent with the PE having executed all of the instructions up to but not including the point in the instruction stream where the exception was taken, and none afterwards.

An exception is described as *imprecise* if it is not precise.

比较绕、比较难懂，咱们换一个说法：按照预期产生的异常称之precise，反之imprecise

6.2、系统中有哪些异步异常?

其实主要就是：**irq, fiq, SError**

Physical interrupts Are signals sent to the PE from outside the PE. They are:

- SError. System Error.
- IRQ.
- FIQ.

Virtual interrupts Are interrupts that software executing at EL2 can enable and make pending. A virtual interrupt is taken from EL0 or EL1 to EL1. Virtual interrupts have names that correspond to the physical interrupts:

- vSError.
- vIRQ.
- vFIQ

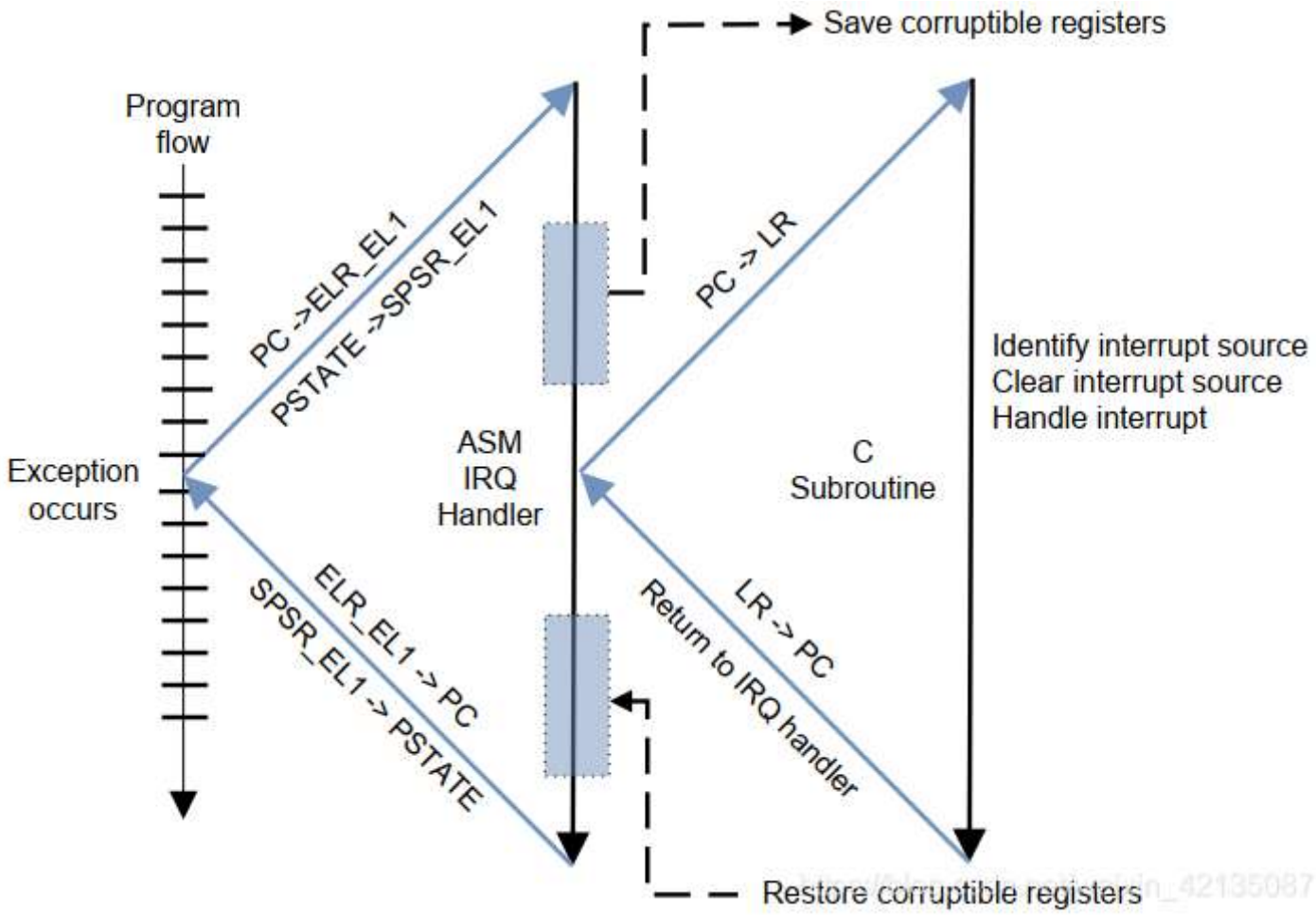
6.3、系统中有哪些同步异常?

- 尝试执行UNDEFINED指令产生的任何异常，包括：
  (1)、尝试在不适当的异常级别执行指令。
  (2)、当指令被禁用时尝试执行指令。
  (3)、尝试执行尚未分配的指令位模式。
- 非法执行状态异常。这些是由尝试执行指令引起的 `PSTATE .IL` 为 1，(详细可参考D1-2486 页上的 AArch64 状态的非法返回事件)
- 使用未对齐的 SP 导致的异常。
- 尝试使用未对齐的 PC 执行指令导致的异常。

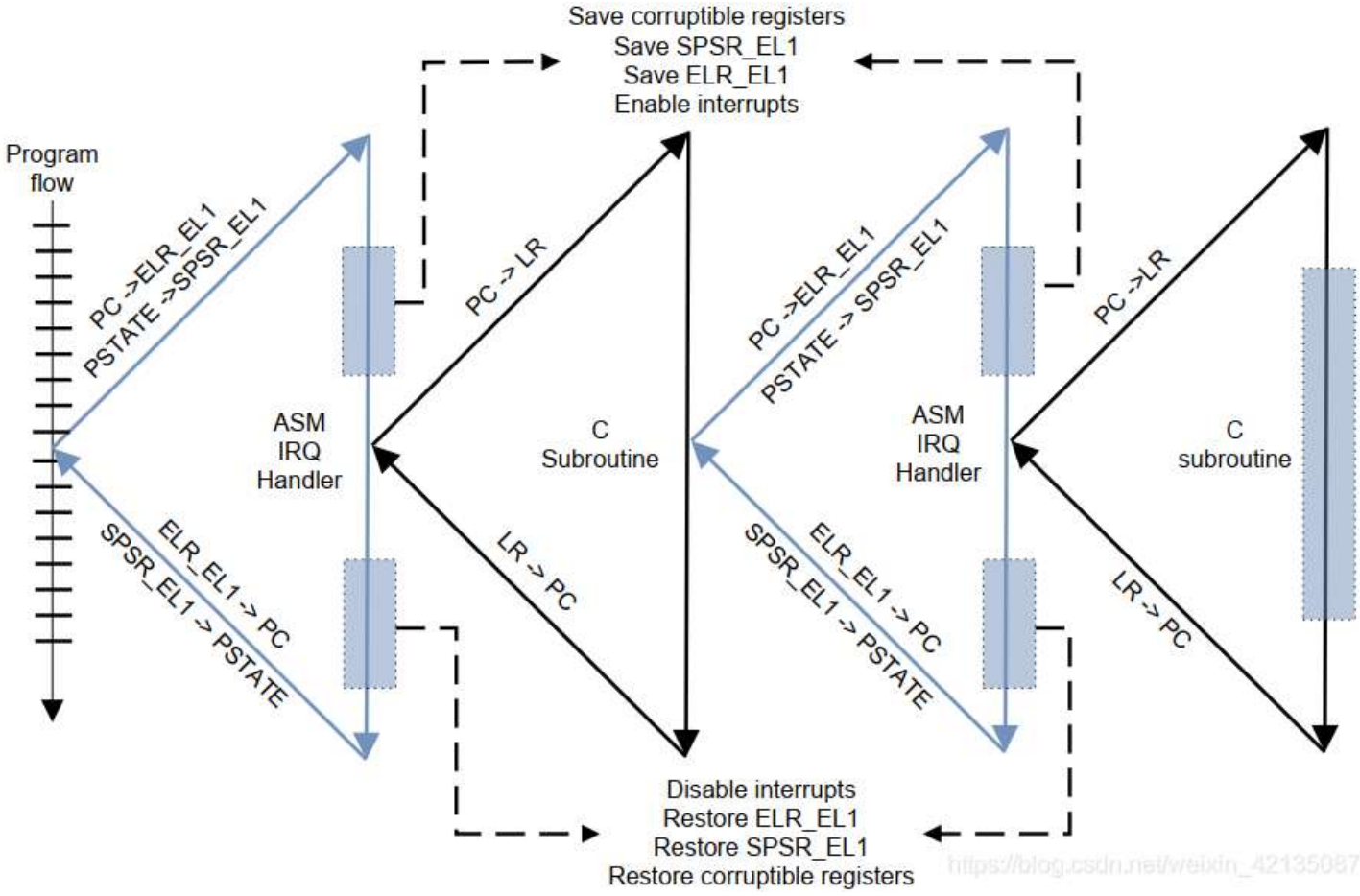
- 由异常生成指令SVC、HVC或SMC引起的异常。
- 尝试执行系统寄存器定义为被捕获到更高的异常级别。(详细可参考可配置的指令使能和禁止，在D1-2510 页)
- 由内存地址转换系统生成的指令中止与尝试相关联从产生故障的内存区域执行指令。
- 内存地址转换系统生成的数据中止与尝试读取或写入产生故障的内存。
- 由地址未对齐引起的数据中止。
- 如果实施FEAT_MTE2，则由标记检查故障引起的数据中止。。
- 所有调试异常：
  (1)、Breakpoint Instruction exceptions.
  (2)、Breakpoint exceptions.
  (3)、Watchpoint exceptions.
  (4)、Vector Catch exceptions.
  (5)、Software Step exceptions.
- 在支持捕获浮点异常的实现中，由捕获的IEEE 浮点异常引起的异常
- 在某些实现中，外部中止。外部中止是失败的内存访问，包括访问地址转换期间发生的内存系统的那些部分。

7 软件对中断的处理流程

正常情况下，当一个中断(异常)进来之后，PE(cpu)跳转到中断向量表，在中断向量表中会再次调用C语言函数，完成中断的处理，流程图如下所示：



ARM Core支持中断抢占，当一个中断正常处理的时候，可能又触发了一个高优先级的中断，示例如下所示：



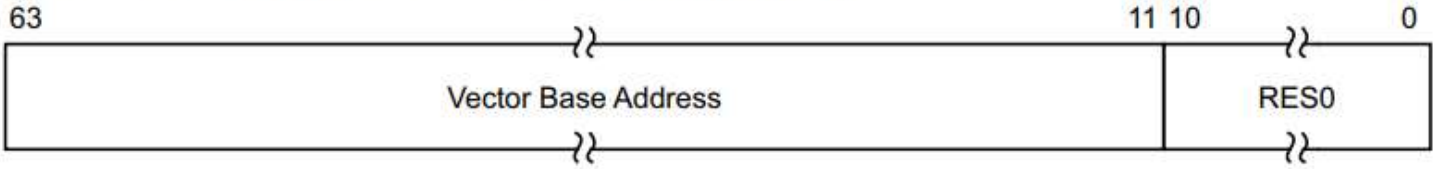
思考你所用的操作系统就真的支持中断嵌套吗？ 如果想支持中断嵌套，需要满足哪些条件呢？ 后文会有详细介绍。

8 向量表基地址寄存器的介绍

armv8定义了VBAR_EL1、VBAR_EL2、VBAR_EL3三个基地址寄存器

General name	Short description	AArch64 register	AArch32 register
VCR	PL1&0 stage 2 Translation Control Register	VTCR_EL2	VTCR
VBAR	Vector Base Address Register	VBAR_EL1 VBAR_EL2 VBAR_EL3	VBAR HVBAR MVBAR
VTTBR	PL1&0 stage 2 Translation Table Base Register	VTTBR_EL2	VTTBR

VBAR_EL1, Vector Base Address Register (EL1)



> **思考：**
>
> 1、VBAR_EL1、VBAR_EL2、VBAR_EL3写入的基地址，是物理地址还是虚拟地址？
> 2、基地址不再放0x00000000的位置吗？
> 3、异常向量表中，没有reset offset了？
> 4、异常向量表中的每一个offset为啥是0x80（128）地址空间？ 以前是多少？
> 5、VBAR_ELx中，为啥末尾11个bit是reserved？

9 中断向量表的介绍

Exception taken from	Offset for exception type			
	Synchronous	IRQ or vIRQ	FIQ or vFIQ	SError or vSError
Current Exception level with SP_EL0.	0x000[a]	0x080	0x100	0x180
Current Exception level with SP_ELx, x>0.	0x200[a]	0x280	0x300	0x380
Lower Exception level, where the implemented level immediately lower than the target level is using AArch64.[b]	0x400[a]	0x480	0x500	0x580
Lower Exception level, where the implemented level immediately lower than the target level is using AArch32.[b]	0x600[a]	0x680	0x700	0x780

我们可以看出，实际上有四组表，每组表有四个offset，分别对应sync，IRQ，FIQ和serror。

- 如果发生异常后并没有exception level切换，并且发生异常之前使用的栈指针是SP_EL0，那么使用第一组异常向量表。

- 如果发生异常后并没有exception level切换，并且发生异常之前使用的栈指针是SP_EL1/2/3，那么使用第二组异常向量表。
- 如果发生异常导致了exception level切换，并且发生异常之前的exception level运行在AARCH64模式，那么使用第三组异常向量表。
- 如果发生异常导致了exception level切换，并且发生异常之前的exception level运行在AARCH32模式，那么使用第四组异常向量表。

另外我们还可以看到的一点是，每一个异常入口不再仅仅占用4bytes的空间，而是占用0x80 bytes空间，也就是说，每一个异常入口可以放置多条指令，而不仅仅是一条跳转指令

注意，到了armv9上，增加了 `FEAT_DoubleFault` 之后，异常向量表稍微变化了一丁点变化，如图中的标注所示：

Exception taken from	Offset for exception type			
	Synchronous (including synchronous External aborts if SCR_EL3.EASE is 0)	IRQ or vIRQ	FIQ or vFIQ	SError, vSError, or Synchronous External aborts if SCR_EL3.EASE is 1 and the target is EL3
Current Exception level with SP_EL0.	0x000	0x080	0x100	0x180
Current Exception level with SP_ELx, x0.	0x200	0x280	0x300	0x380
Lower Exception level, where the implemented level immediately lower than the target level is using AArch64.	0x400	0x480	0x500	0x580
Lower Exception level, where the implemented level immediately lower than the target level is using AArch32.	0x600	0x680	0x700	0x780

也就是说，当 `FEAT_DoubleFault` 开启之后，且 `SCR_EL3.EASE` 比特设置为1, 那么此时target到EL3的Synchronous External abort将会跳转到Serror offset。

EASE, bit [19]
When FEAT_DoubleFault is implemented:

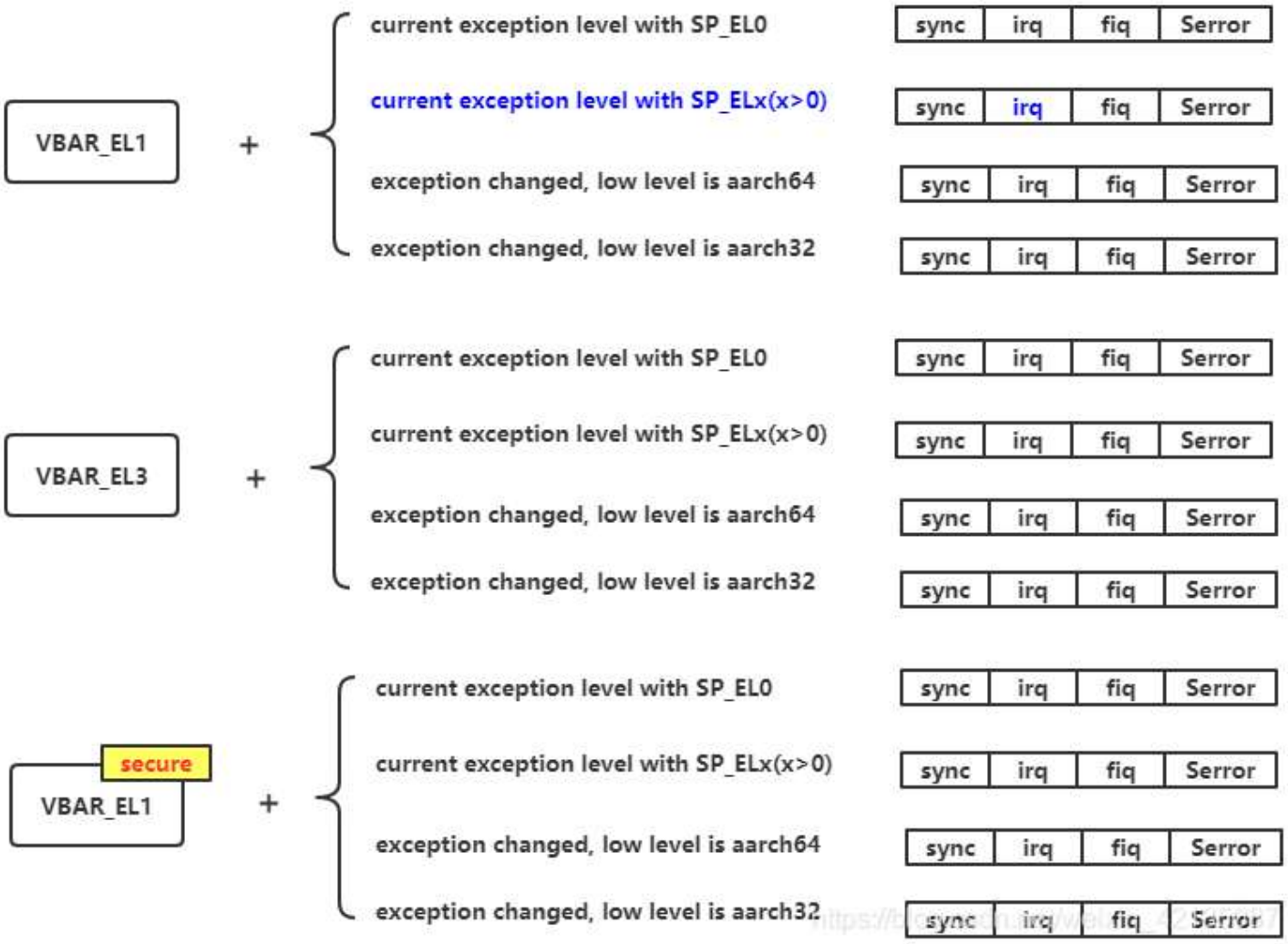
External aborts to SError interrupt vector.

EASE	Meaning
0b0	Synchronous External abort exceptions taken to EL3 are taken to the appropriate synchronous exception vector offset from VBAR_EL3.
0b1	Synchronous External abort exceptions taken to EL3 are taken to the appropriate SError interrupt vector offset from VBAR_EL3.

在中断产生之后，PC(或PE 或 Core 或 cpu)将跳转到VBAR + 中断offset处。 事实上在armv8-aarch64或armv9体系中，有3个VBAR_ELx寄存器，另外对于VBAR_EL1虽然只有一个，但是在不同Security状态的操作系统中，有着不同的cpu context，即也是可以看做成两份。 如果是要考虑虚拟化，那么VBAR_EL2可能也会有两份，VBAR_EL1可能会有多份。

AArch64	VBAR_EL1	Vector Base Address Register (EL1)
AArch64	VBAR_EL2	Vector Base Address Register (EL2)
AArch64	VBAR_EL3	Vector Base Address Register (EL3)

如下，是在不考虑EL2/虚拟化的时候，画的一张向量表总截图，即当一个中断来时，硬件会自动选择哪一个VBAR_ELx寄存器，硬件会自动选择哪一组向量表，硬件会自动选择哪一个offset



10 中断进入和中断退出时的硬件自动行为

10.1 当异常进来之后ARM CORE的硬件自动的行为（Exception entry）

[for common]

- PE(即当前PSTATE)状态保存在目标异常级别的SPSR_ELx中
- 返回地址保存在目标异常级别的ELR_ELx中
- 所有PSTATE .{D, A, I, F} 都设置为 1。 —即关闭了所有中断
- 所选的堆栈指针寄存器是目标异常级别的专用堆栈指针寄存器 —即使用sp_elx
- 执行移动到目标异常级别，并从异常向量定义的地址开始 —即跳转到VBAR_ELx

[for 同步异常]

- 如果异常是同步异常或 SError 中断，则描述原因的信息， 异常保存在目标异常级别的ESR_ELx 中。
- 如果指令中止异常、数据中止异常、PC 对齐错误异常或Watchpoint异常，且目标异常是 aarch64， 错误的虚拟地址保存在FAR_ELx 中。(Instruction Abort exception, Data Abort exception, PC alignment fault exception, or a Watchpoint exception)
- 如果指令中止异常，或数据中止异常被带到 EL2 并且故障是与第 2 阶段转换，故障 IPA 保存在 HPFAR_EL2 中

[for Serror]

- 对于物理 SError 中断异常，在以下任一情况下，物理 SError 的挂起状态将被清除
  SError 中断是边沿触发的。
  `FEAT_DoubleFault` 已实现
  如果Reliability, Availability, and Serviceability Extension被实施，并且在采取 SError 时中断，记录在ESR_ELx 中的综合症指示除IMPLEMENTATION之外的 SError定义或未分类的 SError 中断综合症
- 对于虚拟 SError 中断异常，虚拟 SError 的挂起状态， `HCR_EL2 .VSE` 位清零

[for FEAT]

- PSTATE .SSBS 设置为SCTLR_ELx .DSSBS的值
- 如果FEAT_UAO实现，PSTATE .UAO被设置为0
- 如果FEAT_MTE实现，PSTATE .TCO设置为1
- 如果实现了FEAT_BTI，从 AArch64 到 AArch64 的**异步异常**，PSTATE .BTYPE 被复制到 SPSR_ELx .BTYPE，然后设置为 0
- 如果实现了FEAT_BTI，在将某些类型的同步异常从 AArch64 转移到 AArch64 时，PSTATE .BTYPE 复制到SPSR_ELx .BTYPE 然后设置为 0 这些类型的同步异常是：
  软件步骤异常。
  PC 对齐错误异常。
  指令中止异常。
  断点异常或地址匹配向量捕获异常。
  非法执行状态异常。
  软件断点异常。
  分支目标异常。

- 如果FEAT_IESB被实现，当有效数值的的SCTLR_ELx .IESB位在目标异常level为1，PE插入错误同步事件

10.2 当异常退出时ARM CORE的硬件自动的行为（Exception return）

（On executing an Exception return instruction at ELx）

- PC从ELR_ELx恢复
- PSTATE从SPSR_ELx恢复

11 中断的标记

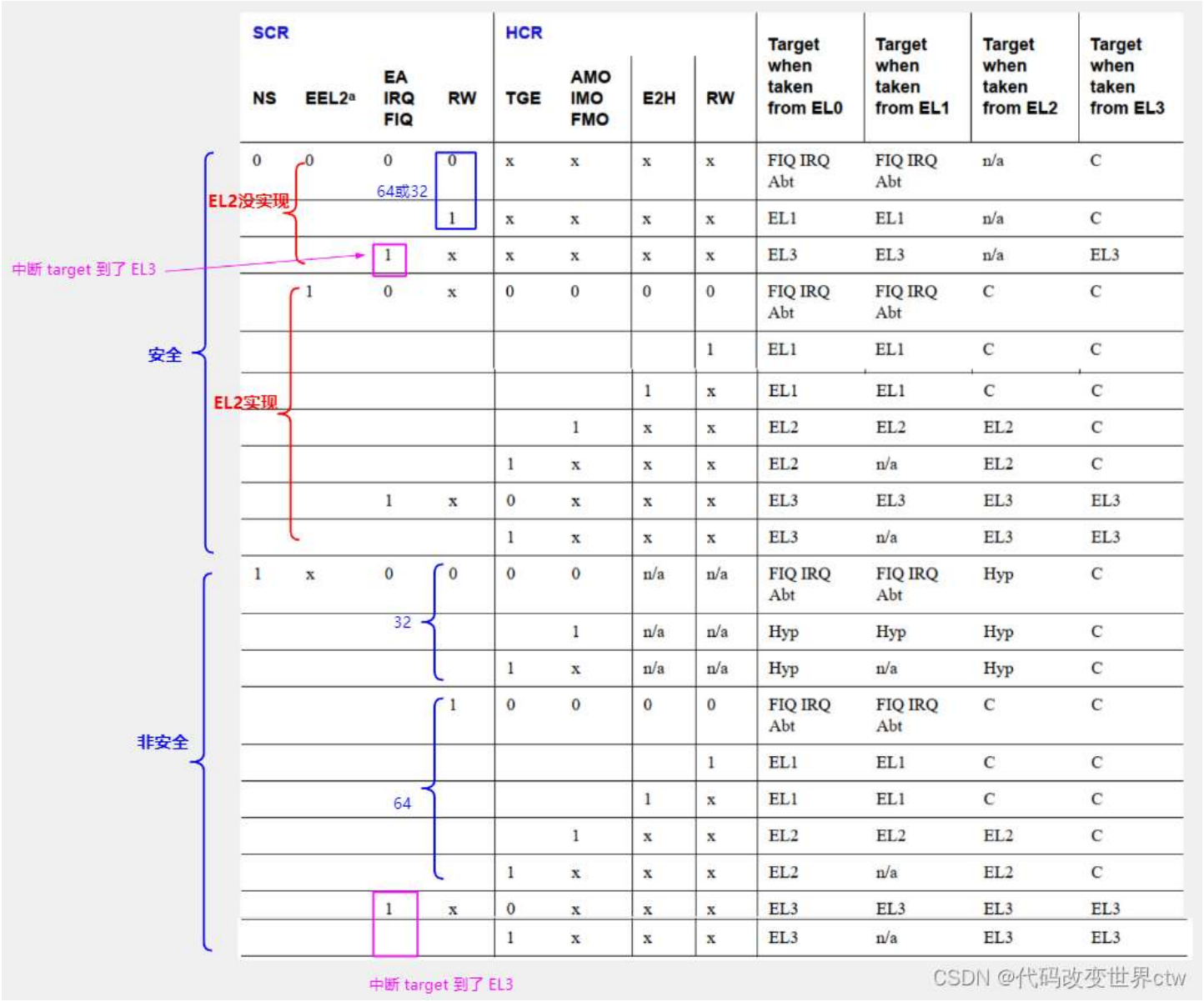
在gicv3中断控制器中，对中断进行了分组：Group0、Secure Group1、Non-secure Group1。当一个中断进来的时候，cpu interface会根据中断的分组类型和当前PE的security状态来决定是标记为IRQ还是FIQ

EL and Security state of PE	Group 0	Group 1	
		Secure	Non-secure
Secure EL0/1	FIQ	IRQ	FIQ
Non-secure EL0/1/2	FIQ	FIQ	IRQ
EL3	FIQ	FIQ	FIQ

12 中断的路由

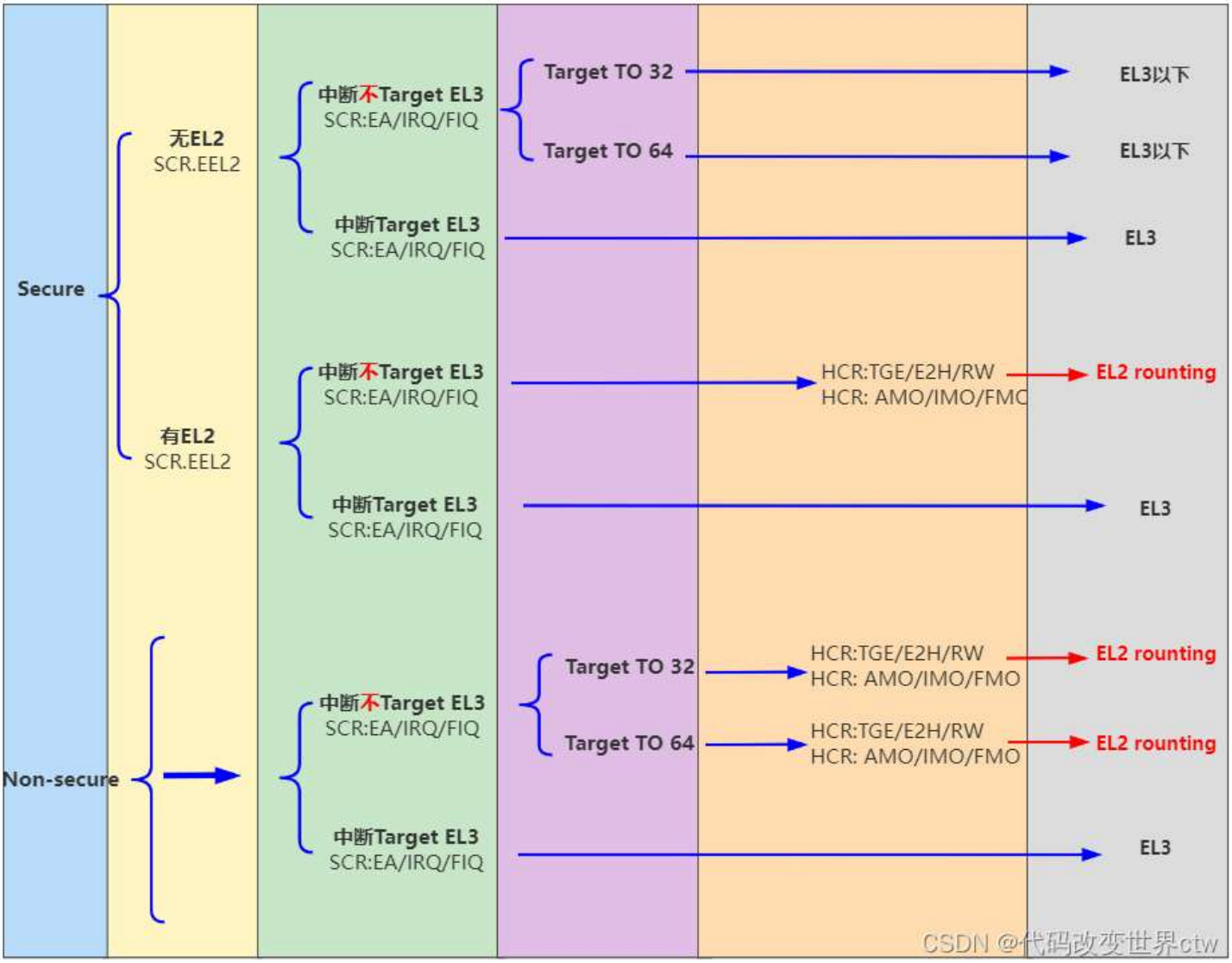
我们知道系统中有三个基地址VBAR_EL1、VBAR_EL3、VBAR_EL1(secure)，那么到底是使用哪一个呢?

由Routing when both EL3 and EL2 are implemented 表来决定，中断routing到了EL1则使用VBAR_EL1，routing到了EL3则使用VBAR_EL3，routing到了secure EL1则使用VBAR_EL1(secure)



SCR				HCR							
NS	EEL2[a]	EA IRQ FIQ	RW	TGE	AMO IMO FMO	E2H	RW	Target when taken from EL0	Target when taken from EL1	Target when taken from EL2	Target when taken from EL3
0	0	0	0	x	x	x	x	FIQ IRQ Abt	FIQ IRQ Abt	n/a	C
			1	x	x	x	x	EL1	EL1	n/a	C
		1	x	x	x	x	x	EL3	EL3	n/a	EL3
	1	0	x	0	0	0	0	FIQ IRQ Abt	FIQ IRQ Abt	C	C
							1	EL1	EL1	C	C
						1	x	EL1	EL1	C	C
					1	x	x	EL2	EL2	EL2	C
				1	x	x	x	EL2	n/a	EL2	C
		1	x	0	x	x	x	EL3	EL3	EL3	EL3
				1	x	x	x	EL3	n/a	EL3	EL3
1	x	0	0	0	0	n/a	n/a	FIQ IRQ Abt	FIQ IRQ Abt	Hyp	C
					1	n/a	n/a	Hyp	Hyp	Hyp	C
				1	x	n/a	n/a	Hyp	n/a	Hyp	C
			1	0	0	0	0	FIQ IRQ Abt	FIQ IRQ Abt	C	C
							1	EL1	EL1	C	C
						1	x	EL1	EL1	C	C
					1	x	x	EL2	EL2	EL2	C
				1	x	x	x	EL2	n/a	EL2	C
		1	x	0	x	x	x	EL3	EL3	EL3	EL3
				1	x	x	x	EL3	n/a	EL3	EL3

为了更直观的理解，总结成了下面的一个流程图：



13 中断的MASK（屏蔽）

在PSTATE中，A/I/F比特分别可以对SError、IRQ、FIQ进行MASK

- SError： PSTATE.**A**
- IRQ： PSTATE.**I**
- FIQ： PSTATE.**F**

但是在有些场景下，MASK将会失效，如在一些中断被强制target到EL3的配置下，中断的taken就不在关心PSTATE的mask位了。

以下表格做出了详细的说明：

SCR				HCR			Effect of the interrupt mask when executing at:			
NS	EEL2[a]	EA IRQ FIQ	RW	TGE	E2H[b]	AMO IMO FMO	EL0	EL1	EL2	EL3
0	0	0	x	x	x	x	B	B	n/a	C
		1	x	x	x	x	A	A	n/a	A/B
	1	0	x	0	x	0	B	B	C	C
						1	A	A	B	C
				1	0	x	A	n/a	B	C
					1	x	B	n/a	B	C
		1	x	0	x	x	A	A	A	A/B
				1	x	x	A	n/a	A	A/B
1	x	0	0	0	n/a	0	B	B	B	C
						1	A	A	B	C
				1	n/a	x	A	n/a	B	C
			1	0	x	0	B	B	C	C
						1	A	A	B	C
				1	0	x	A	n/a	B	C
					1	x	B	n/a	B	C
		1	x	0	x	x	A	A	A	A/B
				1	x	x	A	n/a	A	A/B

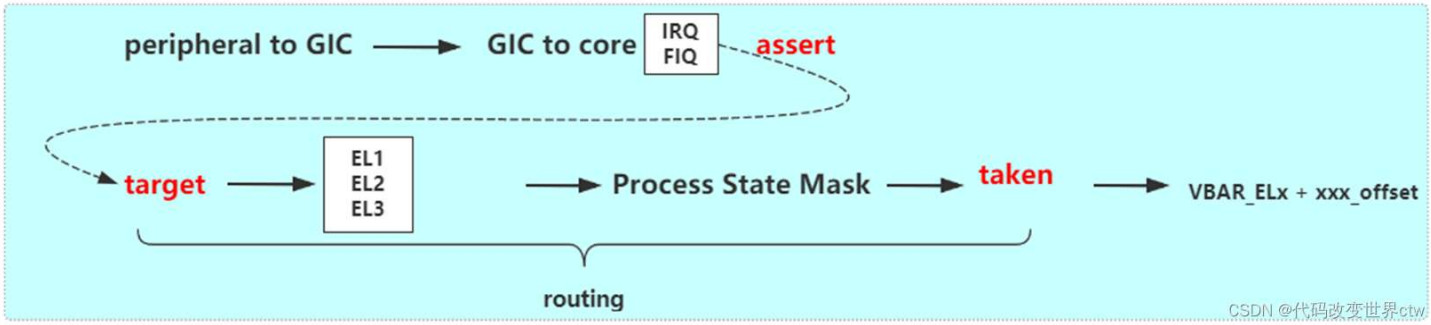
其中：

- A 表示 中断的taken 将忽略 PSTATE的MASK位
- B 表示 中断的taken 不会忽略 PSTATE的MASK位，如果MASK了，就不会taken了。
- C 表示 中断不会被
- A/B是 描述serror且和 `FEAT_DoubleFault` 相关的，暂不介绍

14 中断路由(信号流)的总结

当peripheral产生一个中断后，PE是如何跳转到某个系统中的向量表的?
如下框图展示了这一切：当一个中断到来后，中断信号交给gic，gic会进行中断的识别、优先级、affinity路由等，然后通过AXI stream将信号交给core（cpu internface），cpu interface负责标记中断是irq还是fiq，这就是中断断言了(assert了)，然后就是中断的路由规则，target到相应的EL级别，然后再检查Mask标记位，然后该中断就被taken了(即PE acknowledge了)，接下来PE还会根据EL是否发生改

变、SP_ELx使用的哪一个等信息来决定是跳转到哪一组向量表 最后PE跳转到相应的VBAR_ELx + xxx offset了。



Armv8/Armv9架构从入门到精通，Armv8/Armv9架构从入门到精通（一期），Armv8/Armv9架构从入门到精通（二期）Armv8/Armv9架构从入门到精通（三期），Arm一期、Arm二期、学习资料、免费、下载，全套资料，Secureboot从入门到精通，secureboot训练营，ATF架构从入门到精通、optee系统精讲、secureboot精讲，Trustzone/TEE/安全快速入门班，Trustzone/TEE/安全标准版，Trustzone/TEE/安全高配版。全套资料。周贺贺，baron，代码改变世界，coding_the_world，Arm精选，arm_2023，安全启动，加密启动

optee、ATF、TF-A、Trustzone、optee3.14、MMU、VMSA、cache、TLB、arm、armv8、armv9、TEE、安全、内存管理、页表，Non-cacheable,Cacheable, non-shareable,inner-shareable,outer-shareable, optee、ATF、TF-A、Trustzone、optee3.14、MMU、VMSA、cache、TLB、arm、armv8、armv9、TEE、安全、内存管理、页表…