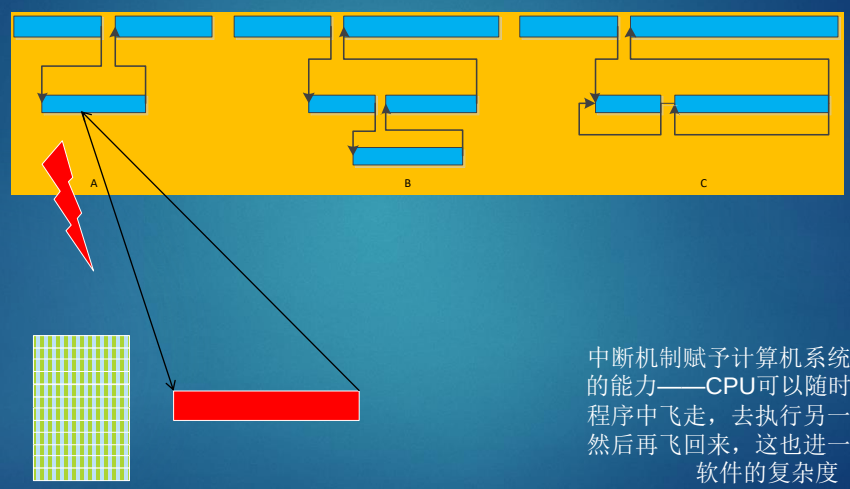
在调试器下理解ARMv8

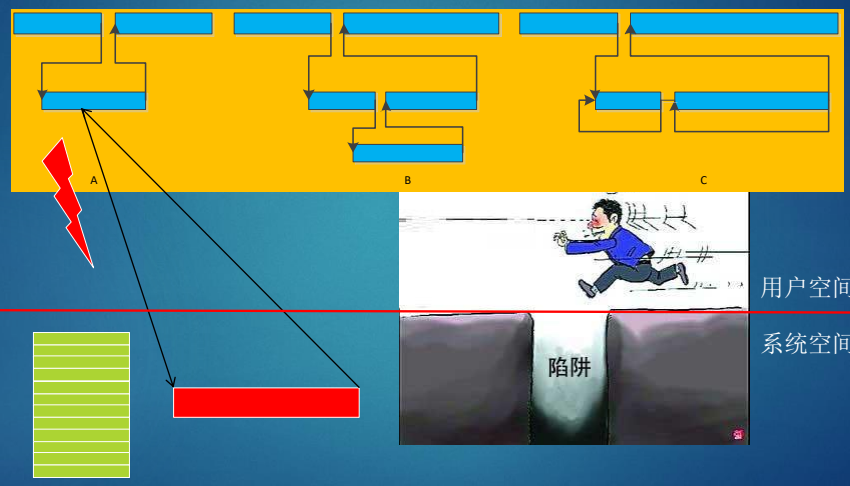
——异常和中断处理

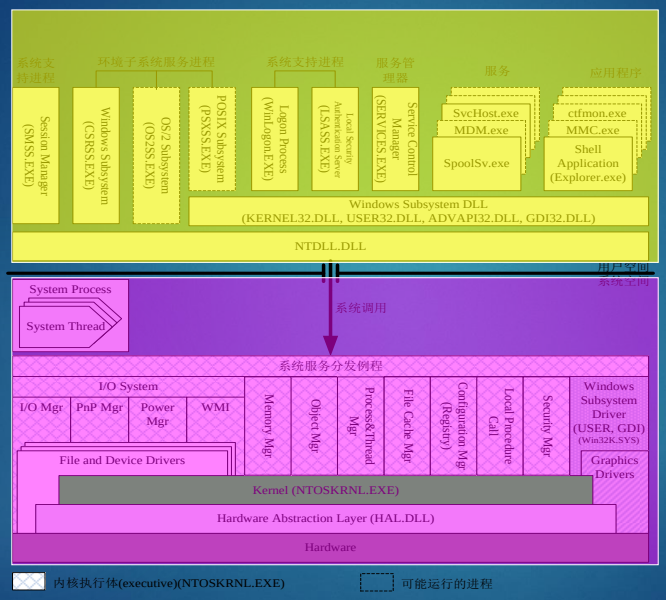
张银奎 2022-1-16 中国·上海·古鹤坡塘

背景 → 两种APIC → GIC-400 → CPU接口 → 向量表



中断机制赋予计算机系统一种飞越的能力——CPU可以随时从当前的程序中飞走，去执行另一段程序，然后再飞回来，这也进一步增加了软件的复杂度





日

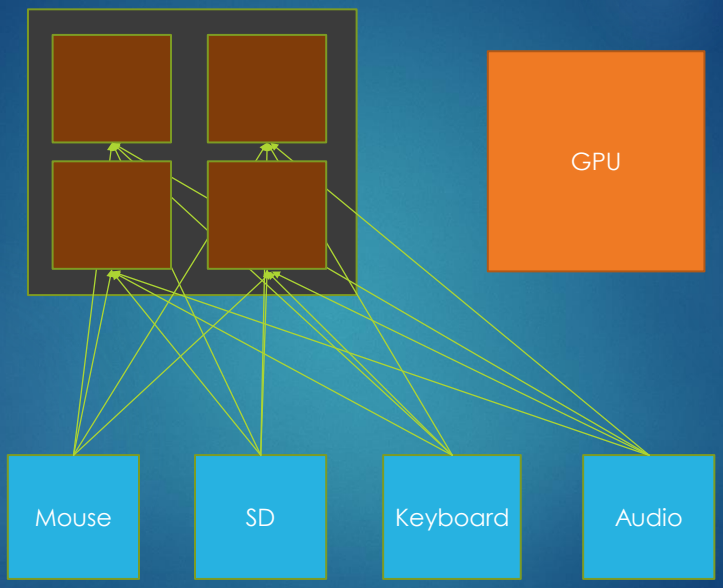
阳 ☰ 乾 天

天行健，君子自强不息

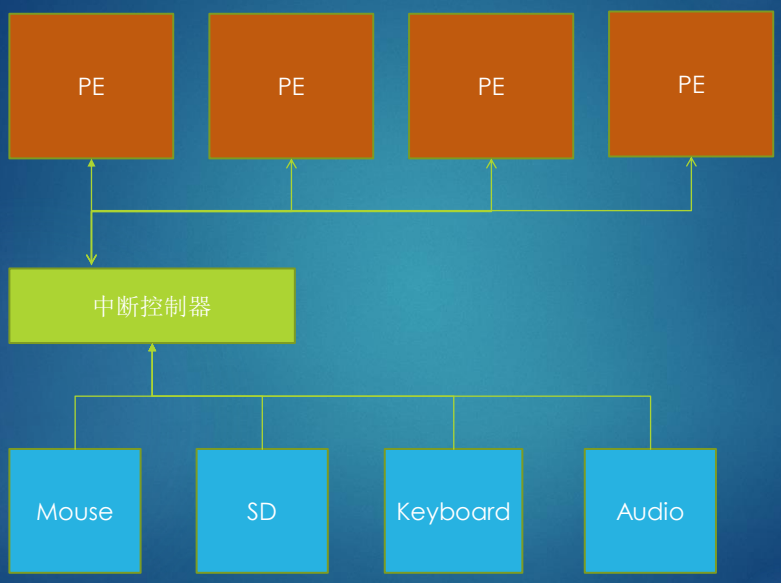
地势坤，君子厚德载物

阴 ☷ 坤 地

月



如果每个人都向CPU发中断：
- 连线会很多
- CPU不想被打扰时被打扰



CPU需要个“秘书”来管理中断

总线上应该有个“管家”来收纳和协调设备的中断

这个秘书便是中断控制器，X86上称为APIC，arm上称为GIC

C = Controller

X86上定义两种APIC

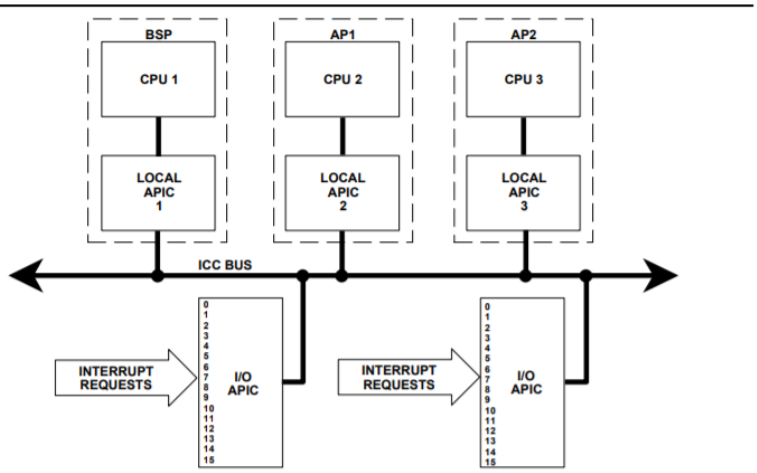
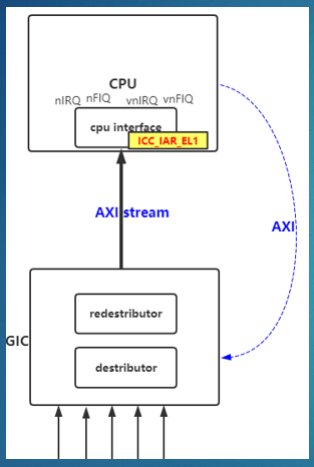


Figure 2-2. APIC Configuration



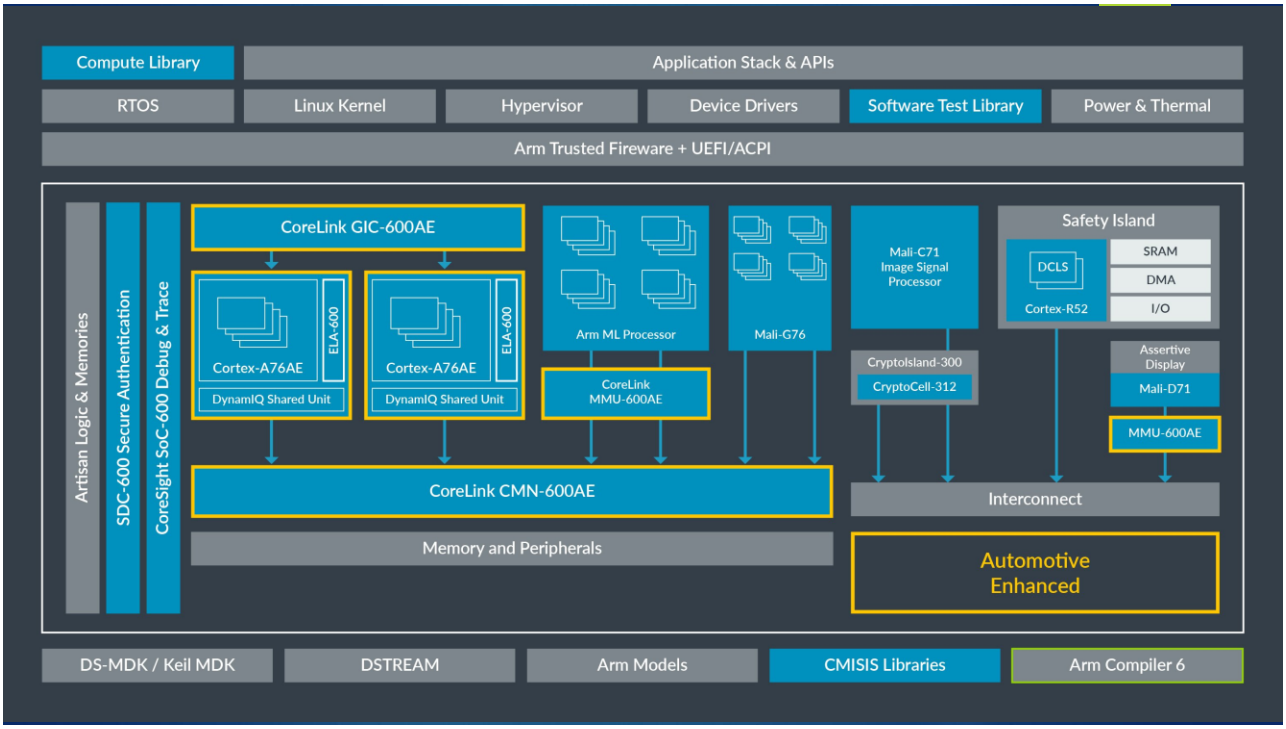
ARM上没有明确定义两种APIC
- 把CPU上的叫CPU接口
- 把总线上的叫GIC

A53 TRM中的GIC CPU Interface描述

- 9: Generic Interrupt Controller CPU Interface
  - 9.1 About the GIC CPU Interface
  - 9.2 GIC programmers model
    - 9.2.1 Memory map
    - 9.2.2 CPU interface register summary
    - 9.2.3 CPU interface register descriptions
    - 9.2.4 Virtual interface control register summary
    - 9.2.5 Virtual interface control register descriptions
    - 9.2.6 Virtual CPU interface register summary
    - 9.2.7 Virtual CPU interface register descriptions
- 10: Generic Timer

这些东西在X86上就是Local APIC

GIC = I/O APIC
GIC CPU Interface = Local APIC



Arm GIC架构规约

ARM® Generic Interrupt Controller
Architecture version 2.0

Architecture Specification

- 相当于x86的APIC手册
- 相对独立的GIC规约
- +214页

https://developer.arm.com/documentation/ihi0048/latest

GDK8中使用的A53实现的是GICv4

A53 TRM

9.1 About the GIC CPU Interface

The GIC CPU Interface, when integrated with an external distributor for supporting and managing interrupts in a cluster system. It provid

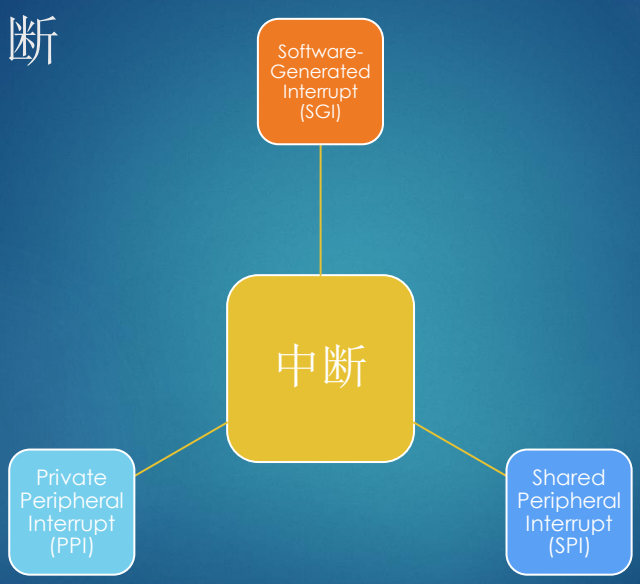
- Registers for managing:
  - Interrupt sources.
  - Interrupt behavior.
  - Interrupt routing to one or more cores.

The Cortex-A53 processor implements the GIC CPU interface as de Interrupt Controller (GICv4) architecture. This interfaces with an ex interrupt distributor component within the system.

The GICv4 architecture supports:

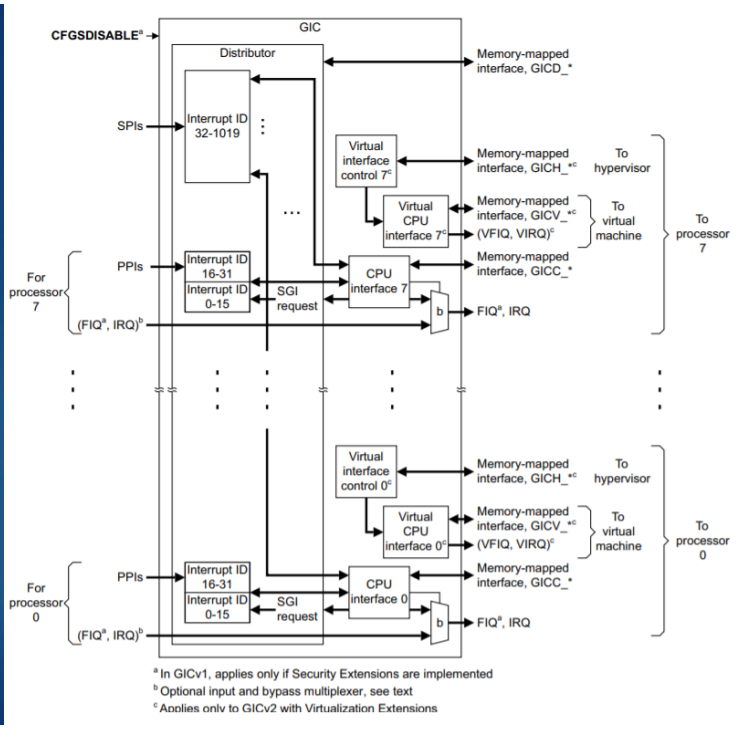
The Cortex-A53 processor implements the GIC CPU interface as described in the Generic Interrupt Controller (GICv4) architecture.

三类中断



中断ID分配

INTID	Interrupt Type	Notes
0 - 15	SGIs	Banked per PE
16 - 31	PPIs	Banked per PE
32 - 1019	SPIs	-
1020 - 1023	Special interrupt number	Used to signal special cases, see section 5.3
1024 - 8191	Reserved	-
8192 and greater	LPIs	The upper boundary is IMPLEMENTATION DEFINED



来自GIC架构规约

RK3328 TRM-Part1

Chapter 7 Generic Interrupt Controller (GIC)

7.1 Overview

There is a generic interrupt controller(GIC400) in RK3328 which generates physical interrupts to Cortex-A53. It has two interfaces, the distributor interface connects to the interrupt source, and the CPU interface connects to Cortex-A53. The details of CPU interface connectivity are shown in the following table.

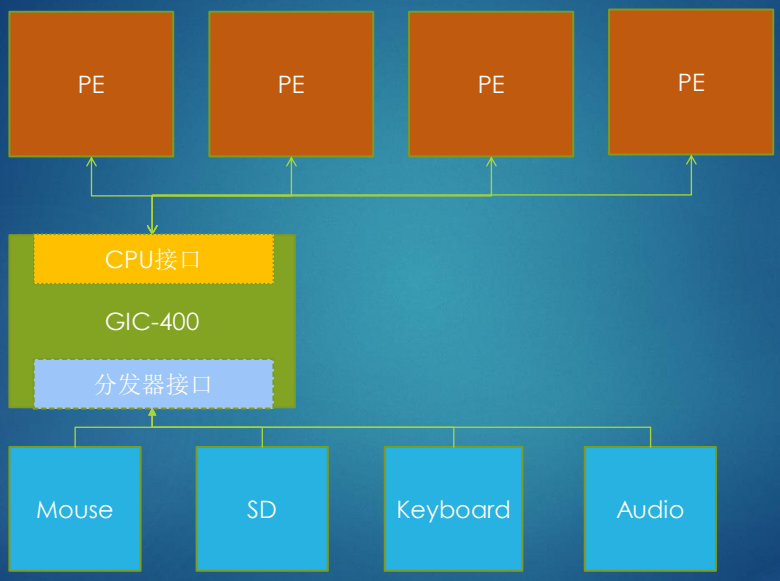
Table 1-1 CPU interface connectivity

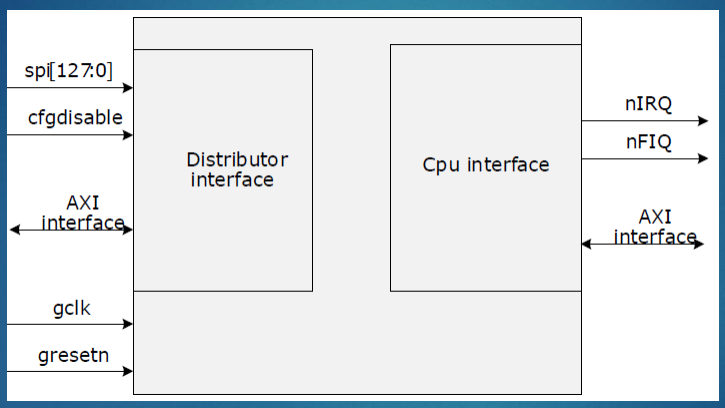
CPU Interface Number	Connectivity
CPU interface 0	CPU0
CPU interface 1	CPU1
CPU interface 2	CPU2
CPU interface 3	CPU3

It supports the following features:

- Supports 128 hardware interrupt inputs
- Masking of any interrupts
- Prioritization of interrupts
- Distribution of the interrupts to the target Cortex-A53 processor(s)
- Generation of interrupts by software
- Supports Security Extensions

GDK8的RK3328中集成的GIC是GIC-400



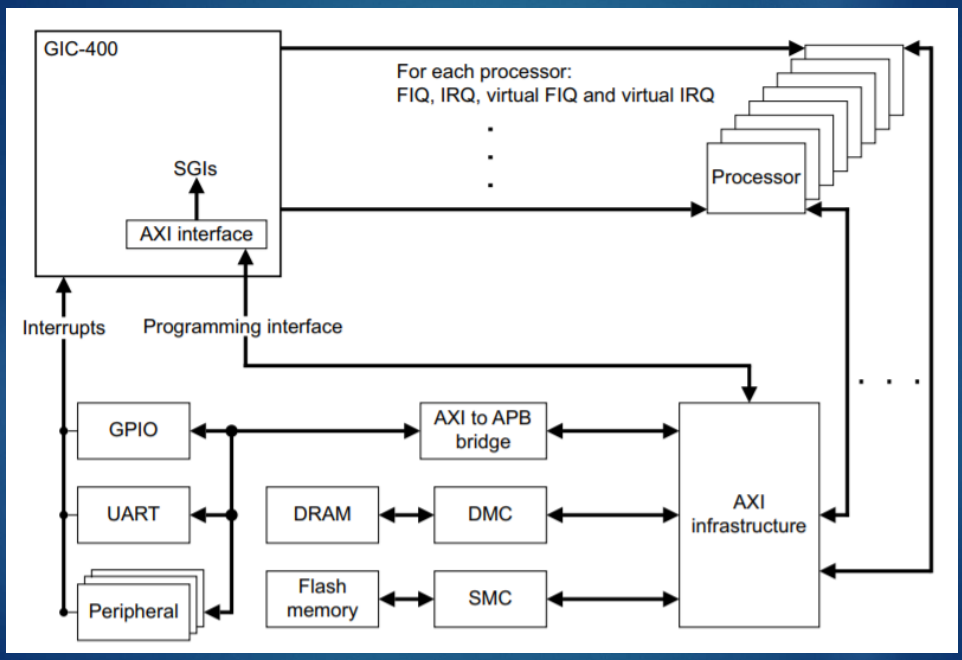


RK3288 TRM中的GIC-400框图

CoreLink™ GIC-400 Generic Interrupt Controller

Revision: r0p1

Technical Reference Manual



GIC-400 TRM中的工作原理图

```
interrupt-controller@ff811000 {
    reg = <0x0 0xff811000 0x0 0x1000
           0x0 0xff812000 0x0 0x2000
           0x0 0xff814000 0x0 0x2000
           0x0 0xff816000 0x0 0x2000>;
    interrupts = <0x1 0x9 0xf04>;
    compatible = "arm,gic-400";
    #interrupt-cells = <0x3>;
    #address-cells = <0x0>;
    phandle = <0x1>;
    interrupt-controller;
};
```

寄存器接口描述，分别为：
/* GIC Dist */
/* GIC CPU */
/* GIC VCPU Control */
/* GIC VCPU */

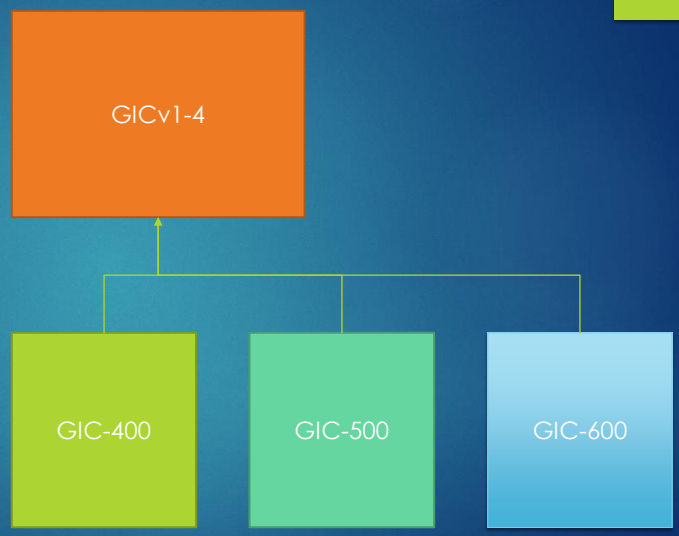
在父GIC上的中断源，也叫GIC Maintenence IRQ

配置每个中断源的参数个数，三个，见下一页

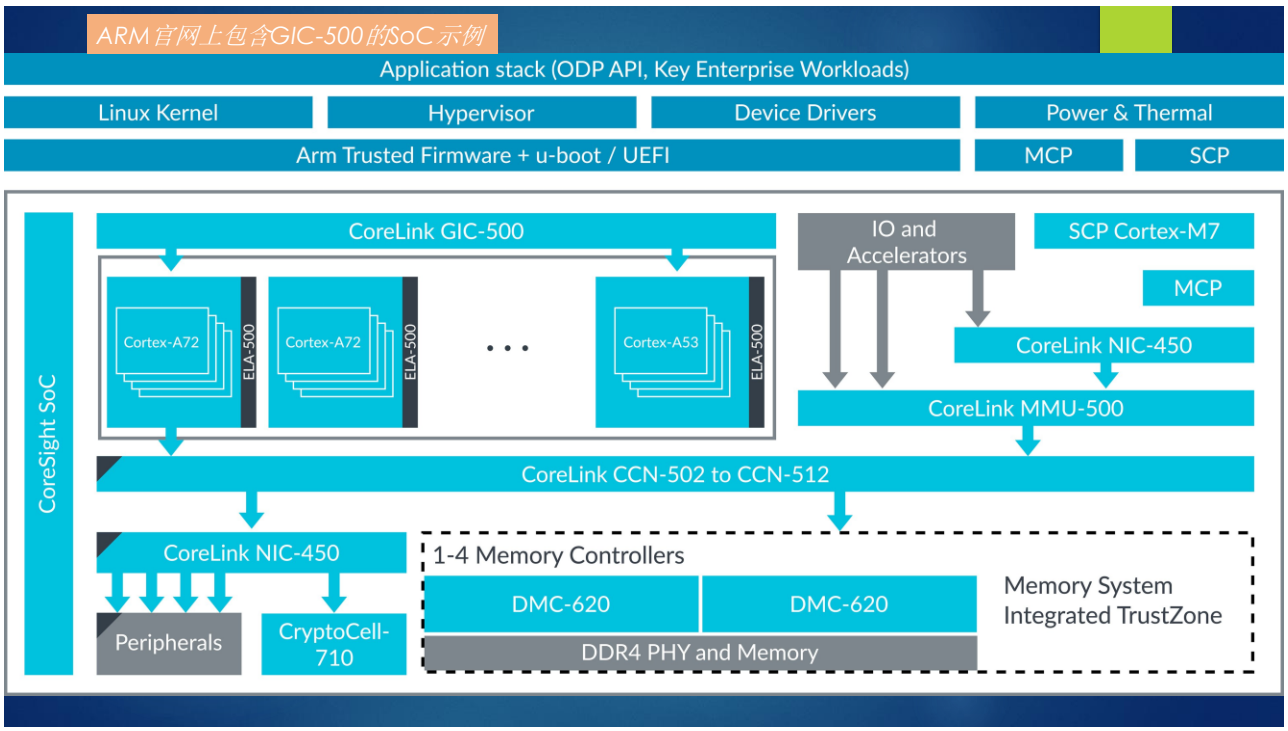
来自GDK8的DTS文件

GIC-400是对GIC架构规约的一种实现

GICv1-4是宏架构，给上层软件的接口，长期稳定，手册叫ARM GIC-400是微架构，考虑硬件成本和场景的一种实现，手册叫**TRM**



ARM官网上包含GIC-500的SoC示例



图中术语

- CCN - **Cache Coherent Network**
- CoreLink – Core Link
- GIC - Generic Interrupt Controller
- ITS - Interrupt Translation Services

developer.arm.com/ip-products/system-ip/corelink-interconnect

arm Developer

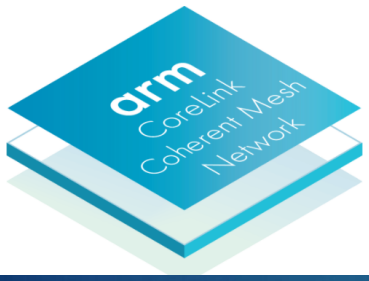
Documentation Downloads Community Support

IP Products Tools and Software Architectures Solutions

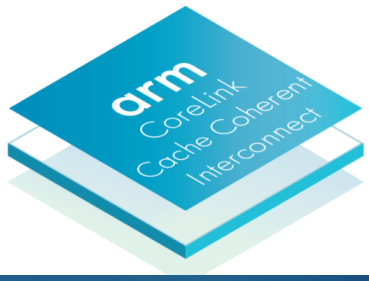
Overview CoreLink Cache Coherent Interconnect Family CoreLink Cache Coherent Network Family CoreLink Coherent Mesh Network Family

CoreLink Network Interconnect Family CoreLink SIE-200 CoreLink SIE-300 CoreLink XHB-500

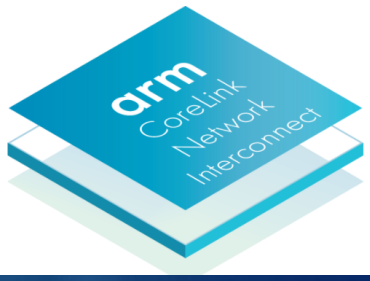
CoreLink Coherent Mesh Network

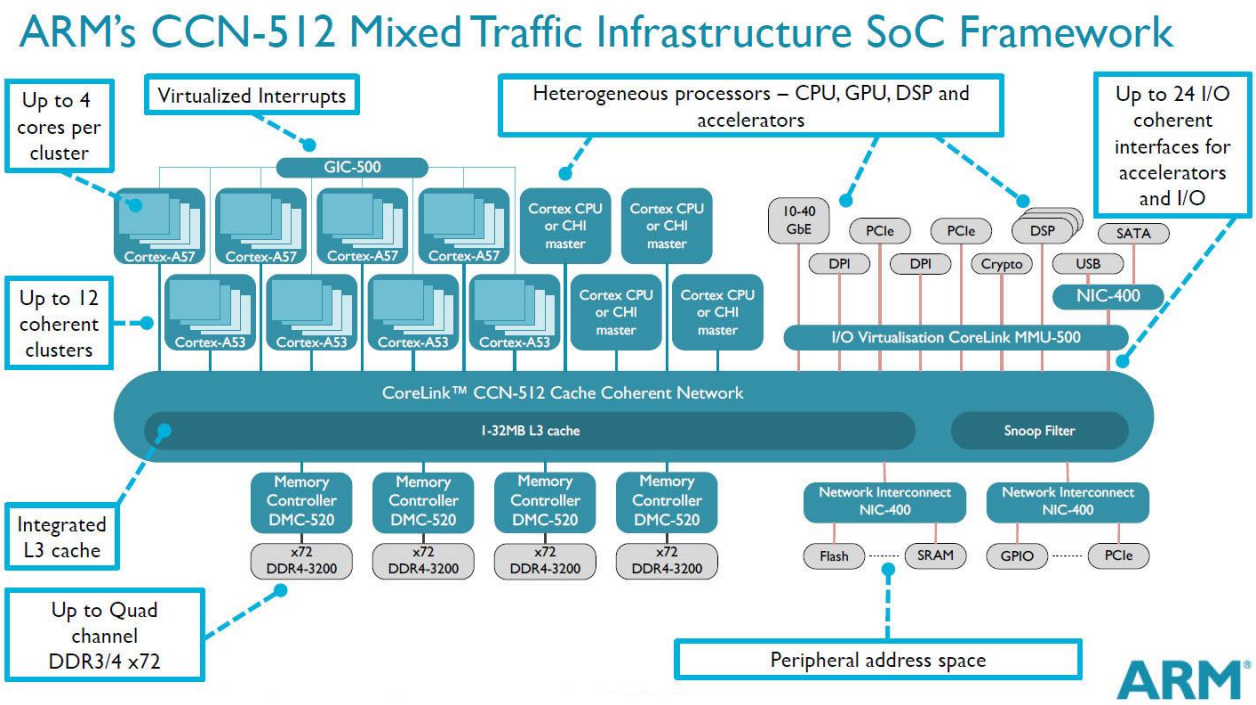


CoreLink Cache Coherent Interconnect



CoreLink Network Interconnect





drivers > irqchip > C irq-gic-v3.c

```
1  /*
```

- C irq-gic-common.c
- C irq-gic-common.h
- C irq-gic-pm.c
- C irq-gic-realview.c
- C irq-gic-v2m.c
- C irq-gic-v3-its-pci-msi.c
- C irq-gic-v3-its-platform-msi.c
- C irq-gic-v3-its.c
- C irq-gic-v3.c
- C irq-gic-v4.c
- C irq-gic.c

GIC的管理代码，
或者说GIC本身
的驱动

配置中断源

```
serial@ff110000 {
        reg = <0x0 0xff11000
        dmas = <0xc 0x2 0xc
        interrupts = <0x0 0x37 0x4>;
        pinctrl-0 = <0x35 0x36>;
        reg-shift = <0x2>;
        compatible = "rockchip,rk3328 apb-uart";
        clock-names = "baudclk", "apl
        clocks = <0x2 0x26 0x2 0xd2>;
        status = "okay";
        reg-io-width = <0x4>;
        phandle = <0xa7>;
        pinctrl-names = "default";
};
```

中断类型，0代表SPI，1代表PPI

4 = active high level-sensitive

中断源ID

以GPIO形式配置中断源

```
wireless-bluetooth {
        pinctrl-0 = <0x99>;
        pinctrl-1 = <0x9a>;
        uart_rts_gpios = <0x6e 0xa 0x1>;
        compatible = "bluetooth-platdata";
        clock-names = "ext_clock";
        clocks = <0x2 0x1e>;
        BT,wake_gpio = <0x6e 0x17 0x0>;
        BT,reset_gpio = <0x6e 0x15 0x0>;
        status = "okay";
        pinctrl-names = "default", "rts_gpio";
        BT,wake_host_irq = <0x6e 0x1a 0x0>;
};
```

蓝牙和WIFI无线模块

蓝牙模块通过GPIO管脚连到SoC，触发中断

```
 33:          0          0          0          0     GICv2  53 Level     eth0
 42:       4950          0          0          0     GICv2  64 Level     ff370000.vop
105:          0          0          0          0     gpio1  26 Edge      bt_default_wake_host_irq
144:      57485          0          0          0     gpio3   1 Level     bcmsdh_sdmmc
175:          0          0          0          0     GICv2  39 Level     ff360000.rkvdec
```

```
geduer@gdk8:~$ cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
  1:          0          0          0          0     GICv2  29 Edge      arch_timer
  2:     132012     140116      97041     182397     GICv2  30 Edge      arch_timer
  5:        931          0          0          0     GICv2  32 Level     ff1f0000.dmac
  6:          0          0          0          0     GICv2  33 Level     ff1f0000.dmac
  9:          0          0          0          0     GICv2  48 Level     ehci_hcd:usb2
 10:          0          0          0          0     GICv2  49 Level     ohci_hcd:usb3
 11:          0          0          0          0     GICv2  90 Level     rockchip_thermal
 12:          0          0          0          0     GICv2  41 Level     ff350000.vpu_service, ff351000.avsd
 14:          0      34067          0          0     GICv2  82 Level     rk_pwm_irq
 16:       4128          0          0          0     GICv2  67 Level     ff3c0000.hdmi, dw-hdmi-cec
 18:          0          0          0          0     GICv2  44 Level     dw-mci
 19:      64883          0          0          0     GICv2  46 Level     dw-mci
 20:       1190          0          0          0     GICv2  87 Level     serial
 21:     277893          0          0          0     GICv2  36 Level     dw-mci
 22:          0          0          0          0     GICv2 122 Level     Mali_GP
 23:          0          0          0          0     GICv2 119 Level     Mali_GP_MMU
 24:          0          0          0          0     GICv2 125 Level     Mali_PP_Broadcast
 25:          0          0          0          0     GICv2 120 Level     Mali_PP0
 26:          0          0          0          0     GICv2 121 Level     Mali_PP0_MMU
 27:          0          0          0          0     GICv2 123 Level     Mali_PP1
 28:          0          0          0          0     GICv2 124 Level     Mali_PP1_MMU
 29:          0          0          0          0     GICv2 112 Level     ff280000.saradc
 30:          0          0          0          0     GICv2 115 Level     ff430000.hdmiphy
 31:          0          0          0          0     GICv2 127 Level     ff330000.h265e
 32:          0          0          0          0     GICv2 159 Edge      debug-signal
 33:          0          0          0          0     GICv2  53 Level     eth0
 42:       4950          0          0          0     GICv2  64 Level     ff370000.vop
105:          0          0          0          0     gpio1  26 Edge      bt_default_wake_host_irq
144:      33864          0          0          0     gpio3   1 Level     bcmsdh_sdmmc
175:          0          0          0          0     GICv2  39 Level     ff360000.rkvdec
176:          0          0          0          0     GICv2 129 Level     ff340000.vepu
177:          0          0          0          0     GICv2 109 Level     rockchip_u3phy
178:          2          0          0          0     GICv2  55 Level     ff580000.usb, ff580000.usb, dwc2_hsotg:usb1
179:          0          0          0          0     GICv2  89 Level     debug
180:          0          0          0          0     GICv2  91 Level     rockchip_usb2phy_bvalid
181:          0          0          0          0     GICv2  93 Level     rockchip_usb2phy
182:          0          0          0          0     GICv2  92 Level     rockchip_usb2phy_id
183:          0          0          0          0     GICv2  94 Level     rockchip_usb2phy
184:       2709          0          0          0     GICv2  99 Level     xhci-hcd:usb4
```

```
[    0.000000] GIC: Using split EOI/Deactivate mode
```

```
[    0.000000] NR_IRQS:64 nr_irqs:64 0
[    0.180409] genirq: Setting trigger mode 8 for irq 32 failed (gic_set_type+0x0/0x64)
[    0.528883] ff110000.serial: ttyS0 at MMIO 0xff110000 (irq = 20, base_baud = 1500000) is a 16550A
[    0.819068] dwc2 ff580000.usb: irq 178, io mem 0xff580000
[    1.471936] ehci-platform ff5c0000.usb: irq 9, io mem 0xff5c0000
[    1.488440] ohci-platform ff5d0000.usb: irq 10, io mem 0xff5d0000
[    1.553017] xhci-hcd xhci-hcd.9.auto: irq 184, io mem 0xff600000
[    1.623229] dwmmc_rockchip ff500000.dwmmc: DW MMC controller at irq 18,32 bit host data width,256 deep fifo
[    1.650944] dwmmc_rockchip ff520000.dwmmc: DW MMC controller at irq 19,32 bit host data width,256 deep fifo
[    2.175684] dwmmc_rockchip ff5f0000.dwmmc: DW MMC controller at irq 21,32 bit host data width,256 deep fifo
[    2.233655] of_get_named_gpiod_flags: parsed 'WIFI,host_wake_irq' property of node '/wireless-wlan[0]' - status (0
[    2.233663] [WLAN_RFKILL]: wlan_platdata_parse_dt: get property: WIFI,host_wake_irq = 97, flags = 0.
[    2.238390] of_get_named_gpiod_flags: parsed 'BT,wake_host_irq' property of node '/wireless-bluetooth[0]' - status
[    2.238394] [BT_RFKILL]: bluetooth_platdata_parse_dt: get property: BT,wake_host_irq = 58.
[    2.238478] [BT_RFKILL]: Request irq for bt wakeup host
[    2.238520] [BT_RFKILL]: ** disable irq and enable wake
[    2.238528] [BT_RFKILL]: ** irq wake (105) is enabled on GPIO 58
[    7.526901] [WLAN_RFKILL]: rockchip_wifi_get_oob_irq: Enter
[    7.526912] dhd_wlan_init_gpio: WL_HOST_WAKE=-1, oob_irq=144, oob_irq_flags=0x414
[    7.881570] bcmsdh_oob_intr_unregister: irq is not registered
[    8.671179] bcmsdh_oob_intr_register: HW_OOB irq=144 flags=0x4
```

晃动GDK8 USB 3.0口上的鼠标，这个数字会变化

此处的4是流水号，代表是系统里的第4个USB设备

```
184:        2709          0          0          0     GICv2  99 Level     xhci-hcd:usb4
```

[2.008771] xhci-hcd xhci-hcd.9.auto: irq 184, io mem 0xff600000

从DTS中获取配置

```
irq = platform_get_irq_byname(dwc3_pdev, "dwc_usb3");
irq = platform_get_irq(dwc3_pdev, 0);
```

drivers > usb > dwc3 > C host.c > dwc3_host_init(dwc3 *)

```
int dwc3_host_init(struct dwc3 *dwc)
{
```

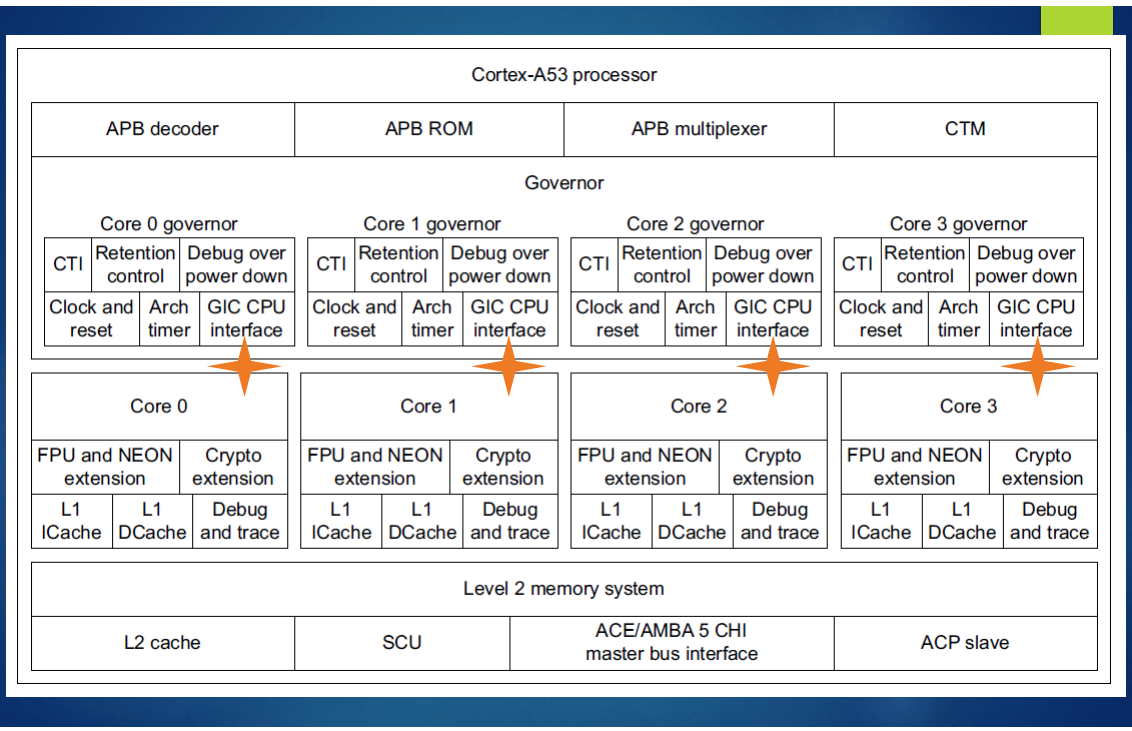
```
dwc->xhci_resources[1].start = irq;
dwc->xhci_resources[1].end = irq;
dwc->xhci_resources[1].flags = res->flags;
dwc->xhci_resources[1].name = res->name;

ret = platform_device_add_resources(xhci, dwc->xhci_resources,
                                    DWC3_XHCI_RESOURCES_NUM);

ret = platform_device_add(xhci);
```

arch > arm64 > boot > dts > arm > foundation-v8.dts

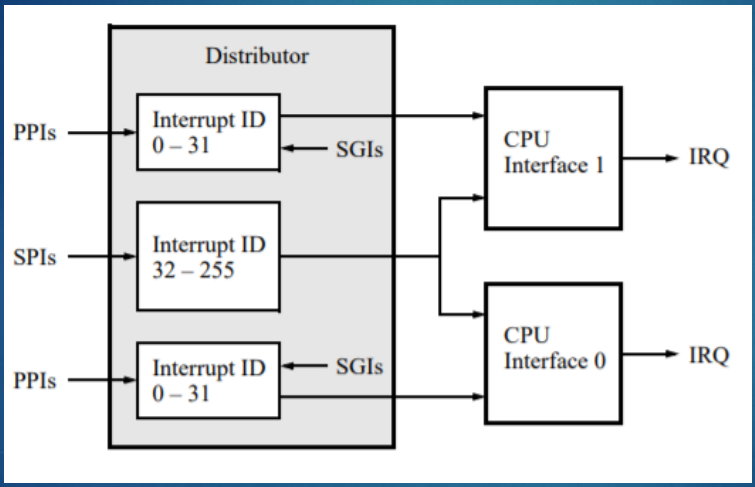
```
    };

    smb {
        compatible = "arm,vexpress,v2m-p1", "simple-bus";
        arm,v2m-memory-map = "rs1";
        #address-cells = <2>; /* SMB chipselect number and offset */
        #size-cells = <1>;

        ranges = <0 0 0 0x08000000 0x04000000>,
             <1 0 0 0x14000000 0x04000000>,
             <2 0 0 0x18000000 0x04000000>,
             <3 0 0 0x1c000000 0x04000000>,
             <4 0 0 0x0c000000 0x04000000>,
             <5 0 0 0x10000000 0x04000000>;

        #interrupt-cells = <1>;
        interrupt-map-mask = <0 0 63>;
        interrupt-map = <0 0  0 &gic 0  0 4>,
                <0 0  1 &gic 0  1 4>,
                <0 0  2 &gic 0  2 4>,
                <0 0  3 &gic 0  3 4>,
                <0 0  4 &gic 0  4 4>,
                <0 0  5 &gic 0  5 4>,
                <0 0  6 &gic 0  6 4>,
                <0 0  7 &gic 0  7 4>,
                <0 0  8 &gic 0  8 4>,
                <0 0  9 &gic 0  9 4>,
                <0 0 10 &gic 0 10 4>,
                <0 0 11 &gic 0 11 4>,
```

中断映射，可以通过DTS定义规则，也可以在操作系统中设置中断亲缘性，Linux内核也有irqbalance服务（GDK8中没有启用）


背景
两种 APIC
GIC-400
CPU接口
向量表


Cortex-A53 processor
APB decoder
APB ROM
APB multiplexer
CTM
Governor
Core 0 governor
Core 1 governor
Core 2 governor
Core 3 governor
CTI
Retention control
Debug over power down
Clock and reset
Arch timer
GIC CPU interface
Core 0
Core 1
Core 2
Core 3
FPU and NEON extension
Crypto extension
L1 ICache
L1 DCache
Debug and trace
Level 2 memory system
L2 cache
SCU
ACE/AMBA 5 CHI master bus interface
ACP slave

CPU接口



PPI - private peripherals interrupts
SPI – Shared peripherals interrupts

https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Tutorials/Using_GIC.pdf

Offset	Name	Type	Reset	Description
0x0000	GICC_CTLR	RW	0x00000000	CPU Interface Control Register
0x0004	GICC_PMR	RW	0x00000000	Interrupt Priority Mask Register
0x0008	GICC_BPR	RW	0x00000002 (S)[a] 0x00000003 (NS)[b]	Binary Point Register
0x000C	GICC_IAR	RO	-	Interrupt Acknowledge Register
0x0010	GICC_EOIR	WO	-	End Of Interrupt Register
0x0014	GICC_RPR	RO	0x000000FF	Running Priority Register
0x0018	GICC_HPPIR	RO	0x000003FF	Highest Priority Pending Interrupt Register
0x001C	GICC_ABPR	RW	0x00000003	Aliased Binary Point Register
0x0020	GICC_AIAR	RO	-	Aliased Interrupt Acknowledge Register
0x0024	GICC_AEOIR	WO	-	Aliased End of Interrupt Register
0x0028	GICC_AHPPIR	RO	0x000003FF	Aliased Highest Priority Pending Interrupt Register
0x00D0	GICC_APR0	RW	0x00000000	*Active Priority Register* on page 9-6
0x00E0	GICC_NSAPR0	RW	0x00000000	Non-secure Active Priority Register
0x00FC	GICC_IIDR	RO	0x0034443B	*CPU Interface Identification Register* on page 9-7
0x1000	GICC_DIR	WO	-	Deactivate Interrupt Register

CPU接口寄存器

来自A53 TRM

两种访问方式

通过内存映射访问

Address	31 ... 10	9 8	7 ... 1	0	Register name
0xFFFEC100	Unused			E	ICCICR
0xFFFEC104	Unused		Priority		ICCPMR
0xFFFEC10C	Unused	Interrupt ID			ICCIAR
0xFFFEC110	Unused	Interrupt ID			ICCEOIR

- CPU通过虚拟地址访问，x86中称为Local APIC，每个CPU有自己的一份实例

4.4.2 Interrupt Priority Mask Register, GICC_PMR

The GICC_PMR characteristics are:

Purpose Provides an interrupt priority filter. Only interrupts with higher priority than the value in this register are signaled to the processor.

Note

Higher priority corresponds to a lower Priority field value.

Usage constraints If the GIC implements the Security Extensions then:

- a Non-secure access to this register can only read or write a value that corresponds to the lower half of the priority range, see *Interrupt grouping and interrupt prioritization* on page 3-53.
- if a Secure write has programmed the GICC_PMR with a value that corresponds to a value in the upper half of the priority range then:
  - any Non-secure read of the GICC_PMR returns `0x00`, regardless of the value held in the register
  - any Non-secure write to the GICC_PMR is ignored.

For more information see *Non-secure access to register fields for Group 0 interrupt priorities* on page 4-81.

When determining interrupt preemption, the priority value can be split into two parts, using the Binary Point register, GICC_BPR.

Configurations This register is available in all configurations of the GIC. If the GIC implements the Security Extensions, this register is Common.

Attributes See the register summary in Table 4-2 on page 4-76.

Figure 4-25 shows the GICC_PMR bit assignments.

31 … 8	7 … 0
Reserved	Priority

CBAR_EL1

- GIC内存映射寄存器的基地址寄存器
- Configuration Base Address Register
- EL1或者更高级别才能访问

31 … 15	14 … 8	7 … 0
PERIPHBASE[31:15]	Reserved	PERIPHBASE[39:32]

ARM Cortex-A15 MPCore Processor Technical Reference Manual

中断和异常向量表

Offset from VBAR_EL1	Exception type	Exception set level
+0x000	Synchronous	Current EL with SP0
+0x080	IRQ/vIRQ	“
+0x100	FIQ/vFIQ	“
+0x180	SError/vSError	“
+0x200	Synchronous	Current EL with SPx
+0x280	IRQ/vIRQ	“
+0x300	FIQ/vFIQ	“
+0x380	SError/vSError	“
+0x400	Synchronous	Lower EL using ARM64
+0x480	IRQ/vIRQ	“
+0x500	FIQ/vFIQ	“
+0x580	SError/vSError	“
+0x600	Synchronous	Lower EL with ARM32
+0x680	IRQ/vIRQ	“
+0x700	FIQ/vFIQ	“
+0x780	SError/vSError	“

arch/arm64/kernel/entry.S

```
ENTRY(vectors)
ventry el1_sync_invalid // Synchronous EL1t
ventry el1_irq_invalid // IRQ EL1t
ventry el1_fiq_invalid // FIQ EL1t
ventry el1_error_invalid // Error EL1t

ventry el1_sync // Synchronous EL1h
ventry el1_irq // IRQ EL1h
ventry el1_fiq_invalid // FIQ EL1h
ventry el1_error_invalid // Error EL1h

ventry el0_sync // Synchronous 64-bit EL0
ventry el0_irq // IRQ 64-bit EL0
ventry el0_fiq_invalid // FIQ 64-bit EL0
ventry el0_error_invalid // Error 64-bit EL0

#ifdef CONFIG_COMPAT
ventry el0_sync_compat // Synchronous 32-bit EL0
ventry el0_irq_compat // IRQ 32-bit EL0
ventry el0_fiq_invalid_compat // FIQ 32-bit EL0
ventry el0_error_invalid_compat // Error 32-bit EL0
#else
ventry el0_sync_invalid // Synchronous 32-bit EL0
ventry el0_irq_invalid // IRQ 32-bit EL0
ventry el0_fiq_invalid // FIQ 32-bit EL0
ventry el0_error_invalid // Error 32-bit EL0
#endif
END(vectors)
```

每个ventry占32个字节，8条指令

* https://eastrivervillage.com/Anatomy-of-Linux-system-call-in-ARM64/

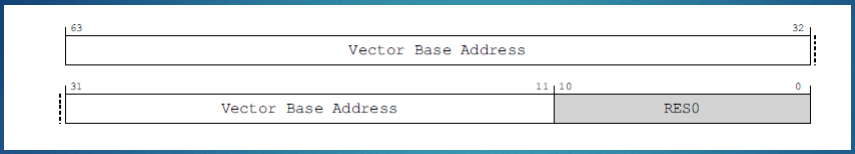
加载向量表

```
adr_l   x8, vectors          // load VBAR_EL1 with virtual
msr     vbar_el1, x8         // vector table address
Isb   // instruction sync barrier
```

```
void ge_arm_sysregs(void)
{
    rd_arm_reg(ID_AA64ISAR0_EL1);
    rd_arm_reg(ID_AA64ISAR1_EL1);
    rd_arm_reg(ID_AA64MMFR0_EL1);
    rd_arm_reg(ID_AA64MMFR1_EL1);
    rd_arm_reg(ID_AA64PFR0_EL1);
    rd_arm_reg(ID_AA64PFR1_EL1);
    rd_arm_reg(ID_AA64DFR0_EL1);
    rd_arm_reg(ID_AA64DFR1_EL1);

    rd_arm_reg(MIDR_EL1);
    rd_arm_reg(MPIDR_EL1);
    rd_arm_reg(REVIDR_EL1);
    rd_arm_reg(VBAR_EL1);
    /* Unexposed register access causes SIGILL */
    rd_arm_reg(ID_MMFR0_EL1);
}
```

VBAR

Vector Base Address Register



Bits	Name	Function
[31:5]	VBA	Bits[31:5] of the base address of the low exception vectors.
[4:0]	-	Reserved.

AArch64 System Register Descriptions
D13.2 General system control registers

D13.2.140 VBAR_EL1, Vector Base Address Register (EL1)

```
el0_sync:
kernel_entry 0
mrs x25, esr_el1 // 把异常症状寄存器读到x25
lsr x24, x25, #ESR_ELx_EC_SHIFT // 把x25左移26位，得到的值（exception class）放入x24
cmp x24, #ESR_ELx_EC_SVC64 // SVC in 64-bit state
b.eq el0_svc
cmp x24, #ESR_ELx_EC_DABT_LOW // data abort in EL0
b.eq el0_da
cmp x24, #ESR_ELx_EC_IABT_LOW // instruction abort in EL0
b.eq el0_ia
cmp x24, #ESR_ELx_EC_FP_ASIMD // FP/ASIMD access
b.eq el0_fpsimd_acc
cmp x24, #ESR_ELx_EC_FP_EXC64 // FP/ASIMD exception
b.eq el0_fpsimd_exc
cmp x24, #ESR_ELx_EC_SYS64 // configurable trap
b.eq el0_sys
cmp x24, #ESR_ELx_EC_SP_ALIGN // stack alignment exception
b.eq el0_sp_pc
cmp x24, #ESR_ELx_EC_PC_ALIGN // pc alignment exception
b.eq el0_sp_pc
cmp x24, #ESR_ELx_EC_UNKNOWN // unknown exception in EL0
b.eq el0_undef
cmp x24, #ESR_ELx_EC_BREAKPT_LOW // debug exception in EL0
b.ge el0_dbg
b el0_inv
```

// arch/arm64/kernel/entry.S

```
[27948.980226] ID_AA64ISAR0_EL1    : 0x0000000000011120
[27948.980232] ID_AA64ISAR1_EL1    : 0x0000000000000000
[27948.980237] ID_AA64MMFR0_EL1    : 0x0000000000001122
[27948.980243] ID_AA64MMFR1_EL1    : 0x0000000000000000
[27948.980248] ID_AA64PFR0_EL1     : 0x0000000000002222
[27948.980254] ID_AA64PFR1_EL1     : 0x0000000000000000
[27948.980259] ID_AA64DFR0_EL1     : 0x0000000010305106
[27948.980265] ID_AA64DFR1_EL1     : 0x0000000000000000
[27948.980270] MIDR_EL1            : 0x00000000410fd034
[27948.980275] MPIDR_EL1           : 0x0000000080000003
[27948.980281] REVIDR_EL1          : 0x0000000000000180
[27948.980286] VBAR_EL1            : 0xffffff8008081800
[27948.980292] ID_MMFR0_EL1        : 0x0000000010201105
```

ESR_EL1 Exception Syndrome Register

Bits	Name	Function
[31:26]	EC	Exception Class: 0b100000 Instruction Abort that caused entry from a lower Exception level in AArch32 or AArch64. 0b100001 Instruction Abort that caused entry from a current Exception level in AArch64.
[25]	IL	Instruction Length for synchronous exceptions.
[24:10]	-	Reserved, RES0.
[9]	EA	External abort type. This bit indicates whether an AXI decode or slave error caused an abort. The possible values are: **0** External abort marked as DECERR. **1** External abort marked as SLVERR. For aborts other than external aborts this bit always returns 0.
[8]	-	Reserved, RES0.
[7]	S1PTW	When 1, indicates the instruction fault came from a second stage fault during a first stage translation table walk.
[6]	-	Reserved, RES0.
[5:0]	IFSC	Instruction Fault Status Code. This field indicates the type of exception generated. The possible values are:

```
#define ESR_ELx_EC_UNKNOWN   (0x00)
#define ESR_ELx_EC_WFx       (0x01)
/* Unallocated EC: 0x02 */
#define ESR_ELx_EC_CP15_32   (0x03)
#define ESR_ELx_EC_CP15_64   (0x04)
#define ESR_ELx_EC_CP14_MR   (0x05)
#define ESR_ELx_EC_CP14_LS   (0x06)
#define ESR_ELx_EC_FP_ASIMD  (0x07)
#define ESR_ELx_EC_CP10_ID   (0x08)   /* EL2 only */
#define ESR_ELx_EC_PAC       (0x09)   /* EL2 and above */
/* Unallocated EC: 0x0A - 0x0B */
#define ESR_ELx_EC_CP14_64   (0x0C)
/* Unallocated EC: 0x0d */
#define ESR_ELx_EC_ILL       (0x0E)
/* Unallocated EC: 0x0F - 0x10 */
#define ESR_ELx_EC_SVC32     (0x11)
#define ESR_ELx_EC_HVC32     (0x12)   /* EL2 only */
#define ESR_ELx_EC_SMC32     (0x13)   /* EL2 and above */
/* Unallocated EC: 0x14 */
#define ESR_ELx_EC_SVC64     (0x15)
#define ESR_ELx_EC_HVC64     (0x16)   /* EL2 and above */
#define ESR_ELx_EC_SMC64     (0x17)   /* EL2 and above */
#define ESR_ELx_EC_SYS64     (0x18)
#define ESR_ELx_EC_SVE       (0x19)
```

```
#define ESR_ELx_EC_IMP_DEF      (0x1f)    /* EL3 only */
#define ESR_ELx_EC_IABT_LOW     (0x20)
#define ESR_ELx_EC_IABT_CUR     (0x21)
#define ESR_ELx_EC_PC_ALIGN     (0x22)
/* Unallocated EC: 0x23 */
#define ESR_ELx_EC_DABT_LOW     (0x24)
#define ESR_ELx_EC_DABT_CUR     (0x25)
#define ESR_ELx_EC_SP_ALIGN     (0x26)
/* Unallocated EC: 0x27 */
#define ESR_ELx_EC_FP_EXC32     (0x28)
/* Unallocated EC: 0x29 - 0x2B */
#define ESR_ELx_EC_FP_EXC64     (0x2C)
/* Unallocated EC: 0x2D - 0x2E */
#define ESR_ELx_EC_SERROR (0x2F)
#define ESR_ELx_EC_BREAKPT_LOW    (0x30)
#define ESR_ELx_EC_BREAKPT_CUR    (0x31)
#define ESR_ELx_EC_SOFTSTP_LOW    (0x32)
#define ESR_ELx_EC_SOFTSTP_CUR    (0x33)
#define ESR_ELx_EC_WATCHPT_LOW    (0x34)
#define ESR_ELx_EC_WATCHPT_CUR    (0x35)
#define ESR_ELx_EC_BKPT32   (0x38)
#define ESR_ELx_EC_VECTOR32   (0x3A)    /* EL2 only */
#define ESR_ELx_EC_BRK64    (0x3C)
#define ESR_ELx_EC_MAX          (0x3F)
```

\arch\arm64\include\asm\esr.h

```
#define ESR_ELx_EC_SHIFT      (26)
#define ESR_ELx_EC_SVC64    (0x15)
```

系统服务入口

```
/*
* SVC handler.
*/
.align 6
el0_svc:
adrp stbl, sys_call_table // load syscall table pointer
uxtw scno, w8 // syscall number in w8
mov sc_nr, #__NR_syscalls
el0_svc_naked: // compat entry point
stp x0, scno, [sp, #S_ORIG_X0] // save the original x0 and syscall number enable_dbg_and_irq
ct_user_exit 1

ldr x16, [tsk, #TI_FLAGS] // check for syscall hooks
tst x16, #_TIF_SYSCALL_WORK
b.ne __sys_trace
cmp scno, sc_nr // check upper syscall limit
b.hs ni_sys
ldr x16, [stbl, scno, lsl #3] // 将服务号左移3位，得到偏移，加上基地址，得到address in the syscall table
blr x16 // call sys_* routine
b ret_fast_syscall
ni_sys:
mov x0, sp
bl do_ni_syscall
b ret_fast_syscall
ENDPROC(el0_svc)
```

寄存器别名

```
/*
* These are the registers used in the syscall handler, and allow us to
* have in theory up to 7 arguments to a function - x0 to x6.
*
* x7 is reserved for the system call number in 32-bit mode.
*/
sc_nr .req x25 // number of system calls
scno .req x26 // syscall number
stbl .req x27 // syscall table pointer
tsk .req x28 // current thread_info
```

系统服务表

```
#undef __SYSCALL
#define __SYSCALL(nr, sym) [nr] = sym,

/*
* The sys_call_table array must be 4K aligned to be accessible from
* kernel/entry.S.
*/
void * const sys_call_table[__NR_syscalls] __aligned(4096) = {
[0 ... __NR_syscalls - 1] = sys_ni_syscall,
#include <asm/unistd.h>
};
```

arch/arm64/kernel/sys.c

调用系统服务

```
0x7fb7f407a8 <__GI___libc_write> stp x29, x30, [sp, #-48]!
0x7fb7f407ac <__GI___libc_write+4> adrp x3, 0x7fb7fd1000 <__libc_pthread_functions+184>
0x7fb7f407b0 <__GI___libc_write+8> mov x29, sp
0x7fb7f407b4 <__GI___libc_write+12> str x19, [sp, #16]
0x7fb7f407b8 <__GI___libc_write+16> sxtw x19, w0
0x7fb7f407bc <__GI___libc_write+20> ldr w0, [x3, #264]
0x7fb7f407c0 <__GI___libc_write+24> cbnz w0, 0x7fb7f407f0 <__GI___libc_write+72>
0x7fb7f407c4 <__GI___libc_write+28> mov x0, x19
0x7fb7f407c8 <__GI___libc_write+32> mov x8, #0x40 // #64
0x7fb7f407cc <__GI___libc_write+36>          #0x0
```

SVC

SVC指令 SuperVisor Call.

Syntax

`SVC{cond} #imm`

where:

`cond`

is an optional condition code.

`imm`

is an expression evaluating to an integer in the range:

- 0 to 2^{24}–1 (a 24-bit value) in an ARM instruction.
- 0-255 (an 8-bit value) in a Thumb instruction.

https://www.keil.com/support/man/docs/armasm/armasm_dom1361289909139.htm

Instruction Fault Status Code

- `0b000000` Address size fault in TTBR0 or TTBR1.
- `0b000101` Translation fault, 1st level.
- `00b00110` Translation fault, 2nd level.
- `00b00111` Translation fault, 3rd level.
- `0b001001` Access flag fault, 1st level.
- `0b001010` Access flag fault, 2nd level.
- `0b001011` Access flag fault, 3rd level.
- `0b001101` Permission fault, 1st level.
- `0b001110` Permission fault, 2nd level.
- `0b001111` Permission fault, 3rd level.
- `0b010000` Synchronous external abort.
- `0b011000` Synchronous parity error on memory access.
- `0b010101` Synchronous external abort on translation table walk, 1st level.
- `0b010110` Synchronous external abort on translation table walk, 2nd level.
- `0b010111` Synchronous external abort on translation table walk, 3rd level.
- `0b011101` Synchronous parity error on memory access on translation table walk, 1st level.
- `0b011110` Synchronous parity error on memory access on translation table walk, 2nd level.
- `0b011111` Synchronous parity error on memory access on translation table walk, 3rd level.
- `0b100001` Alignment fault.
- `0b100010` Debug event.

切问而近思

欢迎关注格友公众号