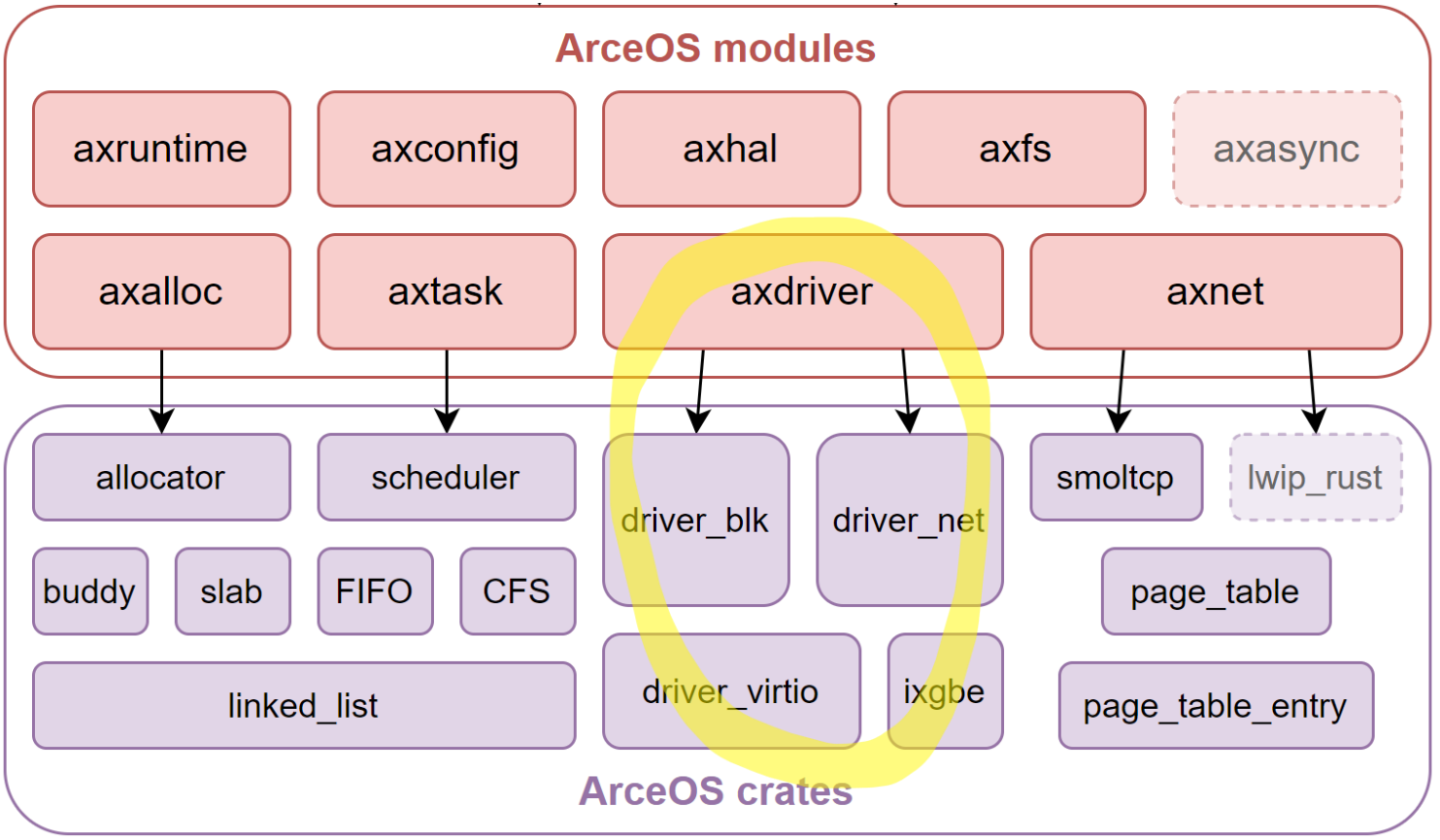
开源技术训练营

独立硬件驱动开发（第一周）

萧络元
清华大学计算机系
2025-06-16

ArceOS驱动框架

驱动课程仓库：
github.com/elliott10/dev-hw-driver



ArceOS驱动框架

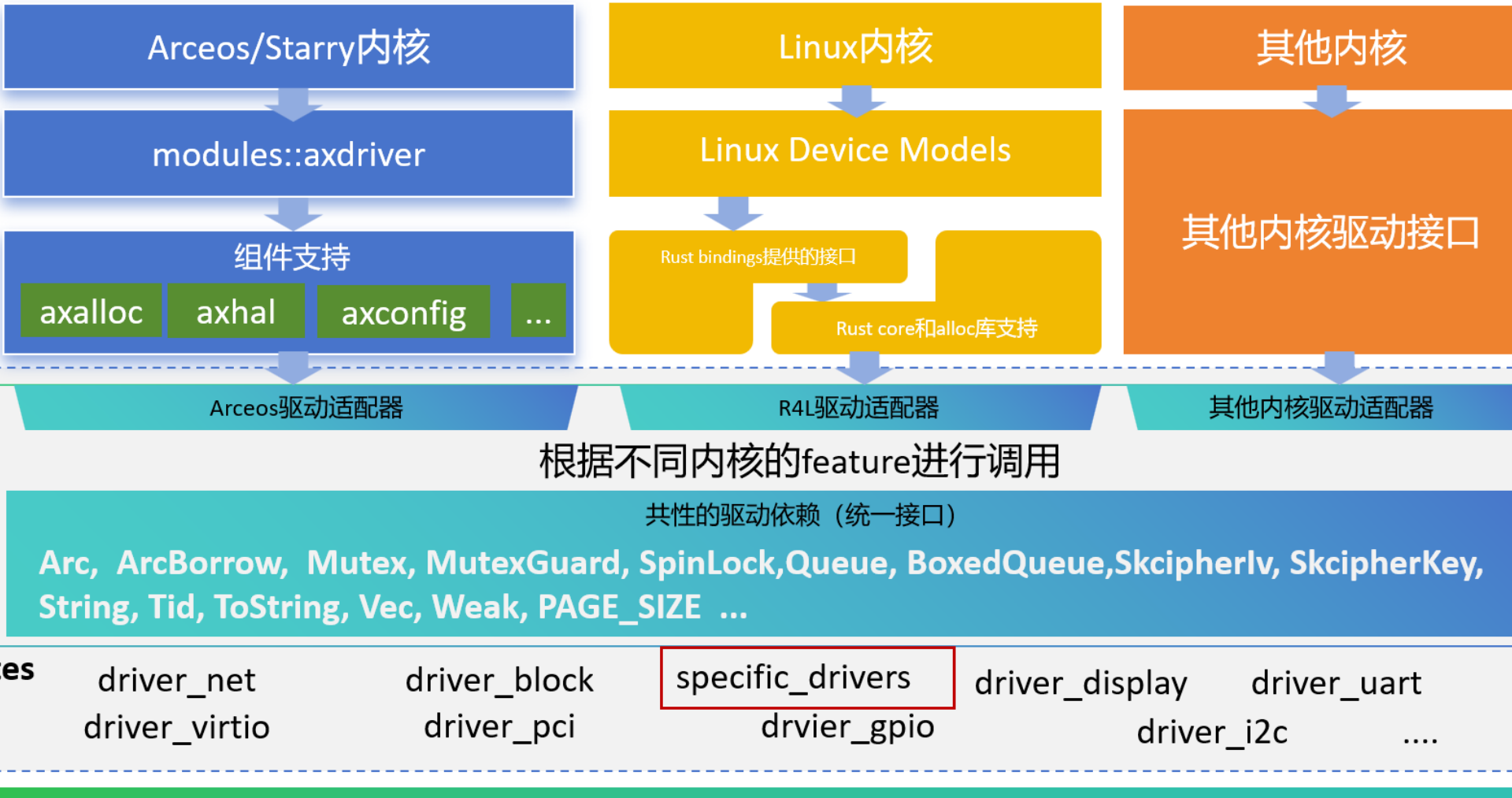
```rust
#[cfg(feature = "block")]
pub use self::structs::AxBlockDevice;
#[cfg(feature = "display")]
pub use self::structs::AxDisplayDevice;
#[cfg(feature = "net")]
pub use self::structs::AxNetDevice;

/// A structure that contains all device drivers, organized by their cat
#[derive(Default)]
pub struct AllDevices {
    /// All network device drivers.
    #[cfg(feature = "net")]
    pub net: AxDeviceContainer<AxNetDevice>,
    /// All block device drivers.
    #[cfg(feature = "block")]
    pub block: AxDeviceContainer<AxBlockDevice>,
    /// All graphics device drivers.
    #[cfg(feature = "display")]
    pub display: AxDeviceContainer<AxDisplayDevice>,
}
```

- 块设备驱动
- 显示驱动
- 网络驱动
  - 注册驱动

register_{net|block|display}_driver

- 网络驱动
  - PCI初始化扫描
  - 网卡设备初始化，实现网络数据收发接口
  - Starry/crates/driver_virtio/src/net.rs
  - Starry/crates/driver_net/src/ixgbe.rs

跨内核的硬件驱动开发


Arceos/Starry内核
modules::axdriver
组件支持
axalloc
axhal
axconfig
...
Linux内核
Linux Device Models
Rust bindings提供的接口
Rust core和alloc库支持
其他内核
其他内核驱动接口
Arceos驱动适配器
R4L驱动适配器
其他内核驱动适配器
根据不同内核的feature进行调用
共性的驱动依赖（统一接口）
Arc, ArcBorrow, Mutex, MutexGuard, SpinLock,Queue, BoxedQueue,SkcipherIv, SkcipherKey,
String, Tid, ToString, Vec, Weak, PAGE_SIZE ...
es
driver_net
driver_virtio
driver_block
driver_pci
specific_drivers
drvier_gpio
driver_display
driver_uart
driver_i2c
....
硬件设备

ArceOS内核代码结构的理解

» 分析Makefile的结构，理解包括编译和运行的执行流；

» 分析Rust的Cargo.toml的结构，理解各个features和依赖库crates的关系；

» 配置字段：

- [package]
- [features]
- [dependencies]： features, optional

  条件编译cfg(target_arch = "riscv64")

启动链

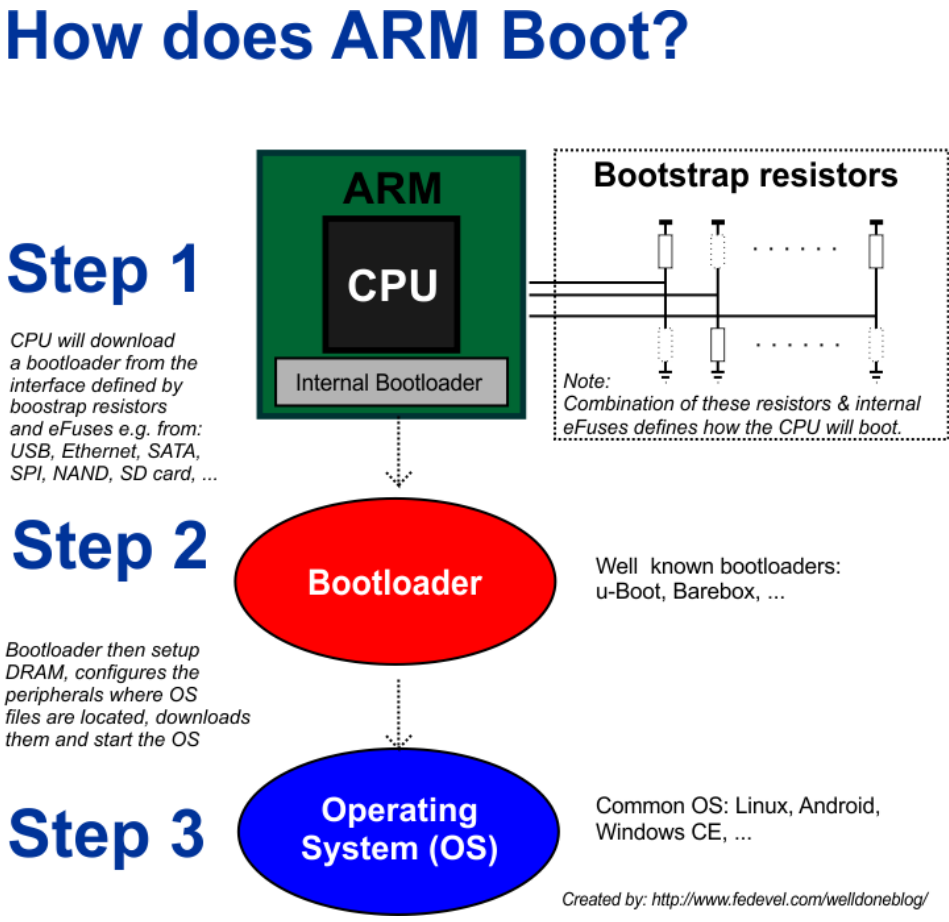
» Qemu, ARM64开发板

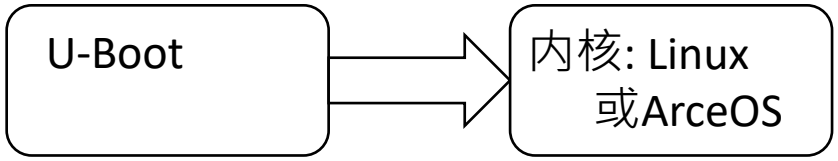
» Bootloader: u-boot

– 加载ArceOS镜像到指定内存；

– 或切换到EL1态；

– 跳转到内核入口ENTRY(_start) 跑起来





ArceOS运行入口

- 让ArceOS跑起来
- 定义入口函数 _start
- 启动入口汇编asm

```
/// The earliest entry point for the primary CPU.
#[naked]
#[no_mangle]
#[link_section = ".text.boot"]
unsafe extern "C" fn _start() -> ! {
    // PC = 0x8_0000
    // X0 = dtb
    core::arch::asm!("
        mrs     x19, mpidr_el1
        and     x19, x19, #0xffffff     // get current CPU id
        mov     x20, x0                 // save DTB pointer

        adrp    x8, {boot_stack}        // setup boot stack
        add     x8, x8, {boot_stack_size}
        mov     sp, x8

        bl      {switch_to_el1}         // switch to EL1
        bl      {init_boot_page_table}
        bl      {init_mmu}              // setup MMU
        bl      {enable_fp}             // enable fp/neon

        mov     x8, {phys_virt_offset}  // set SP to the high
        add     sp, sp, x8

        mov     x0, x19                 // call rust_entry(cp
        mov     x1, x20
        ldr     x8, ={entry}
        blr     x8
        b       .",
```

上帝说要有光

» 有光 便能看得见

» ArceOS的输出与输入

– 定义mod console

– 输出putchar

– 输入getchar

» 让我们的内核有字符显示

设备地址空间定义

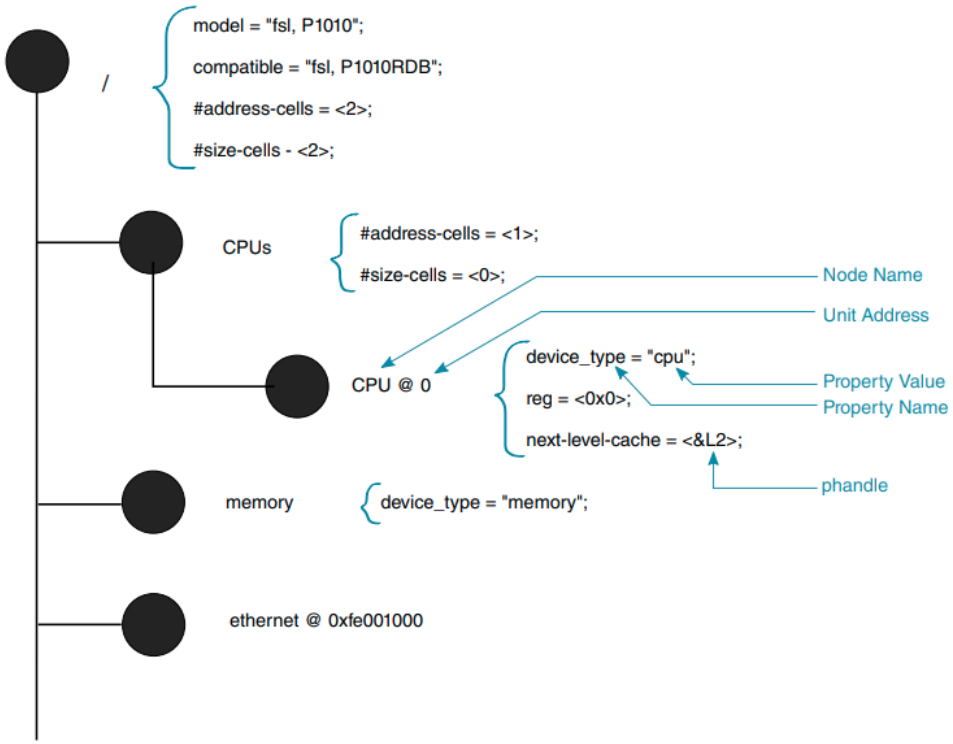
» 内核的设备地址定义于platforms/

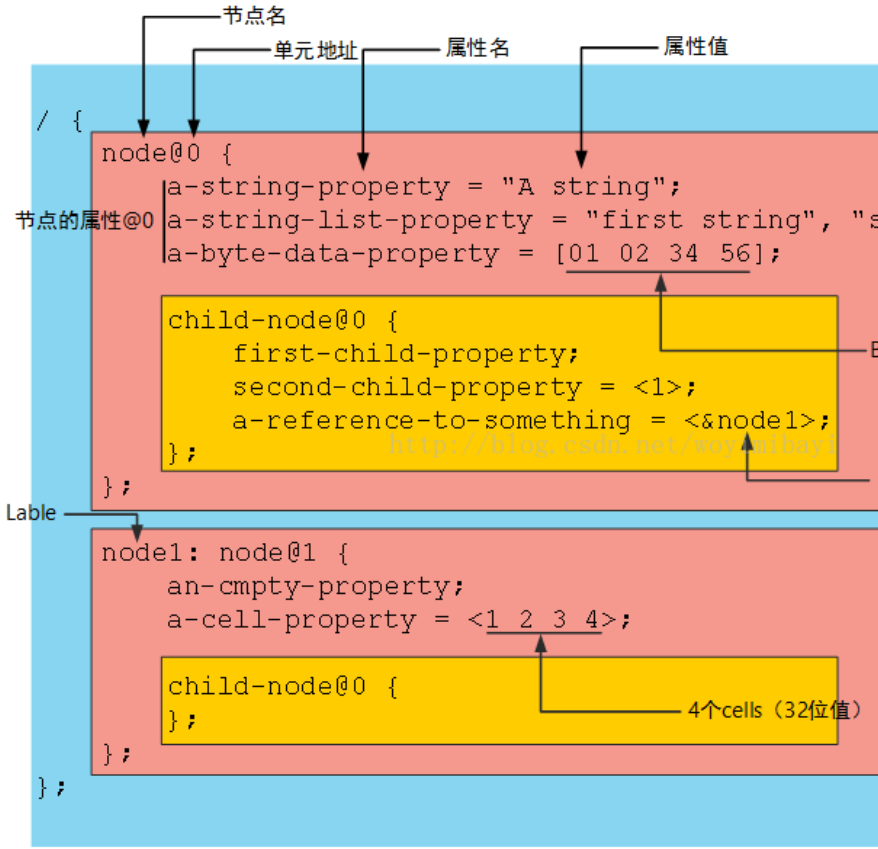
- 设备地址空间可以在设备树中查询；
- 设备地址空间的使用axhal/

```
# MMIO regions with format (`base_paddr`, `size`).
mmio-regions = [
    ["0x20008000", "0x1000"], # uart8250 UART0
    ["0x32000000", "0x8000"], # arm,gic-400
    ["0x32011000", "0x1000"], # CPU CSR
    ["0x33002000", "0x1000"], # Top CRM
    ["0x70035000", "0x1000"], # CRM reg
    ["0x70038000", "0x1000"], # aon pinmux
]

virtio-mmio-regions = []

# Base physical address of the PCIe ECAM space.
# pci-ecam-base = "0x40_1000_0000"
# End PCI bus number (`bus-range` property in device tree).
# pci-bus-end = "0xff"
# PCI device memory ranges (`ranges` property in device tree).
# pci-ranges = []

# UART Address
uart-paddr = "0x20008000"
# UART irq from device tree
uart-irq = "0xd5"
# GICD Address
gicd-paddr = "0x32001000"
# GICC Address
gicc-paddr = "0x32002000"

# BST A1000B board registers
CPU_CSR_BASE = "0x32011000"
A1000BASE_TOPCRM = "0x33002000"
A1000BASE_SAFETYCRM = "0x70035000"
A1000BASE_AONCFG = "0x70038000"

# PSCI
psci-method = "smc"
```

设备地址空间定义

» 设备树Device-tree





Trap-异常和中断

异常的上下文切换

异常向量表的基地址寄存器VBAR_ELn

- 异常
  - 指令，内存等
  - 同步异常产生的原因ESR_Eln
  - 出错时的虚拟地址FAR_ELn
- 中断FIQ
  - GICC, GICD，串口等

Address	Exception type	Description
VBAR_ELn + 0x000	Synchronous	
+ 0x080	IRQ/vIRQ	
+ 0x100	FIQ/vFIQ	Current EL with SP
+ 0x180	SError/vSError	
+ 0x200	Synchronous	
+ 0x280	IRQ/vIRQ	
+ 0x300	FIQ/vFIQ	Current EL with SP
+ 0x380	SError/vSError	
+ 0x400	Synchronous	
+ 0x480	IRQ/vIRQ	
+ 0x500	FIQ/vFIQ	Lower EL using AA
+ 0x580	SError/vSError	
+ 0x600	Synchronous	
+ 0x680	IRQ/vIRQ	
+ 0x700	FIQ/vFIQ	Lower EL using AA
+ 0x780	SError/vSError	

Trap-异常和中断

同步异常的类型寄存器

ESR_ELn

- 异常类型ESR_Eln::EC
  - 断点
  - 指令访问
  - 数据内存访问等

```
pub ESR_EL1 [
    /// Exception Class. Indicates the reason for the exception that this
    /// information about.
    ///
    /// For each EC value, the table references a subsection that gives in
    ///   - The cause of the exception, for example the configuration requ
    ///     trap.
    ///   - The encoding of the associated ISS.
    ///
    /// Incomplete listing - to be done.
    EC  OFFSET(26) NUMBITS(6) [
        Unknown                = 0b00_0000,
        TrappedWFIorWFE        = 0b00_0001,
        TrappedMCRorMRC        = 0b00_0011, // A32
        TrappedMCRRorMRRC      = 0b00_0100, // A32
        TrappedMCRorMRC2       = 0b00_0101, // A32
        TrappedLDCorSTC        = 0b00_0110, // A32
        TrappedFP              = 0b00_0111,
        TrappedMRRC            = 0b00_1100, // A32
        BranchTarget           = 0b00_1101,
        IllegalExecutionState  = 0b00_1110,
        SVC32                  = 0b01_0001, // A32
        SVC64                  = 0b01_0101,
        HVC64                  = 0b01_0110,
        SMC64                  = 0b01_0111,
        TrappedMsrMrs          = 0b01_1000,
        TrappedSve             = 0b01_1001,
        PointerAuth            = 0b01_1100,
        InstrAbortLowerEL      = 0b10_0000,
        InstrAbortCurrentEL    = 0b10_0001,
        PCAlignmentFault       = 0b10_0010,
        DataAbortLowerEL       = 0b10_0100,
        DataAbortCurrentEL     = 0b10_0101,
        SPAlignmentFault       = 0b10_0110,
        TrappedFP32            = 0b10_1000, // A32
        TrappedFP64            = 0b10_1100,
        SError                 = 0b10_1111,
        BreakpointLowerEL      = 0b11_0000,
        BreakpointCurrentEL    = 0b11_0001,
        SoftwareStepLowerEL    = 0b11_0010,
        SoftwareStepCurrentEL  = 0b11_0011,
        WatchpointLowerEL      = 0b11_0100,
        WatchpointCurrentEL    = 0b11_0101,
        Bkpt32                 = 0b11_1000, // A32 BKTP instruction
        Brk64                  = 0b11_1100  // A64 BRK instruction
    ],
```

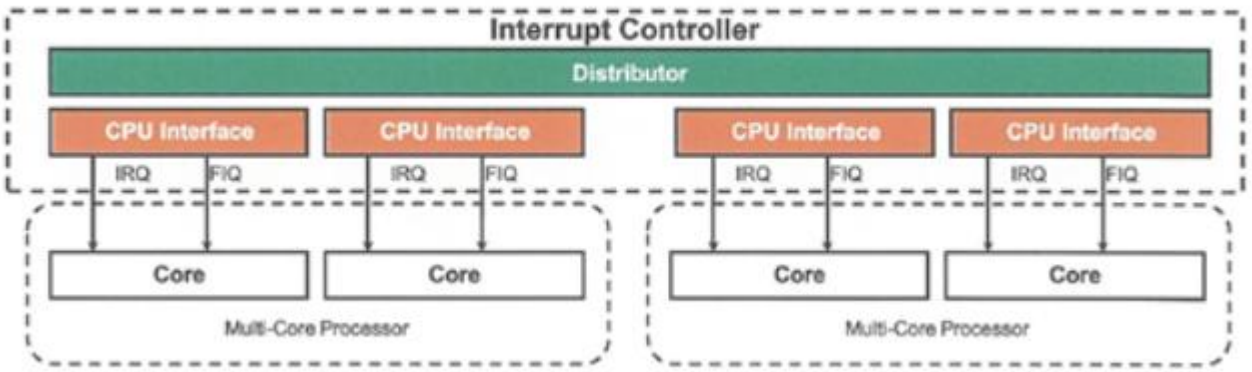
GIC中断

» ARM体系结构定义了通用中断控制器(GIC)

– 分发器(Distributor)系统中的所有中断源都连接到该单元。可以通过寄存器来控制各个中断源的属性，例如优先级、状态、安全性、路由信息和使能状态。

– CPU接口单元(CPU Interface) CPU核通过控制器的CPU接口单元接收中断。CPU接口单元寄存器用于屏蔽，识别和控制转发到CPU核的中断的状态。软件可以使用中断ID来识别中断源并调用相应的处理程序来处理中断。

The programmer's model for GICv2 is structured like this:



GIC中断

» GICD-分发器

» Distributor register descriptions

Table 4-1 Distributor register map

Offset	Name	Type	Reset[a]	Description
0x000	GICD_CTLR	RW	0x00000000	Distributor Control Register
0x004	GICD_TYPER	RO	IMPLEMENTATION DEFINED	Interrupt Controller Type Register
0x008	GICD_IIDR	RO	IMPLEMENTATION DEFINED	Distributor Implementer Identification Register
0x00C-0x01C	-	-	-	Reserved
0x020-0x03C	-	-	-	IMPLEMENTATION DEFINED registers
0x040-0x07C	-	-	-	Reserved
0x080	GICD_IGROUPRn[b]	RW	IMPLEMENTATION DEFINED[c]	Interrupt Group Registers
0x084-0x0FC			0x00000000	
0x100-0x17C	GICD_ISENABLERn	RW	IMPLEMENTATION DEFINED	Interrupt Set-Enable Registers
0x180-0x1FC	GICD_ICENABLERn	RW	IMPLEMENTATION DEFINED	Interrupt Clear-Enable Registers
0x200-0x27C	GICD_ISPENDRn	RW	0x00000000	Interrupt Set-Pending Registers
0x280-0x2FC	GICD_ICPENDRn	RW	0x00000000	Interrupt Clear-Pending Registers
0x300-0x37C	GICD_ISACTIVERn[d]	RW	0x00000000	GICv2 Interrupt Set-Active Registers
0x380-0x3FC	GICD_ICACTIVERn[e]	RW	0x00000000	Interrupt Clear-Active Registers
0x400-0x7F8	GICD_IPRIORITYRn	RW	0x00000000	Interrupt Priority Registers
0x7FC	-	-	-	Reserved
0x800-0x81C	GICD_ITARGETSRn	RO[f]	IMPLEMENTATION DEFINED	Interrupt Processor Targets Registers
0x820-0xBF8		RW[f]	0x00000000	
0xBFC	-	-	-	Reserved
0xC00-0xCFC	GICD_ICFGRn	RW	IMPLEMENTATION DEFINED	Interrupt Configuration Registers
0xD00-0xDFC	-	-	-	IMPLEMENTATION DEFINED registers
0xE00-0xEFC	GICD_NSACRn[e]	RW	0x00000000	Non-secure Access Control Registers, optional
0xF00	GICD_SGIR	WO	-	Software Generated Interrupt Register
0xF04-0xF0C	-	-	-	Reserved
0xF10-0xF1C	GICD_CPENDSGIRn[e]	RW	0x00000000	SGI Clear-Pending Registers
0xF20-0xF2C	GICD_SPENDSGIRn[e]	RW	0x00000000	SGI Set-Pending Registers
0xF30-0xFCC	-	-	-	Reserved
0xFD0-0xFFC	-	RO	IMPLEMENTATION DEFINED	*Identification registers* on page 4-119

GIC中断

GICC- CPU接口单元，CPU interface register descriptions

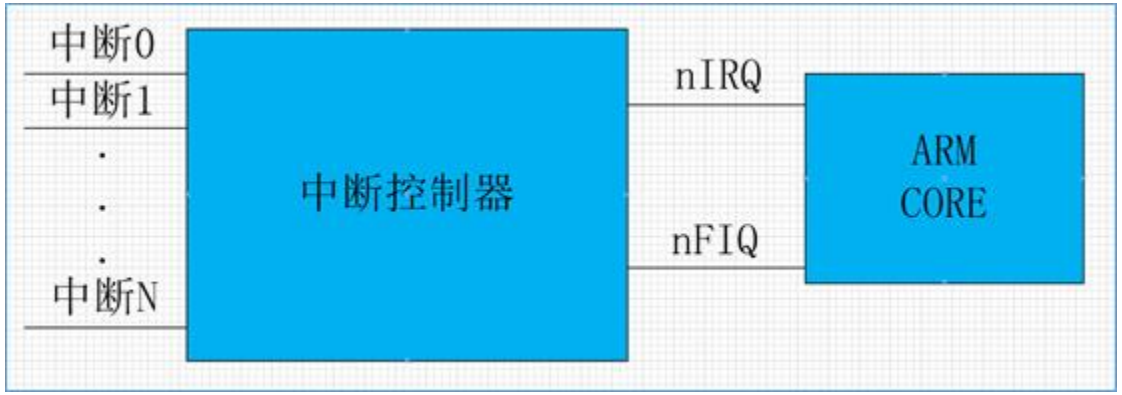
Table 4-2 CPU interface register map

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	0x0000000x[a]	Binary Point Register
0x000C	GICC_IAR	RO	0x000003FF	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[b]	RW	0x0000000x[a]	Aliased Binary Point Register
0x0020	GICC_AIAR[c]	RO	0x000003FF	Aliased Interrupt Acknowledge Register
0x0024	GICC_AEOIR[c]	WO	-	Aliased End of Interrupt Register
0x0028	GICC_AHPPIR[c]	RO	0x000003FF	Aliased Highest Priority Pending Interrupt Register
0x002C-0x003C	-	-	-	Reserved
0x0040-0x00CF	-	-	-	IMPLEMENTATION DEFINED registers
0x00D0-0x00DC	GICC_APRn[c]	RW	0x00000000	Active Priorities Registers
0x00E0-0x00EC	GICC_NSAPRn[c]	RW	0x00000000	Non-secure Active Priorities Registers
0x00ED-0x00F8	-	-	-	Reserved
0x00FC	GICC_IIDR	RO	IMPLEMENTATION DEFINED	CPU Interface Identification Register
0x1000	GICC_DIR[c]	WO	-	Deactivate Interrupt Register

GIC中断

» 中断类型

- SGI软件中断(核间中断) = [0, 15]，由软件通过写入专用寄存器即软件触发中断寄存器（ICDSGIR）显式生成的。它最常用于CPU核间通信。
- PPI私有中断(如Timer) = [16, 31]，由单个CPU核私有的外设生成的。
- SPI外设中断(如Uart) = [32, X]，由各种外围设备发出的，中断控制器可以将其路由到多个核。



硬件设备驱动-Pinctrl引脚复用

Pinctrl: Pin Controller, 顾名思义就是用来控制引脚的：

» 引脚枚举与命名(Enumerating and naming): 如：单个引脚，各组引脚

» 引脚复用(Multiplexing)：比如用作GPIO、I2C或其他功能

– 通用 GPIO，即用来做输入、输出和中断

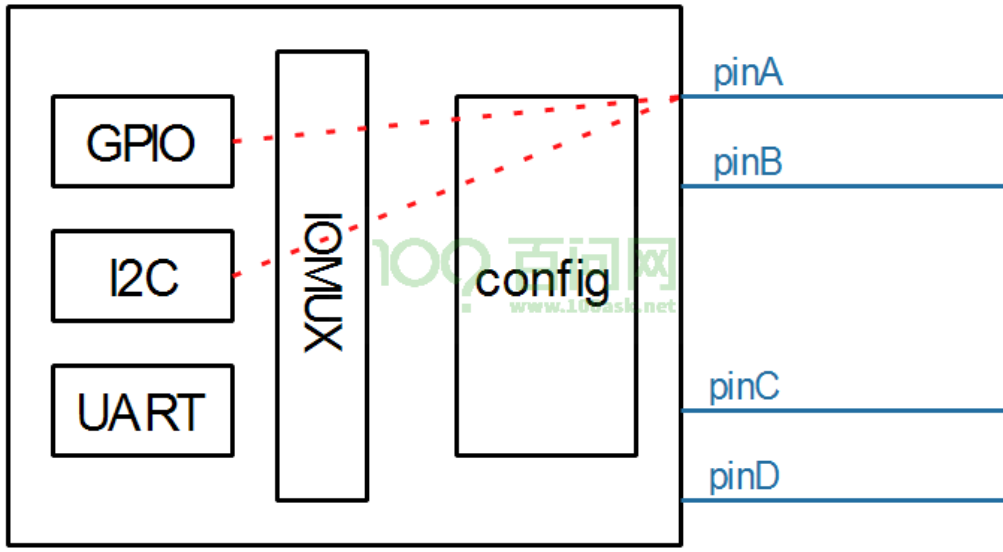
» 引脚配置(Configuration)：比如上拉、下来、open drain、驱动强度等

在一般的设备驱动程序里，甚至可以没有pinctrl的代码Pinctrl驱动由芯片厂家的BSP提供，一般的只需要在设备树里：

– 指明使用那些引脚

– 复用为哪些功能

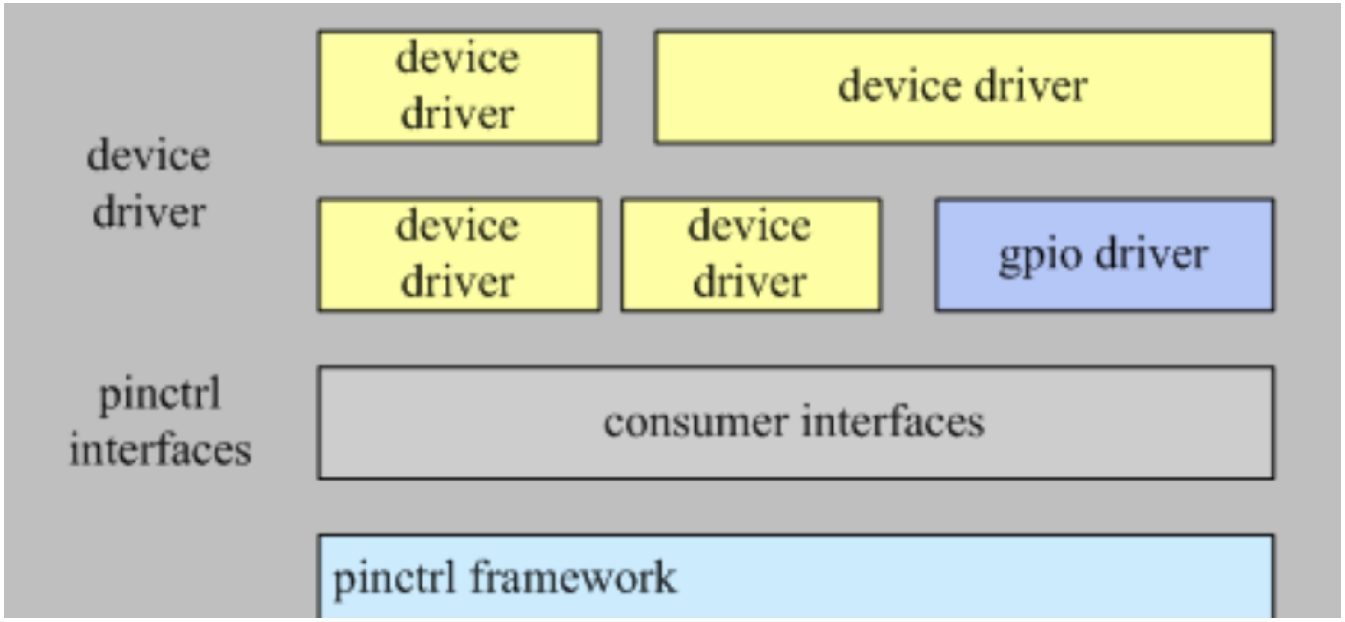
– 配置为哪些状态



硬件设备驱动-Pinctrl引脚复用

Pinctrl 驱动整体框架:

- Pinctrl api: pinctrl 提供给上层用户调用的接口。
- Pinctrl framework：Linux 提供的 pinctrl 驱动框架。
- Pinctrl board driver：板级平台需要实现的驱动。
- Board configuration：设备 pin 配置信息，一般采用设备树进行配置。



硬件设备驱动-GPIO

芯片内部有很多引脚，这些引脚可以接到GPIO模块，也可以接到I2C等模块。可以通过Pinctrl子系统来选择引脚的功能(mux function)、配置引脚。

» 当一个引脚被复用为GPIO功能时，我们可以去设置它的方向、设置/读取它的值。

» GPIO名为"General Purpose Input/Output"，通用目的输入/输出，就是常用的引脚。它的通用功能包括：

  – 触发中断，如按键

  – 输出：让它输出高低电平，如点亮LED；

  – 输入，读取引脚当前电平；

» 属性包括：Active-High and Active-Low

  – LED为例，需要设置GPIO电平。但是有些电路可能是高电平点亮LED，有些是低电平点亮LED。

硬件设备驱动-GPIO

» GPIO驱动实验-Qemu平台的GPIO关机功能

» ARM64 virt机器提供了GPIO来实现关机功能
。我们将编写GPIO相关的驱动来实现关机功能定义

– 分析设备树的GPIO节点
– 编写pl061 GPIO 通用输入输出模块的驱动
– 实现关机功能

硬件设备驱动-GPIO

» Qemu平台的GPIO关机功能

– 设备树的GPIO节点

– gpio-keys 中定义了一个 poweroff 键

- gpios = <0x8003 0x03 0x00> 中的第一项 0x8003 表示它的 phandle 是 0x8003，即 pl061，代表 gpio-keys 是设备 pl061 的组成部分
- 第二项 0x03 表示该键是 pl061 的第三根 GPIO 线
- 第三项是flag

– GPIO控制器pl061的起始地址：0x9030000

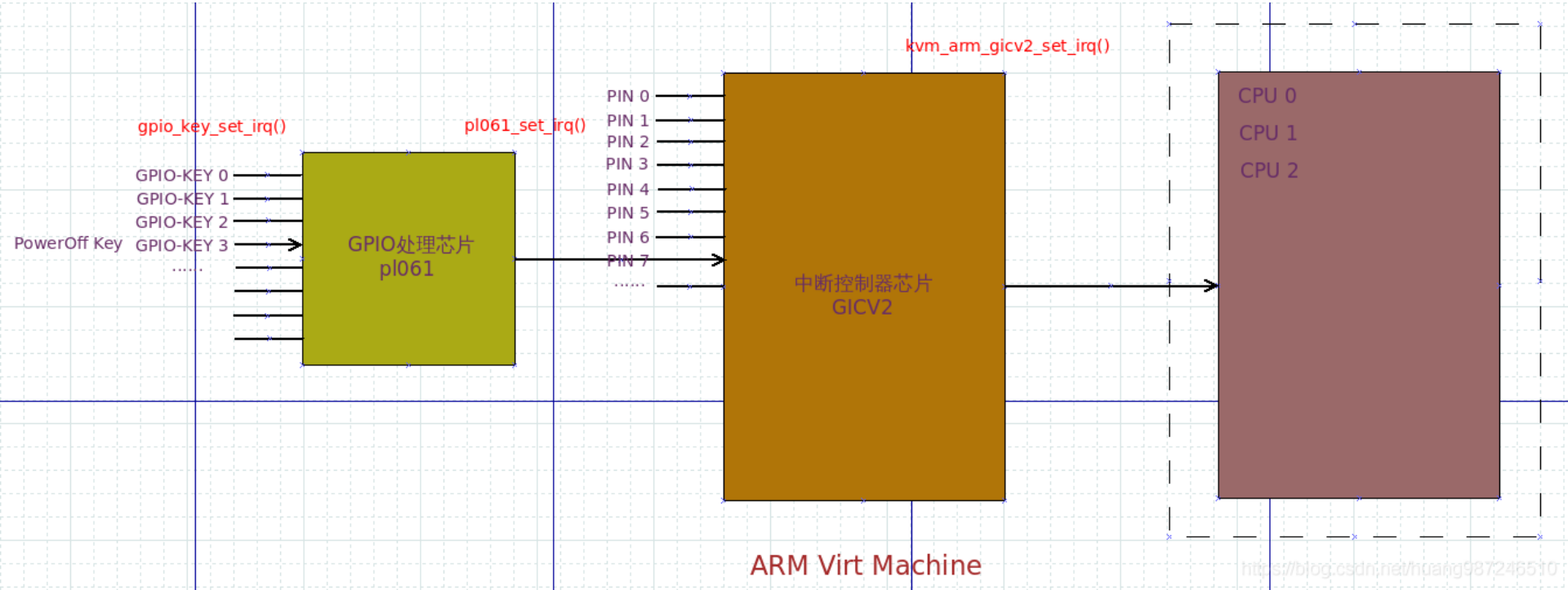
```
gpio-keys {
        compatible = "gpio-keys";

        poweroff {
                gpios = <0x8004 0x03 0x00>
                linux,code = <0x74>;
                label = "GPIO Key Poweroff"
        };
};

pl061@9030000 {
        phandle = <0x8004>;
        clock-names = "apb_pclk";
        clocks = <0x8000>;
        interrupts = <0x00 0x07 0x04>;
        gpio-controller;
        #gpio-cells = <0x02>;
        compatible = "arm,pl061\0arm,primec
        reg = <0x00 0x9030000 0x00 0x1000>
};
```

硬件设备驱动-GPIO

» Qemu平台的GPIO关机功能

– 如图可以看到，GPIO-KEY3关机键接入到了GPIO 控制器 : pl061芯片的3号输入口。

– 当外部按下关机按钮或输入关机指令时，3号线将产生一次信号并发生一次中断。我们需要在OS中处理该关机中断，实现关机功能。



硬件设备驱动-GPIO

GPIO寄存器包括:

* GPIORIS -
中断状态寄存器

* GPIOIE -
中断掩码寄存器

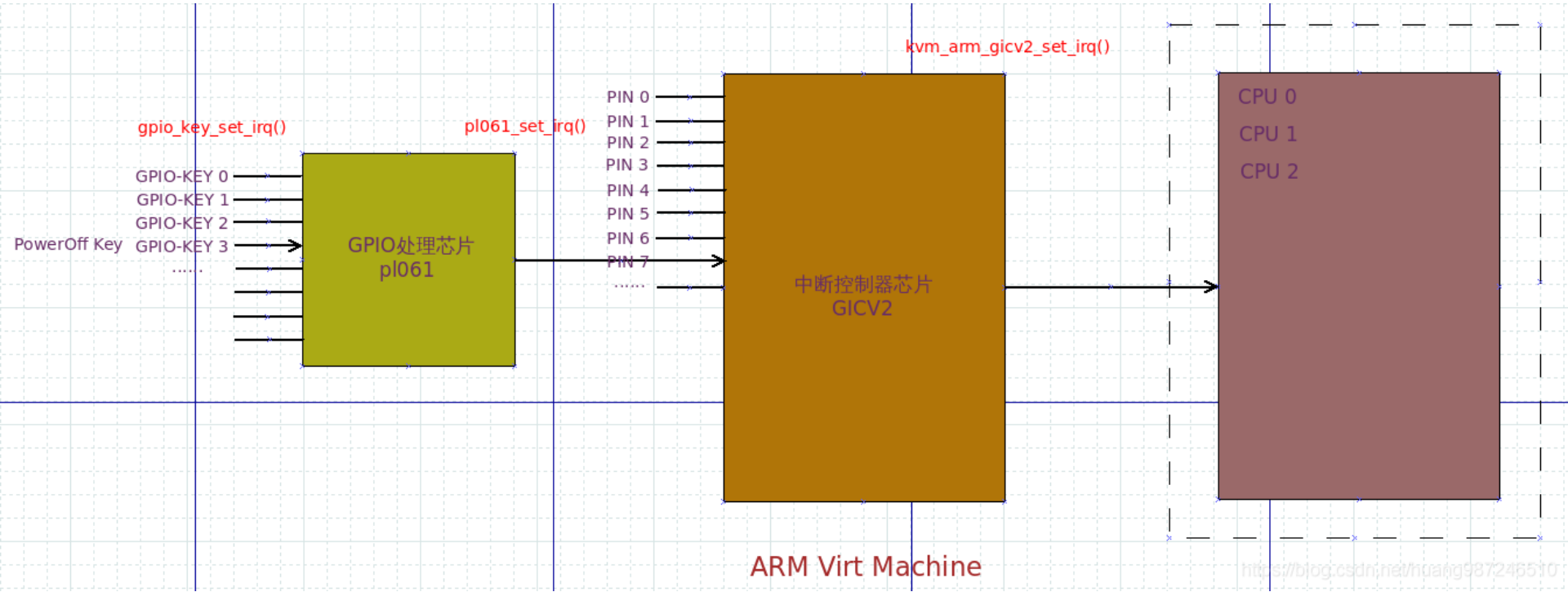
* GPIOIC -
中断清除寄存器

The PrimeCell GPIO registers are shown in Table 3-1.

Table 3-1 PrimeCell GPIO register summary

Address	Type	Width	Reset value	Name	Description
GPIO base + 0x000 to GPIO base + 0x3FC	Read/write	8	0x00	GPIODATA	PrimeCell GPIO data register
GPIO base + 0x400	Read/write	8	0x00	GPIODIR	PrimeCell GPIO data direction register
GPIO base + 0x404	Read/write	8	0x00	GPIOIS	PrimeCell GPIO interrupt sense register
GPIO base + 0x408	Read/write	8	0x00	GPIOIBE	PrimeCell GPIO interrupt both edges register
GPIO base + 0x40C	Read/write	8	0x00	GPIOIEV	PrimeCell GPIO interrupt event register
GPIO base + 0x410	Read/write	8	0x00	GPIOIE	PrimeCell GPIO interrupt mask
GPIO base + 0x414	Read	8	0x00	GPIORIS	PrimeCell GPIO raw interrupt status
GPIO base + 0x418	Read	8	0x00	GPIOMIS	PrimeCell GPIO masked interrupt status
GPIO base + 0x41C	Write	8	0x00	GPIOIC	PrimeCell GPIO interrupt clear
GPIO base + 0x420	Read/write	8	0x00	GPIOAFSEL	PrimeCell GPIO mode control select

硬件设备驱动- ArceOS GPIO驱动开发



硬件设备驱动-GPIO

GPIO驱动实验- Qemu ARM64平台的GPIO关机功能驱动编写实例

- 初始化关机的GPIO中断
  - 使能GPIO中断
  - 使能GPIO的poweroff key中断，启用pl061 gpio中的3号线中断，设置GPIOIE::IO3::Enabled
- 对关机GPIO中断进行处理
  - Qemu命令终端输入 system_powerdown 来触发关机按钮
  - 中断处理函数中，处理39号中断，通过读取pl061来清除它的中断信号
  - 执行关机指令：mov w0, #0x18; hlt #0xF000

硬件设备驱动-GPIO

当我们按下GPIO关机按钮,或Qemu命令终端输入system_powerdown时

- GPIORIS（中断状态寄存器PrimeCell GPIO raw interrupt status）中的第三位将从0跳变到1。
- 而当GPIOIE（中断掩码寄存器PrimeCell GPIO interrupt mask）中的第三位为1时，GPIO处理芯片将向GIC中断控制器发送一次中断，中断号为39。
- OS收到中断后，需要对此次GPIO中断进行清除，将GPIOIC（中断清除寄存器PrimeCell GPIO interrupt clear）的对应位置为1
- 然后OS进行关机操作。

谢 谢!