Linux内核编程11期：device tree

主讲：王利涛

01 为什么要引入device tree？

主讲：王利涛

- 本节知识点
  - 什么是device tree？
  - platform驱动编写方法
    - » platform_device->resource: IRQ、reg addr
    - » platform_driver
    - » match->probe—初始化设备
  - platform/i2c/spi 驱动给内核带来的问题
    - » arch/arm/mach-xxx   arch/arm/plat-xxx
  - device tree解决了内核什么问题
    - » .dts----.dtb
    - » 硬件解耦：内核和某个具体的开发板平台
  - 引入DT后，给内核带来的变化
    - » 驱动开发
    - » 内核启动

- 内核中的平台源码占比

版本	2.6.0	2.6.39	5.10.4
arch	14.28%	17.42%	8.18
arch/arm	0.81%	4.99%	1.76%
支持的CPU架构	20	24	24
支持的arm平台	16	64	74

- 本期课程学习重点
  - 设备树语法
  - 如何修改和配置设备树
  - 常用外设的设备树配置及语法
  - 引入DT后，如何编写驱动
  - 掌握驱动开发相关的DT接口
  - 理解设备树的整体框架、运行过程
  - 自定义属性和解析
  - Device binding

- 学习前提
  - 设备模型、总线型驱动
  - platform、i2c、spi
  - SoC芯片架构、AMBA总线
  - 学习平台：vexpress
  - 内核版本：Linux-5.10.x

02 如何编译和运行device tree？

主讲：王利涛

- 本节主要知识点
    - 如何编译device tree
    - 编译过程及Makefile分析
    - device tree的加载和运行
    - 如何反编译dtb文件？

- 如何编译device tree？

```
#  make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- dtbs
手动编译：# ./scripts/dtc/dtc   –I  dts  –O  dtb   -o  xxx.dtb    xxx.dts
反编译：   # ./scripts/dtc/dtc   –I  dtb  –O  dts   -o   xxx.dts     xxx.dtb
反编译：  # fdtdump   xxx.dtb  > xxx.dts
```

- 如何加载和运行device tree？

```
# bootm  <uImage_addr> <initrd_addr> <dtb_addr>
# bootm  0x60003000 – 0x60500000
```

03 使用DT接口编写platform驱动

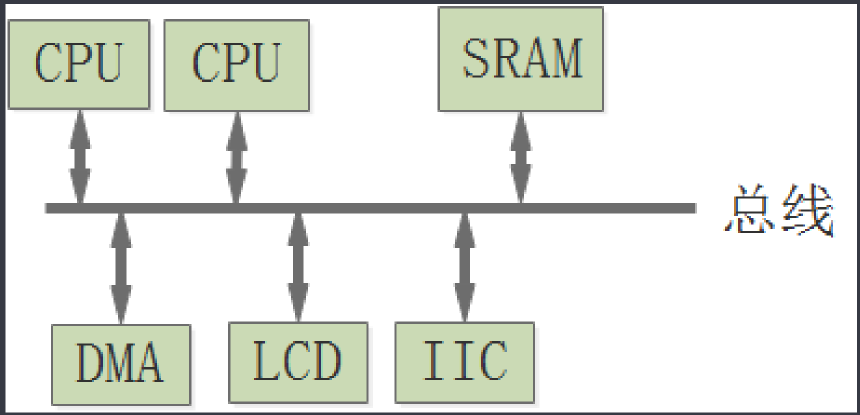
- 本节主要知识点
    - 内核引入DT后，platform 驱动的编写方法
    - platform驱动和设备如何匹配
    - 如何获取寄存器地址
    - 如何获取中断号
    - platform_device和resource的生成过程

04 SoC芯片架构：总线与片选

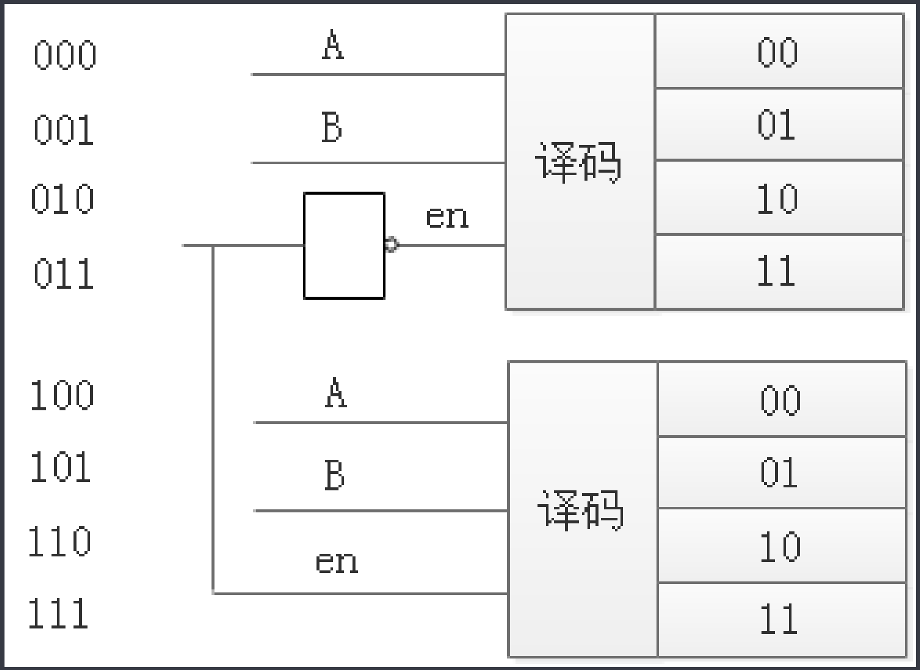
主讲：王利涛

- 本节主要知识点
    - ARM SoC芯片的基本架构
        - CPU、IP、primecell、controller
    - AMBA总线
        - 时钟、地址、数据、信号
    - 总线地址
    - 片选

- 总线基本概念、总线地址、片选
  - SoC构成：CPU、SRAM、controller、IP、Primecell
  - 总线的作用：连接各个设备，进行数据通信
  - 构成：总线带宽、时钟频率、仲裁机制、传输类型
  - 主设备、从设备
  - 总线类型
    - ARM：AMBA总线
    - IBM：CoreConnect 总线
    - Silicore：wishbone总线
    - Altera：Avalon总线

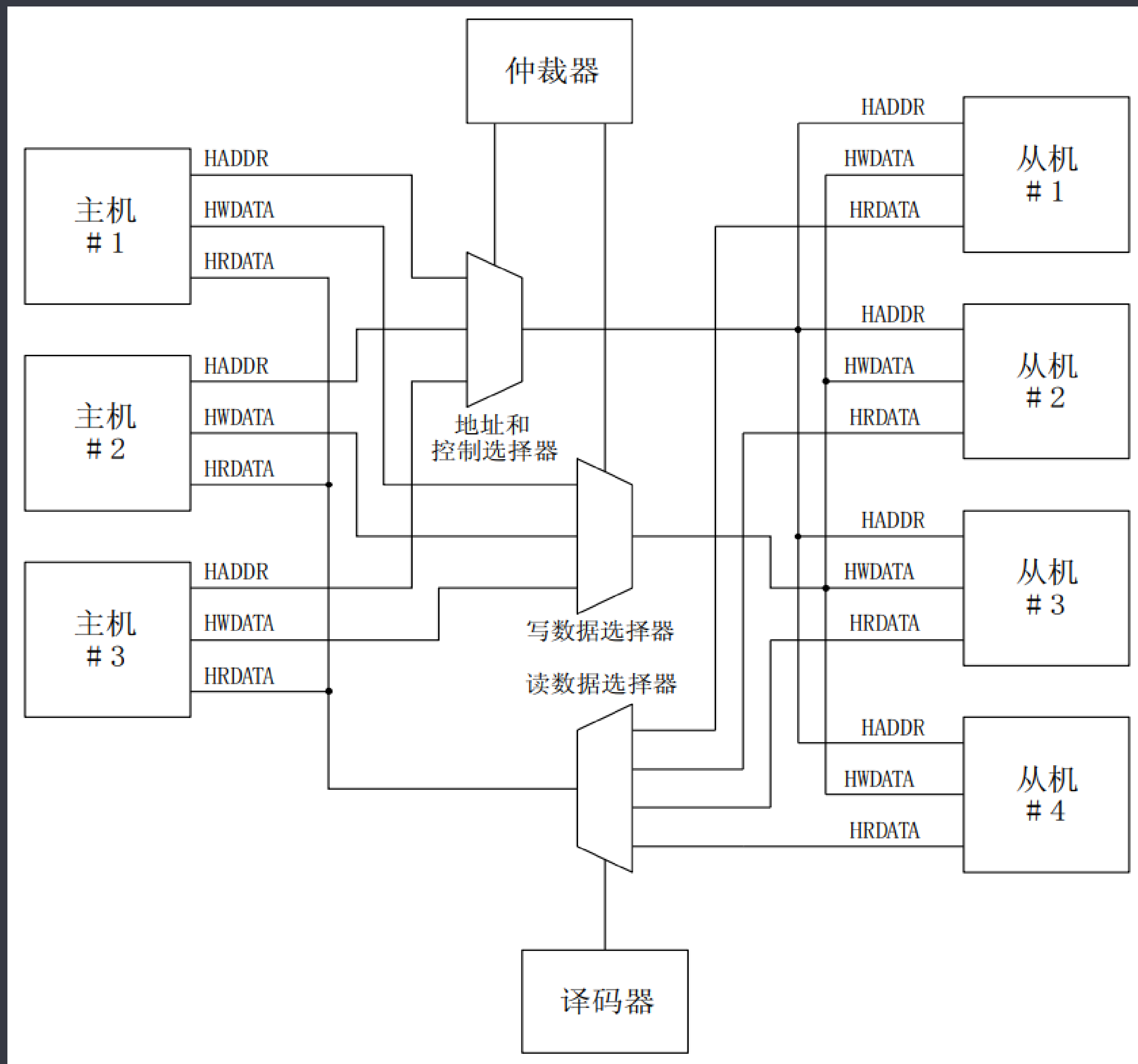


- 总线片选与编址

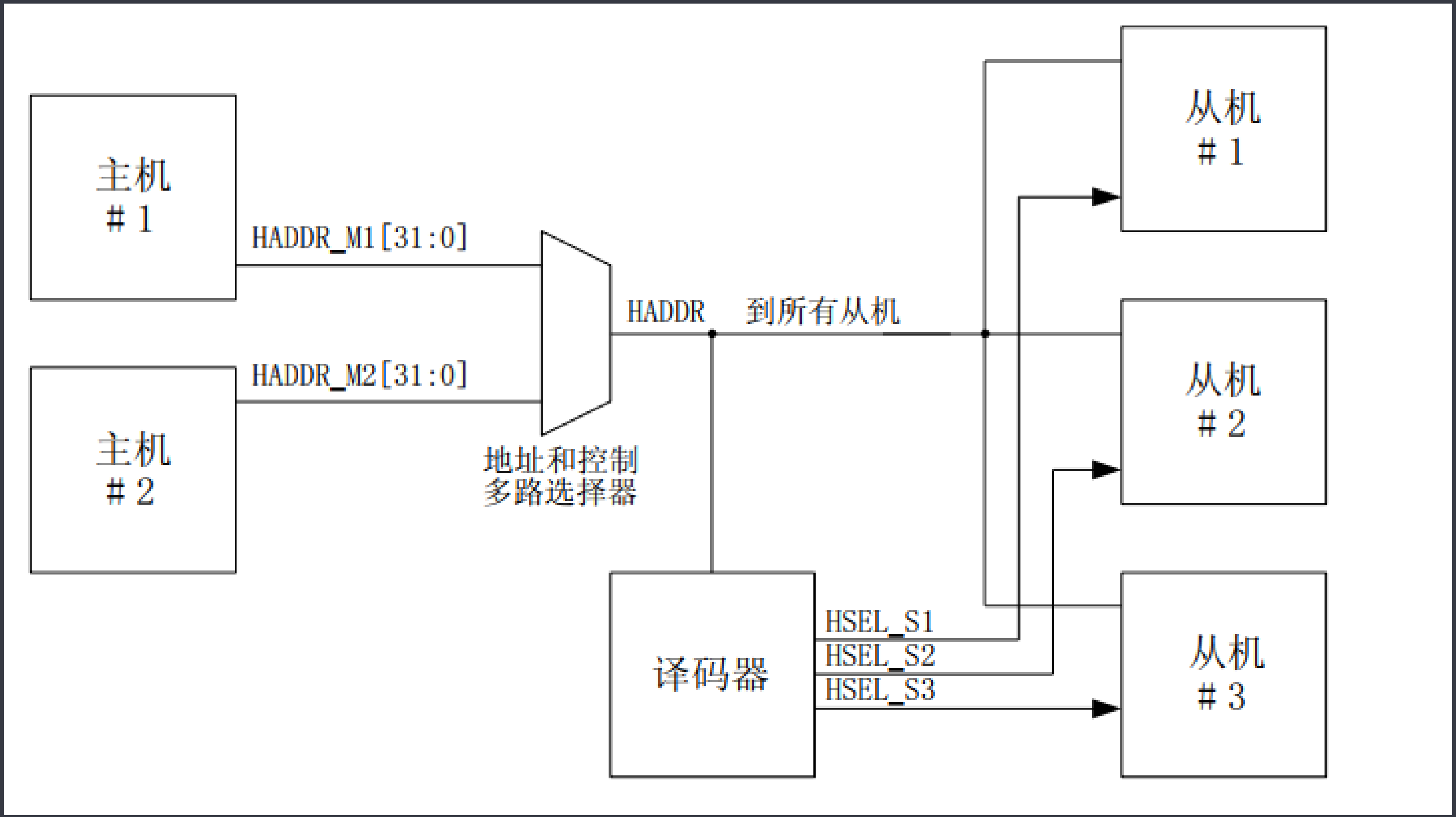


000
001
010
011
100
101
110
111
A
B
en
译码
00
01
10
11
A
B
en
译码
00
01
10
11

- AMBA总线
  - 构成：时钟、地址、数据、信号



- AMBA总线：从机片选

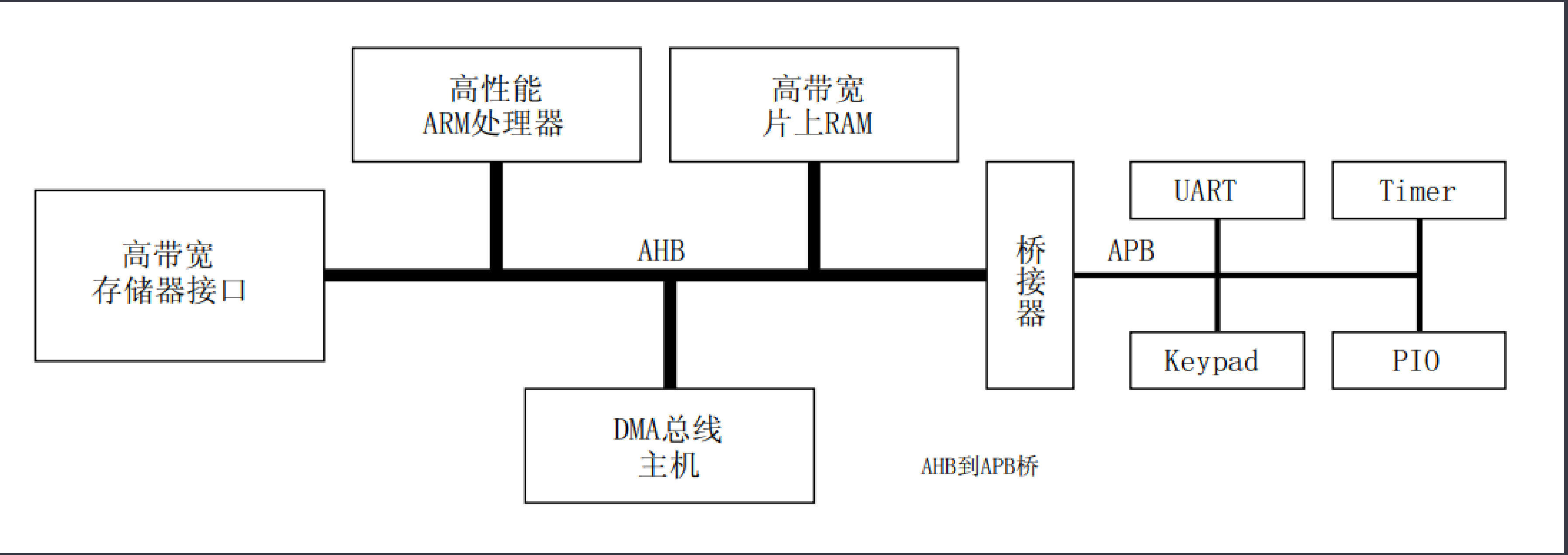


05 SoC芯片架构：桥接(bridge)

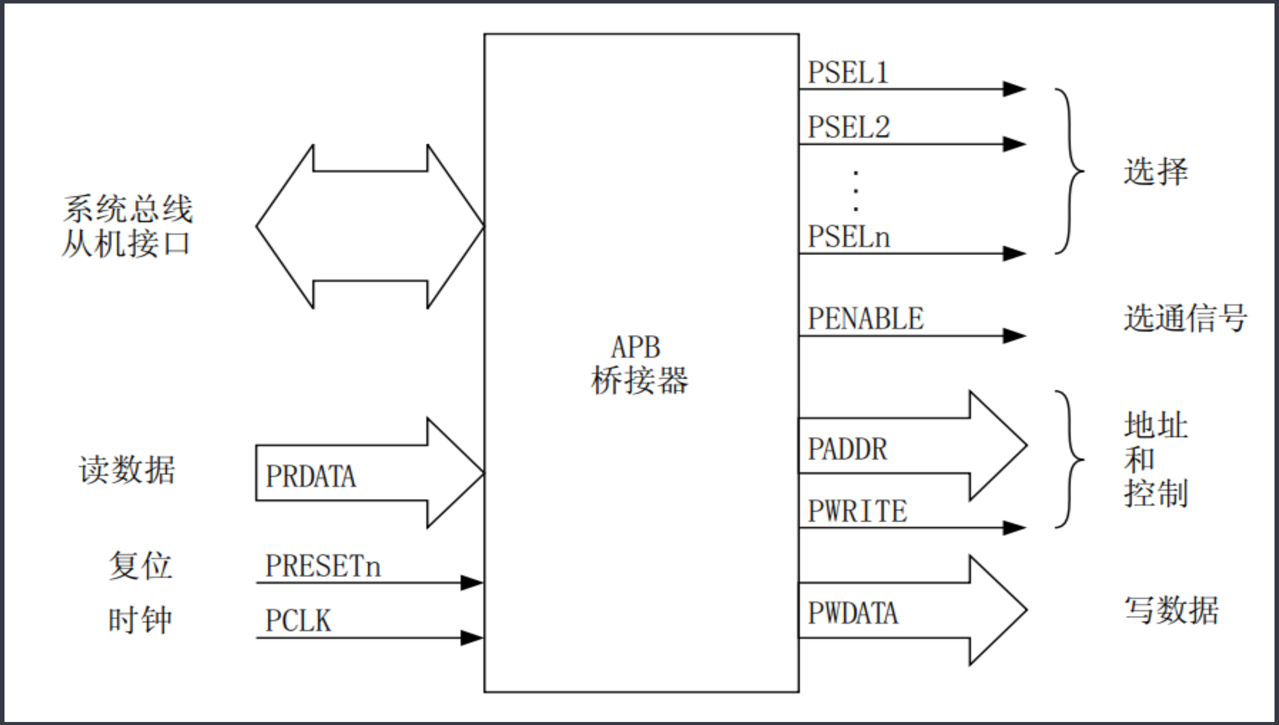
主讲：王利涛

- 本节主要知识点
    - 总线性能与功耗
    - 什么是桥接bridge？
    - 桥接的作用
    - 地址映射及译码
    - 不同地址域(domain)之间的转换

- 桥接：bridge
  - 锁存AHB总线信号：地址、数据、控制信号
  - 二级译码：APB外围设备的片选信号
  - AHB到APB协议的转换



- 桥接：bridge

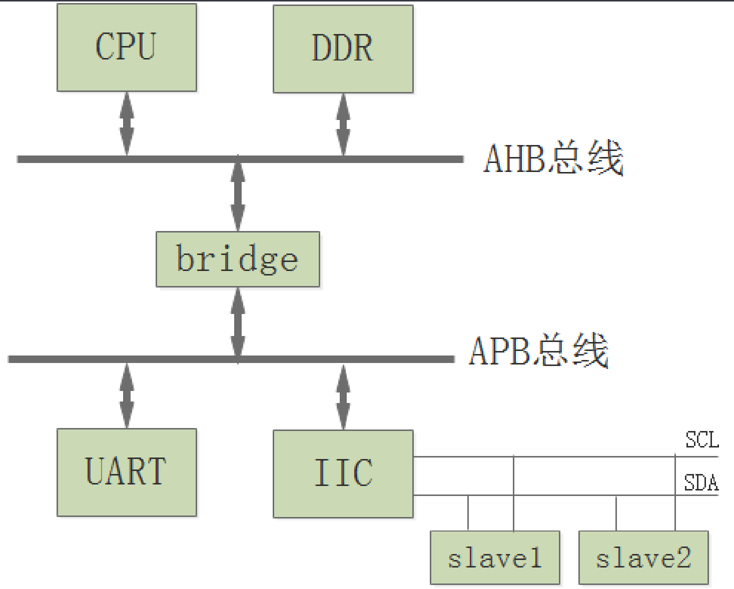


系统总线
从机接口
APB
桥接器
PSEL1
PSEL2
PSELn
选择
PENABLE
选通信号
PADDR
PWRITE
地址
和
控制
读数据
PRDATA
复位
PRESETn
时钟
PCLK
PWDATA
写数据

06 SoC芯片架构：extend bus

主讲：王利涛

- 扩展总线：extend bus
  - IIC总线及IIC控制器
  - IIC Slave地址
  - PCIe、USB总线



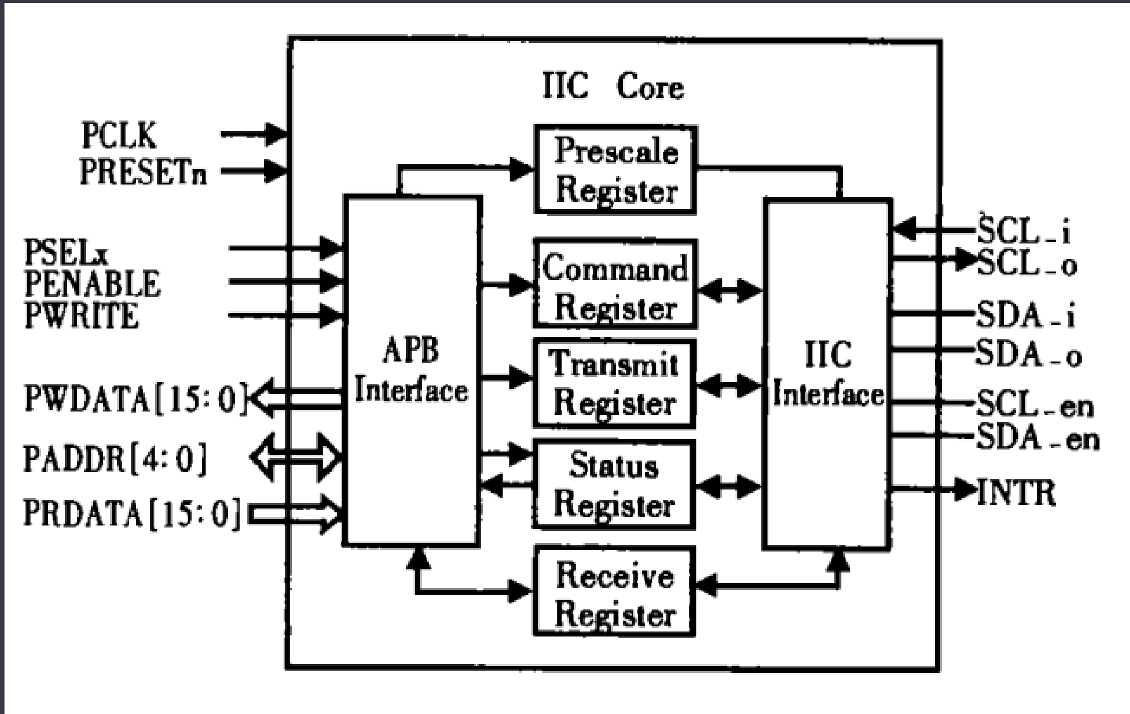


图 3 IIC Core 层次结构图

- 扩展总线设备的编址
  - 非内存映射设备：IIC设备
  - 内存映射设备：memory mapped device
  - 地址域、地址映射：ranges
  - PCI总线和设备的编址
  - SMBUS的地址编址

07 device tree基本语法：node

- 本节主要知识点
    - 节点node的写法
    - 根节点
    - 父节点、子节点
    - 节点的引用：phandle
    - 节点的引用：label
    - aliases节点

- 节点： node

```
// node-name@unit-name
/ {
        compatible = "arm,vexpress,v2p-ca9", "arm,vexpress";
        cpus {
                cpu@0 {
                        compatible = "arm,cortex-a9";
                };
                cpu@1 {
                        compatible = "arm,cortex-a9";
                };
        };

        memory@60000000 {
                device_type = "memory";
                reg = <0x60000000 0x40000000>;
        };
};
```

- 节点的引用

```
gic: interrupt-controller@1e001000 {
        compatible = "arm,cortex-a9-gic";
        phandle = <1>;
        interrupt-controller;
};

rtc@1e005000  {
        interrupt-parent = <1>;   // 使用phandle值为1来引用上述节点
};

labeltest:uart@1e008000  {
        interrupt-parent = <&gic>;  //使用label来引用上述节点
};
```

- aliases node

```
/ {
	compatible = "arm,vexpress,v2p-ca9", "arm,vexpress";
	aliases {
		core0 = /cpus/cpu@0;
		core1 = &cpu1;
	};
	cpus {
		cpu@0 {
			compatible = "arm,cortex-a9";
		};
		cpu1: cpu@1 {
			compatible = "arm,cortex-a9";
	};
};
```

节点命名规范

- atm
- cache-controller
- compact-flash
- can
- cpu
- crypto
- disk
- display
- dma-controller
- ethernet
- ethernet-phy
- fdc
- flash
- gpio
- i2c
- ide
- interrupt-controller
- isa
- keyboard
- mdio
- memory
- memory-controller
- mouse
- nvram
- parallel
- pc-card
- pci
- pcie
- rtc
- sata
- scsi
- serial
- sound
- spi
- timer
- usb
- vme
- watchdog

08 device tree基本语法：property

- 本节主要知识点
  - 属性的定义
  - 根节点的属性
    - model属性
    - compatible属性
  - 属性的类型
    - 标准属性
    - 自定义属性

- 属性类型：property
    - <empty>：为空时，不同的property代表不同的含义
    - <u32>：A 32-bit integer in big-endian format
    - <u64>：a 64-bit integer in big-endian format
    - <string>：
    - <prop-encoded-array>：自定义property格式
    - <phandle>：A <u32> value
    - <stringlist>：A list of <string> values

- 标准属性
  - compatible：“manufacturer,model”
  - model： specifies a model number
  - device_type：
  - phandle： specifies a numerical identifier for a node
  - status： the operational status of a device
  - #address-cells： defines the number of <u32> cells
  - #size-cells： encode size field in child node’s reg property
  - reg： <prop-encoded-array>， (address,length) pairs
  - ranges： <empty> or <prop-encoded-array>，
    (child- bus-address, parent-bus-address, length)
  - dma-ranges： <empty> or <prop-encoded-array>
    (child-bus-address, parent-bus-address, length)
  - name： the name of the node

- compatible属性
    - compatible属性的作用
    - 设备节点的compatible属性

```
static const struct of_device_id of_match_rtc[] = {
	{ .compatible = "arm,pl031", },
	{}
};

static struct platform_driver rtc_drv = {
	.probe  = rtc_driver_probe,
	.remove = rtc_driver_remove,
	.driver = {
	.name = "arm,pl031",
	.of_match_table = of_match_rtc,
	},
};
```

```
rtc@17000 {
		compatible = "arm,pl031", "arm,primecell";
		reg = <0x17000 0x1000>;
		interrupts = <4>;
		clocks = <&smbclk>;
		clock-names = "apb_pclk";

};
```

- compatible属性
  - compatible属性的作用
  - 根节点的compatible属性

```
/ {
	model = "V2P-CA9";
	arm,hbi = <0x191>;
	arm,vexpress,site = <0xf>;
	compatible = "arm,vexpress,v2p-ca9", "arm,vexpress";
	interrupt-parent = <&gic>;
	#address-cells = <1>;
	#size-cells = <1>;
	……
};
```

```
compatible = "arm,vexpress,v2p-ca9", "arm,vexpress";
arch/arm/mach-vexpress/v2m.c:
static const char * const v2m_dt_match[] __initconst = {
			"arm,vexpress",
			NULL,
};

DT_MACHINE_START(VEXPRESS_DT, "ARM-Versatile Express")
	.dt_compat		= v2m_dt_match,
	.l2c_aux_val		= 0x00400000,
	.l2c_aux_mask		= 0xfe0fffff,
	.smp = smp_ops(vexpress_smp_dt_ops),
	.smp_init = smp_init_ops(vexpress_smp_init_ops),
MACHINE_END
```

09 设备树实例分析：CPU node

- 本节主要知识点
    - 和CPU node相关的属性
    - 单核处理器
    - 多核CPU
    - 大小核处理器
    - 超线程处理器

- 单核CPU

```
cpus {
		#address-cells = <1>;
		#size-cells = <0>;
		cpu@0 {
				device_type = "cpu";
				reg = <0>;
				d-cache-block-size = <32>; // L1 - 32 bytes
				i-cache-block-size = <32>; // L1 - 32 bytes
				d-cache-size = <0x8000>; // L1, 32K
				i-cache-size = <0x8000>; // L1, 32K
				timebase-frequency = <82500000>; // 82.5 MHz
				clock-frequency = <825000000>; // 825 MHz
		};
};
```

和CPU相关的property

- CPU node
    - device_type
    - reg
    - clock-frequency
    - timebase-frequency
    - cache-op-block-size
    - reservation-granule-size
    - status
    - enable-method
- Cache property
    - cache-unified
    - cache-size
    - cache-sets
    - cache-block-size
    - cache-line-size
    - i-cache-size
    - i-cache-block-size
    - d-cache-size
    - d-cache-block-size
    - next-level-cache

• 单核CPU & 多级cache

```
cpu@0 {
        device_type = "cpu";
        reg = <0>;
        cache-unified;
        cache-size = <0x8000>;    // L1, 32 KB
        cache-block-size = <32>;
        timebase-frequency = <82500000>; // 82.5 MHz
        next-level-cache = <&L2_0>; // phandle to L2
        L2_0: l2-cache {
                compatible = "cache";
                cache-unified;
                cache-size = <0x40000>; // 256 KB
                cache-sets = <1024>;
                cache-block-size = <32>;
                cache-level = <2>;
                next-level-cache = <&L3>; // phandle to L3
                L3: l3-cache {
                        compatible = "cache";
                        cache-unified;
                        cache-size = <0x40000>; // 256 KB
                        cache-sets = <0x400>; // 1024
                        cache-block-size = <32>;
                        cache-level = <3>;
                };
        };
};
```

- 多核CPU

```
cpus {
		#address-cells = <1>;
		#size-cells = <0>;
		A9_0: cpu@0 {
				device_type = "cpu";
				compatible = "arm,cortex-a9";
				reg = <0>;
				next-level-cache = <&L2>;
		};
		A9_1: cpu@1 {
				device_type = "cpu";
				compatible = "arm,cortex-a9";
				reg = <1>;
				next-level-cache = <&L2>;
		};
		A9_2: cpu@2 {
				device_type = "cpu";
				compatible = "arm,cortex-a9";
				reg = <2>;
				next-level-cache = <&L2>;
		};
		A9_3: cpu@3 {
				device_type = "cpu";
				compatible = "arm,cortex-a9";
				reg = <3>;
				next-level-cache = <&L2>;
		};
};
```

• 多核CPU & 共享 L3 cache

```
cpu@0 {
        device_type = "cpu";
        reg = <0>;
        cache-unified;
        cache-size = <0x8000>;    // L1, 32 KB
        cache-block-size = <32>;
        timebase-frequency = <82500000>; // 82.5 MHz
        next-level-cache = <&L2_0>; // phandle to L2
        L2_0: l2-cache {
            compatible = "cache";
            cache-unified;
            cache-size = <0x40000>; // 256 KB
            cache-sets = <1024>;
            cache-block-size = <32>;
            cache-level = <2>;
            next-level-cache = <&L3>; // phandle to L3
            L3: l3-cache {
                    compatible = "cache";
                    cache-unified;
                    cache-size = <0x40000>; // 256 KB
                    cache-sets = <0x400>; // 1024
                    cache-block-size = <32>;
                    cache-level = <3>;
            };
        };
};
```

```
cpu@1 {
        device_type = "cpu";
        reg = <1>;
        cache-unified;
        cache-block-size = <32>;
        cache-size = <0x8000>; // L1, 32 KB
        timebase-frequency = <82500000>;
        clock-frequency = <825000000>; // 825 MHz
        cache-level = <2>;
        next-level-cache = <&L2_1>; //phandle to L2
        L2_1:l2-cache {
            compatible = "cache";
            cache-unified;
            cache-size = <0x40000>; // 256 KB
            cache-sets = <0x400>; // 1024
            cache-line-size = <32>; // 32 bytes
            next-level-cache = <&L3>; // phandle to L3
        };
};
```

- SMP处理器
  - cpus node
  - cpu-map node
    - socket
    - cluster
    - core
    - thread

```
cpus  {
   cpu-map {
      socket0 {
            cluster0 {
                   core0 { cpu = <&CPU1>; };

                  core1 { cpu = <&CPU2>; };

                  core2 { cpu0 = <&CPU2>; };

                  core3 { cpu0 = <&CPU3>; };
            };
       };
   };
};
```

- big-LITTLE 架构

```
//mt8135.dtsi

cpu-map {
    cluster0 {
        core0 {
            cpu = <&cpu0>;
        };
        core1 {
            cpu = <&cpu1>;
        };
    };
    cluster1 {
        core0 {
            cpu = <&cpu2>;
        };
        core1 {
            cpu = <&cpu3>;
        };
    };
};
```

```
cpus {
        #address-cells = <1>;
        #size-cells = <0>;
        enable-method = "mediatek,mt81xx-tz-smp";

        cpu0: cpu@0 {
                device_type = "cpu";
                compatible = "arm,cortex-a7";
                reg = <0x000>;
        };
        cpu1: cpu@1 {
                device_type = "cpu";
                compatible = "arm,cortex-a7";
                reg = <0x001>;
        };
        cpu2: cpu@100 {
                device_type = "cpu";
                compatible = "arm,cortex-a15";
                reg = <0x100>;
        };
        cpu3: cpu@101 {
                device_type = "cpu";
                compatible = "arm,cortex-a15";
                reg = <0x101>;
        };
};
```

- 超线程

```
cpu-map {
    socket0 {
        cluster0 {
            cluster0 {
                core0 {
                    thread0 {
                        cpu = <&CPU0>;
                    };
                    thread1 {
                        cpu = <&CPU1>;
                    };
                };

                core1 {
                    thread0 {
                        cpu = <&CPU2>;
                    };
                    thread1 {
                        cpu = <&CPU3>;
                    };
                };
            };

            cluster1 {
                core0 {
                    thread0 {
                        cpu = <&CPU4>;
                    };
                    thread1 {
                        cpu = <&CPU5>;
                    };
                };

                core1 {
                    thread0 {
                        cpu = <&CPU6>;
                    };
                    thread1 {
                        cpu = <&CPU7>;
                    };
                };
            };
        };
```

```
        cluster1 {
            cluster0 {
                core0 {
                    thread0 {
                        cpu = <&CPU8>;
                    };
                    thread1 {
                        cpu = <&CPU9>;
                    };
                };
                core1 {
                    thread0 {
                        cpu = <&CPU10>;
                    };
                    thread1 {
                        cpu = <&CPU11>;
                    };
                };
            };

            cluster1 {
                core0 {
                    thread0 {
                        cpu = <&CPU12>;
                    };
                    thread1 {
                        cpu = <&CPU13>;
                    };
                };
                core1 {
                    thread0 {
                        cpu = <&CPU14>;
                    };
                    thread1 {
                        cpu = <&CPU15>;
                    };
                };
            };
        };
    };
```

10 设备树实例分析：memory

- 本节主要知识点
  - memory节点
  - reserved-memory节点
  - reg属性
  - device_type属性
  - compatible属性
  - #address-cells属性
  - #size-cells属性

- memory node

```
/ {
	compatible = "arm,vexpress,v2p-ca9", "arm,vexpress";
	#address-cells = <1>;
	#size-cells = <1>;

	memory@60000000 {
		device_type = "memory";
		reg = <0x60000000 0x40000000>;
	};
	reserved-memory {
		#address-cells = <1>;
		#size-cells = <1>;
		ranges;
		vram: vram@4c000000 {
		/* 8 MB of designated video RAM */
		compatible = "shared-dma-pool";
		reg = <0x4c000000 0x00800000>;
		no-map;
		};
	};
};
```

- 多个内存设备
  - 使用多个memory node描述
  - 在memory node中使用多个reg属性

```
memory@60000000 {
		device_type = "memory";
		reg = <0x60000000 0x40000000>;
};
memory@80000000 {
		device_type = "memory";
		reg = <0x80000000 0x40000000>;
};
```

```
memory {
		device_type = "memory";
		reg = <0x60000000 0x40000000>
			<0x80000000 0x40000000>;
};
```

11 设备树实例分析：外设

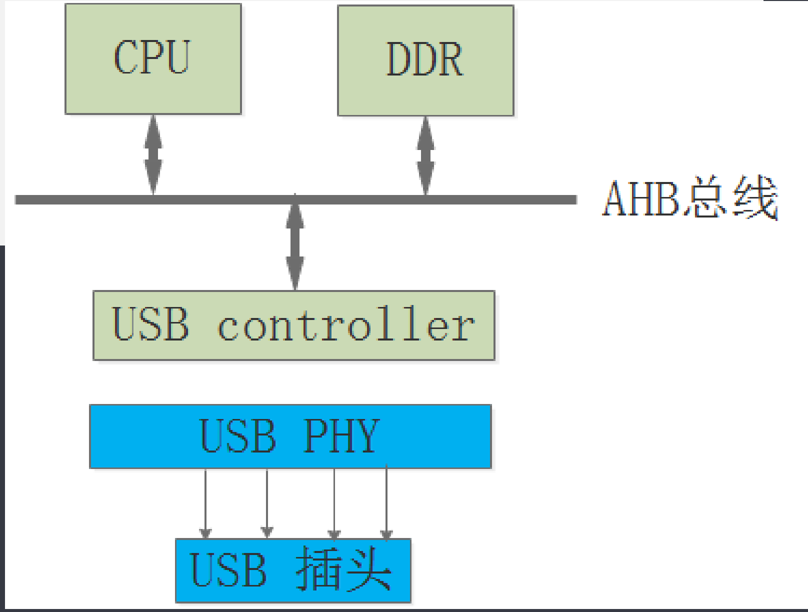
- 本节主要知识点
  - ARM primecell 、IP controller
  - reg、#address-cells、#size-cells
  - clocks属性
  - interrupts属性

- Primecell node

```
memory-controller@100e1000 {
        compatible = "arm,pl354", "arm,primecell";
        reg = <0x100e1000 0x1000>;
        interrupts = <0 45 4>,
                     <0 46 4>;
        clocks = <&oscclk2>;
        clock-names = "apb_pclk";
};

timer@100e4000 {
        compatible = "arm,sp804", "arm,primecell";
        reg = <0x100e4000 0x1000>;
        interrupts = <0 48 4>,
                     <0 49 4>;
        clocks = <&oscclk2>, <&oscclk2>, <&oscclk2>;
        clock-names = "timer0clk", "timer1clk", "apb_pclk";
        status = "disabled";
};
```

• IP controller

```
usb_otg: usb@10180000 {
        compatible = "rockchip,rk3066-usb", "snps,dwc2";
        reg = <0x10180000 0x40000>;
        interrupts = <GIC_SPI  16  IRQ_TYPE_LEVEL_HIGH>;
        clocks = <&cru HCLK_OTG0>;
        clock-names = "otg";
        dr_mode = "otg";
        g-np-tx-fifo-size = <16>;
        g-rx-fifo-size = <275>;
        g-tx-fifo-size = <256 128 128 64 64 32>;
        phys = <&usbphy0>;
        phy-names = "usb2-phy";
        status = "disabled";
};
```

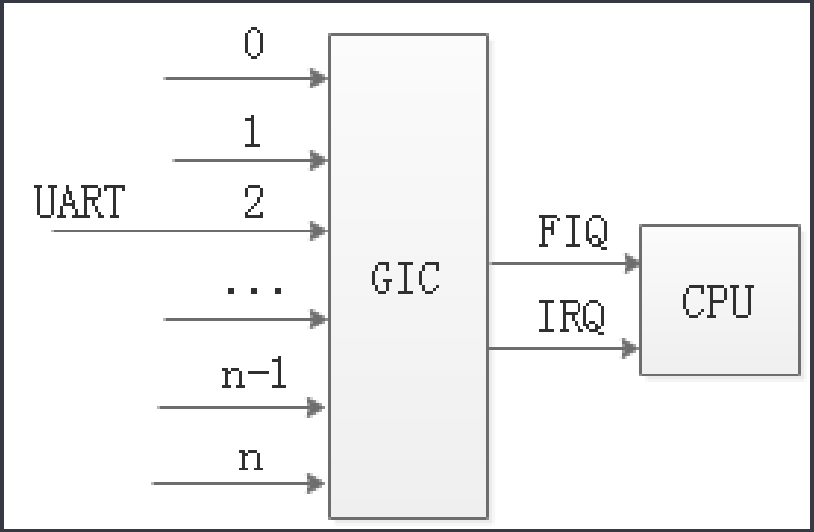


- phy

```
usbphy: phy {
        compatible = "rockchip,rk3288-usb-phy";
        #address-cells = <1>;
        #size-cells = <0>;
        usbphy0: usb-phy0 {
                #phy-cells = <0>;
                reg = <0x320>;
        };
};

 usb0_phy: phy@4100000 {
        compatible = "ti,am654-usb2", "ti,omap-usb2";
        reg = <0x4100000 0x54>;
        syscon-phy-power = <&scm_conf 0x4000>;
        clocks = <&k3_clks 151 0>, <&k3_clks 151 1>;
        clock-names = "wkupclk", "refclk";
        #phy-cells = <0>;
   };
```

12 设备树实例分析：中断控制器

- 本节主要知识点
  - interrupt controller node
  - interrupt client node
  - interrupts 属性
  - interrupt-parent属性
  - #interrupt-cells属性

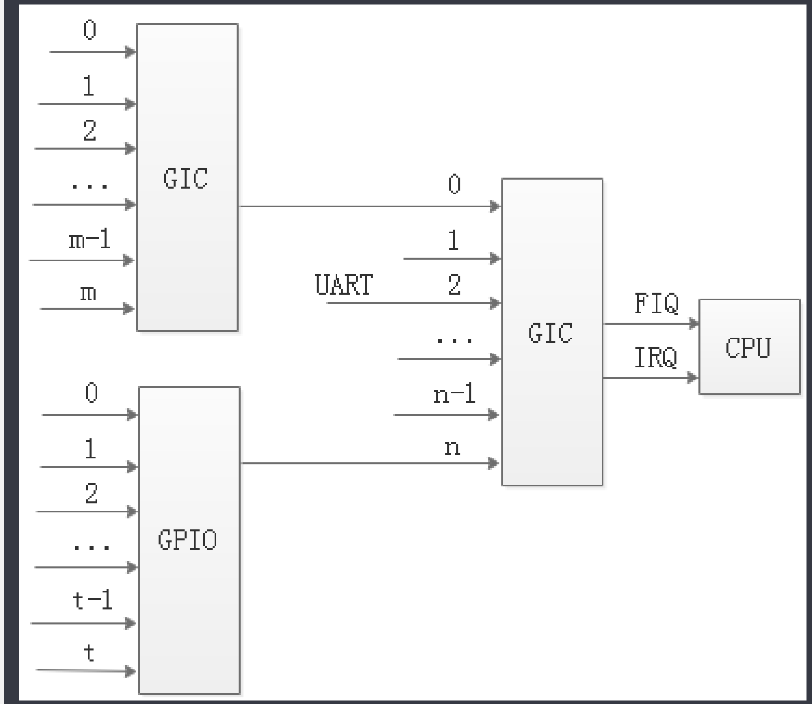


- 中断控制器及引用：3 cells

```
gic: interrupt-controller@1e001000 {
		compatible = "arm,cortex-a9-gic";
		#interrupt-cells = <3>;
		#address-cells = <0>;
		interrupt-controller;
		reg =	<0x1e001000 0x1000>,
			<0x1e000100 0x100>;
};

memory-controller@100e1000 {
		compatible = "arm,pl354", "arm,primecell";
		reg = <0x100e1000 0x1000>;
		interrupt-parent = <&gic>;
		interrupts = <0 45 4>,
			 <0 46 4>;
};
timer@1e000600 {
		compatible = "arm,cortex-a9-twd-timer";
		reg = <0x1e000600 0x20>;
		interrupts = <1 13 4>;
};
```

• 中断级联：Interrupt Nexus

```
vic: intc@10140000 {
        compatible = "arm,versatile-vic";
        interrupt-controller;
        #interrupt-cells = <1>;
        reg = <0x10140000 0x1000>;
};

sic: intc@10003000 {
        compatible = "arm,versatile-sic";
        interrupt-controller;
        #interrupt-cells = <1>;
        reg = <0x10003000 0x1000>;
        interrupt-parent = <&vic>;
        interrupts = <31>; /* Cascaded to vic */
};
```



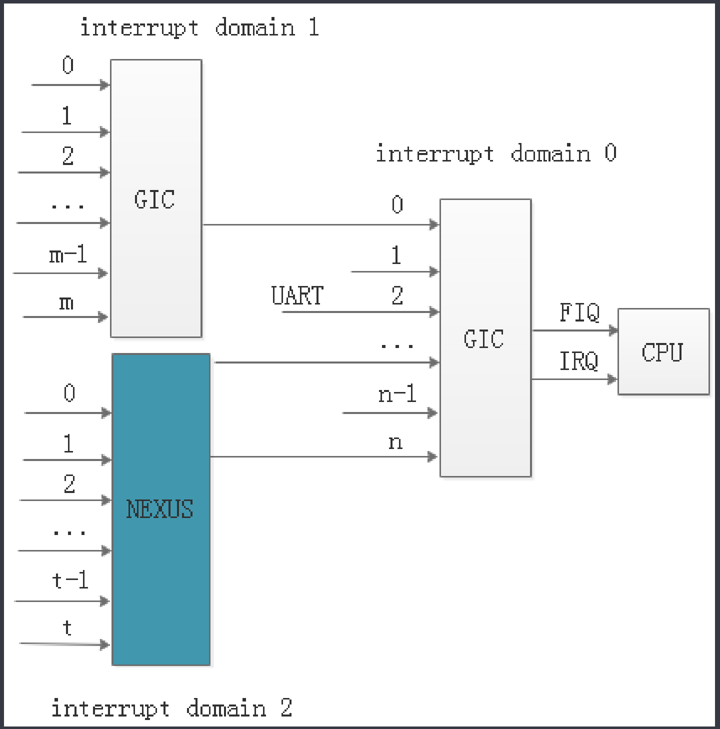
- 一个设备连接多个中断控制器

```
pmic@66 {
		compatible = "national,lp3974";
		interrupts-extended = <&gpx0 7 0>, <&gpx2 7 0>;
};

gpx0: gpx0 {
		gpio-controller;
		#gpio-cells = <2>;
		interrupt-controller;
		interrupt-parent = <&gic>;
		interrupts = <GIC_SPI  16  IRQ_TYPE_LEVEL_HIGH>,
				<GIC_SPI  17  IRQ_TYPE_LEVEL_HIGH>,
				<GIC_SPI  18  IRQ_TYPE_LEVEL_HIGH>,
				<GIC_SPI  19  IRQ_TYPE_LEVEL_HIGH>,
				<GIC_SPI  20  IRQ_TYPE_LEVEL_HIGH>,
				<GIC_SPI  21  IRQ_TYPE_LEVEL_HIGH>,
				<GIC_SPI  22  IRQ_TYPE_LEVEL_HIGH>,
				<GIC_SPI  23  IRQ_TYPE_LEVEL_HIGH>;
		#interrupt-cells = <2>;
};
```

13 设备树实例分析：中断映射

- 本节主要知识点
  - interrupt domain
  - interrupt-map：不同中断域之间的映射
  - interrupt-map-mask属性：<prop-encoded-array>

interrupt-map = <0 0 0 &gic 0 0 4>
<child unit address, child interrupt specifier, interrupt-parent,
parent unit address, parent interrupt specifier >

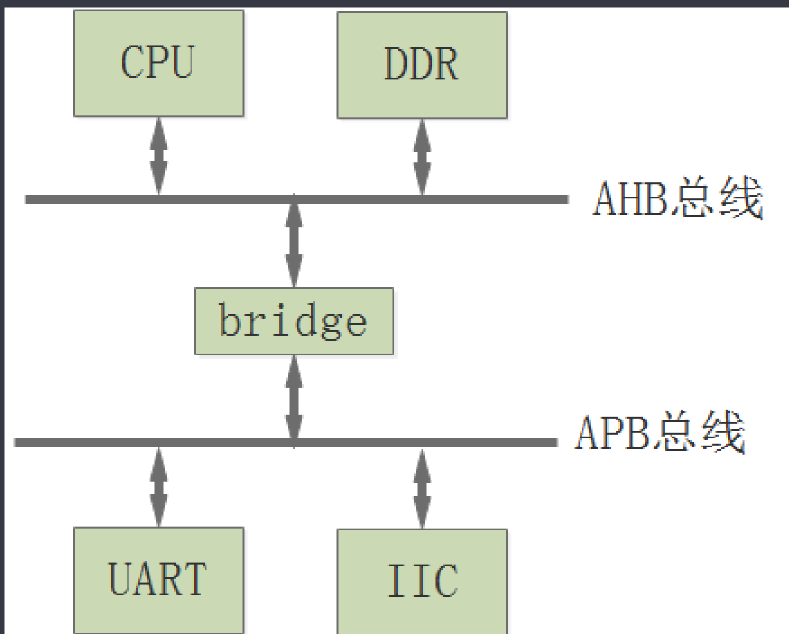


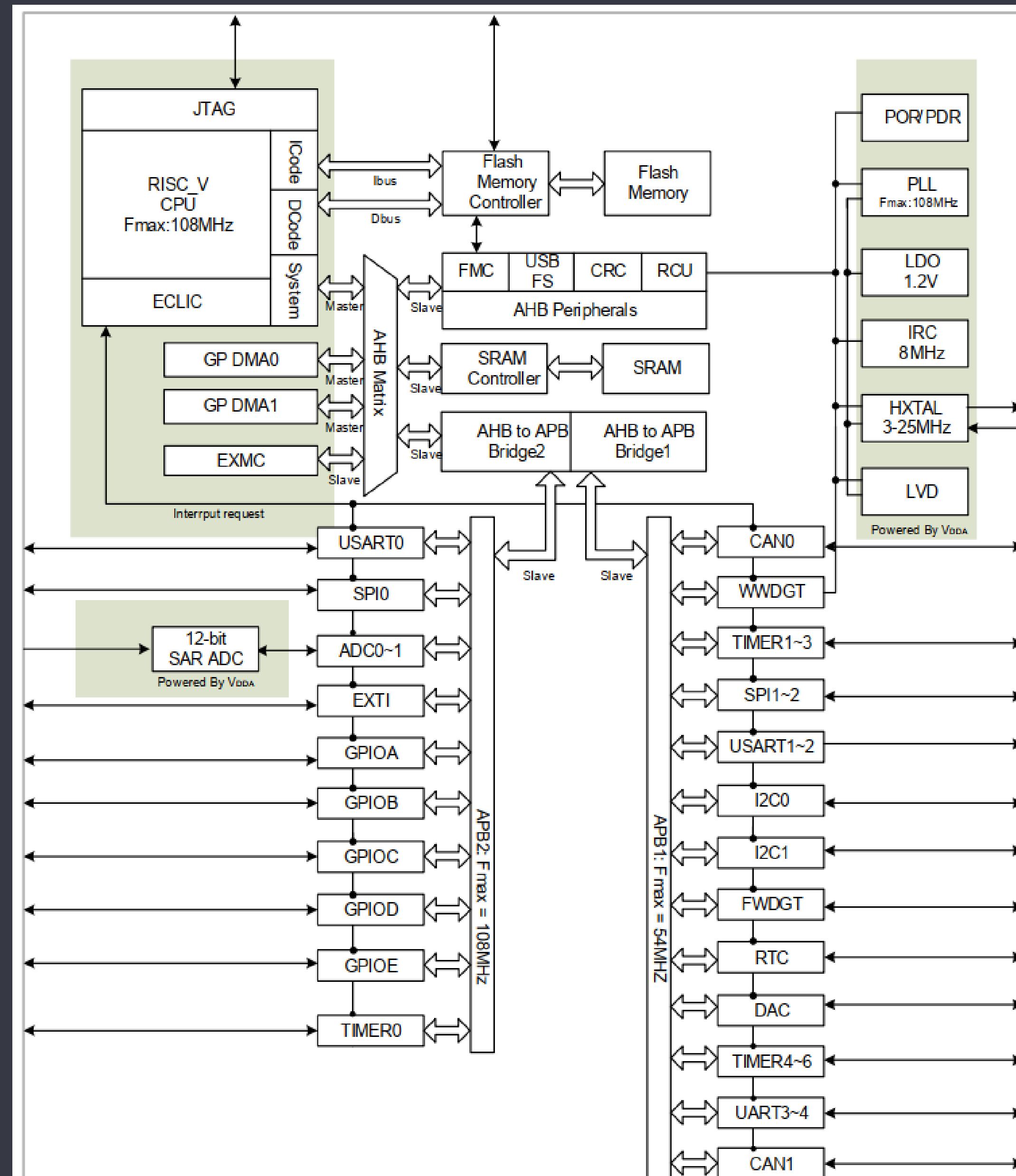
```
/ {
        interrupt-parent = <&gic>;
        #address-cells = <1>;
        #size-cells = <1>;
        #interrupt-cells = <3>;
        memory-controller@100e1000 {
                reg = <0x100e1000 0x1000>;
                interrupts = <0 45 4>,
        };
        bus@4000000 {
                #address-cells = <2>;
                #size-cells = <1>;
                #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  4 &gic 0  4 4>,
                                        ...
                motherboard {
                        #interrupt-cells = <1>;
                        iofpga@7,00000000 {
                                compatible = "simple-bus";
                                rtc@17000 {
                                        reg = <0x17000  0x1000>;
                                        interrupts = <4>;
                                };
                        };
                };
        };
};
```

- interrupt-map
  - child unit address ： 子node被映射的设备地址，通过bus node的 #address-cells 属性指定
  - child interrupt specifier： 通过 #interrupt-cells 属性指定
  - interrupt-parent：一个简单的phandle值，指向子映射的父 interrupt parent
  - parent unit address：父中断域的地址
  - parent interrupt specifier：用来描述父中断域

14 设备树实例分析：时钟

- SoC芯片中的clock





- 本节主要知识点
  - 芯片中的clock tree
  - clock provider
  - clock consumer
  - clock-names属性
  - clock-frequency属性
  - freq-range 属性
  - #clock-cells 属性

- clock provider & consumer

```
oscclk1: clcdclk {
		compatible = "arm,vexpress-osc";
		arm,vexpress-sysreg,func = <1 1>;
		freq-range = <10000000 80000000>;
		#clock-cells = <0>;
		clock-output-names = "clcdclk";
};

oscillator {
		#clock-cells = <1>;
		clock-output-names = "ckil", "ckih";
};

clcd@10020000 {
		compatible = "arm,pl111", "arm,primecell";
		reg = <0x10020000 0x1000>;
		clocks = <&oscclk1>, <&oscclk2>;
		clock-names = "clcdclk", "apb_pclk";
		//clock-ranges;
};
```

- multiple clock outputs

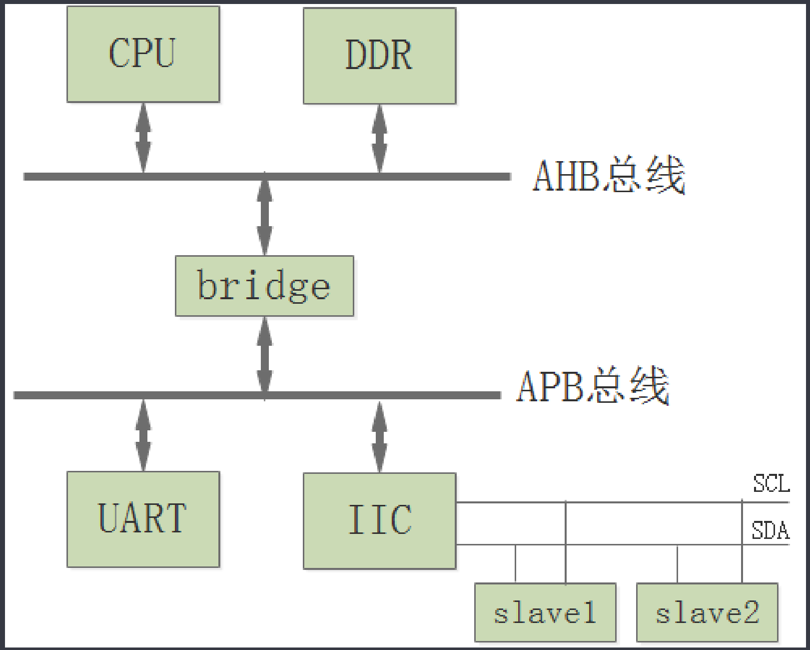
```
// clock provider
osc: oscillator {
		compatible = "myclocktype";
		#clock-cells = <1>;
		clock-indices = <1>, <3>;
		clock-output-names = "clka", "clkb";
};

// clock consumer
 device {
		clocks = <&osc 3>, <&ref 0>;
		clock-names = “clk", "register";
};
```

- external oscillator & pll

```
osc: oscillator {
		compatible = "fixed-clock";
		#clock-cells = <0>;
		clock-frequency  = <32678>;
		clock-output-names = "osc";
};
pll: pll@4c000 {
		compatible = "vendor,some-pll-interface"
		#clock-cells = <1>;
		clocks = <&osc 0>;
		clock-names = "ref";
		reg = <0x4c000 0x1000>;
		clock-output-names = "pll", "pll-switched";
};
uart@a000 {
		compatible = "fsl,imx-uart";
		reg = <0xa000 0x1000>;
		interrupts = <33>;
		clocks = <&osc 0>, <&pll 1>;
		clock-names = "baud", "register";
};
```

15 extend bus(上): I2C client

- 本节主要知识点总线地址
  - IIC控制器的作用
  - IIC 读写时序分析
  - IIC client地址




```
{
        #address-cells = <0x1>;
        #size-cells = <0x1>;
        compatible = "fsl,p1022-immr", "simple-bus";
        i2c@3100 {
                reg = <0x3100 0x100>;
        };
}
```

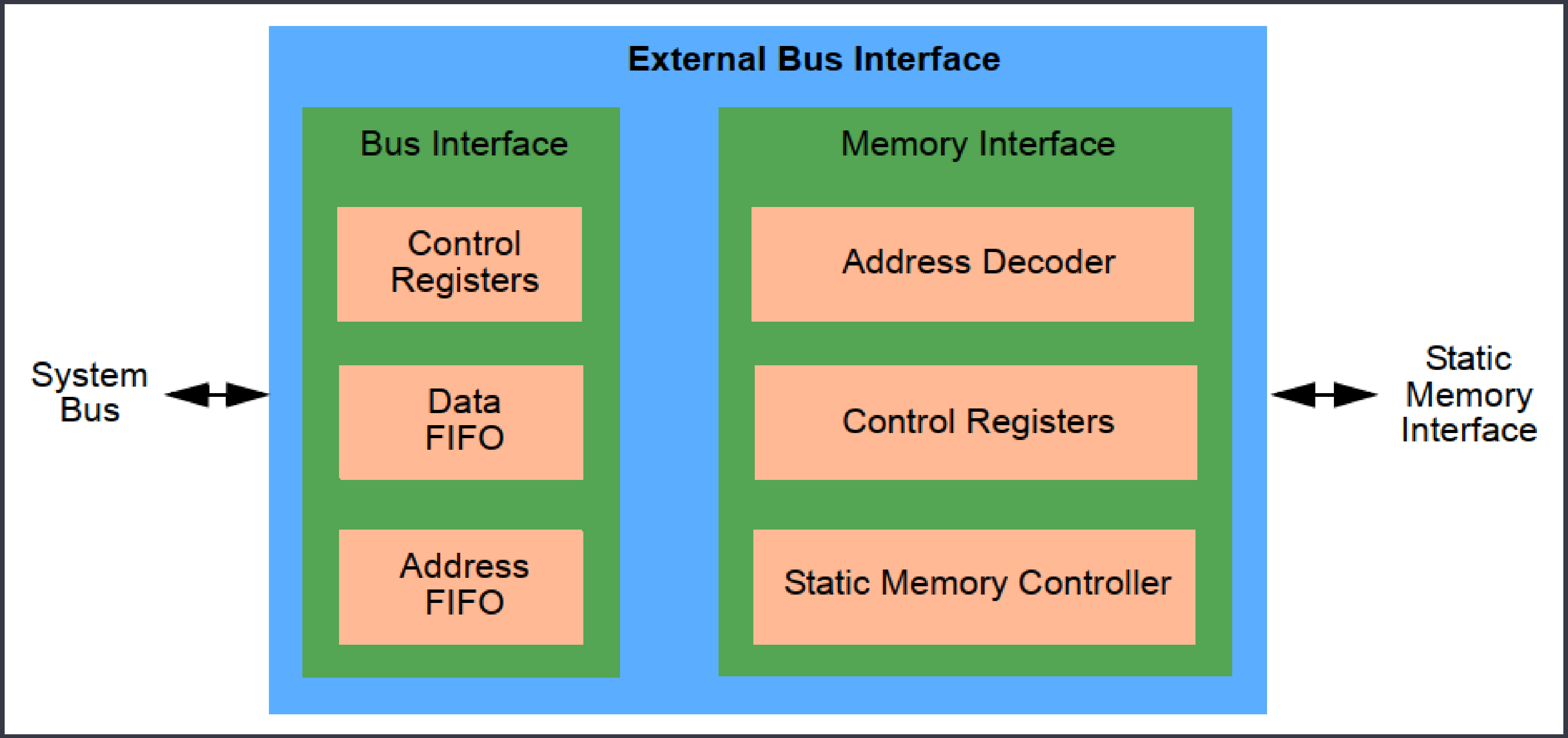
- IIC client demo

```
/* DVI I2C bus */
v2m_i2c_dvi: i2c@16000 {
		compatible = "arm,versatile-i2c";
		reg = <0x16000 0x1000>;
		#address-cells = <1>;
		#size-cells = <0>;

		dvi-transmitter@60 {
			compatible = "sil,sii9022-cpi", "sil,sii9022";
			reg = <0x60>;
		};
		rtc@68 {
			compatible = "dallas,ds1337";
			reg = <0x68>;
		};
};
```

16 extend bus(下)：memory mapped设备

- 本节主要知识点
  - CPU视角下的地址
  - 总线地址
  - memory mapped device
  - 不同地址域之间的转换
  - ranges属性
  - ranges属性和reg属性的组合使用

- SMB: static memory bus
  - Daughterboard
  - Motherboard
  - 作用：子板对母板的设备、内存访问



- 地址转换表：ranges属性

子总线地址(片选 偏移)　父地址　地址空间大小
(child-bus-address, parentbus-address, length)

```
#address-cells = <2>;
#size-cells = <1>;
ranges =  <0   0   0x10100000  0x10000        // Chipselect 1, Ethernet
           1   0   0x10160000  0x10000        // Chipselect 2, i2c controller
           2   0   0x30000000  0x1000000>; // Chipselect 3, NOR Flash
```

- SMB： static memory bus ranges属性分析

```
/ {
        bus@4000000 {
                #address-cells = <2>;
                #size-cells = <1>;
                ranges = <0 0 0x40000000 0x04000000>,
                         <1 0 0x44000000 0x04000000>,
                         <2 0 0x48000000 0x04000000>,
                         <7 0 0x10000000 0x00020000>;  //smb select cs7
                motherboard {    //0x10000000--0x10020000
                        #address-cells = <2>;   // SMB select number and offset
                        #size-cells = <1>;
                        ranges;
                        iofpga@7,00000000 {
                                #address-cells = <1>;
                                #size-cells = <1>;
                                ranges = <0  7  0  0x20000>;
                               // 0~0x20000   →  0x10000000~0x10020000
                                smb bus下的地址域     CPU视角下的地址域
                                rtc@17000 {
                                        reg = <0x17000  0x1000>;
                                };
                        };
                };
        };
};
```

17 设备树实例分析：GPIO

- 主要知识点
  - 如何描述GPIO controller
  - 外设如何引用GPIO管脚
  - gpio-specifier
  - gpio-controller属性
  - #gpio-cells属性
  - ngpios属性
  - gpios属性
  - enable-gpios属性

- gpio-specifier：如何描述GPIO controller

```
gpio-controller@00000000 {
        compatible = "foo";
        reg = <0x00000000 0x1000>;
        gpio-controller;
        #gpio-cells = <2>; // num   type
        ngpios = <18>;
        gpio-reserved-ranges = <0 4>, <12 2>;
        gpio-line-names = "MMC-CD", "MMC-WP", "VDD eth", "RST eth",
                          "LED R", "LED G", "LED B", "Col A", "Col B",
                          "Col C", "Col D", "Row A", "Row B", "Row C",
                          "Row D", "NMI button", "poweroff", "reset";
};
```

- 外设如何引用GPIO管脚

```
v2m_mmc_gpios: gpio@48 {
		compatible = "arm,vexpress-sysreg,sys_mci";
		reg = <0x048 4>;
		gpio-controller;
		#gpio-cells = <2>;
};
```

```
mmci@5000 {
		compatible = "arm,pl180", "arm,primecell";
		reg = <0x05000 0x1000>;
		interrupts = <9>, <10>;
		cd-gpios =  <&v2m_mmc_gpios  0 0>;
		wp-gpios = <&v2m_mmc_gpios 1 0>;
				<&v2m_mmc_gpios 2 0>;
				<&v2m_mmc_gpios 3 0>;
		clocks = <&v2m_clk24mhz>, <&smbclk>;
		clock-names = "mclk", "apb_pclk";
};
```

- 外设如何引用GPIO管脚

```
v2m_led_gpios: gpio@8 {
        compatible = "arm,vexpress-sysreg,sys_led";
        reg = <0x008 4>;
        gpio-controller;
        #gpio-cells = <2>;
};
```

```
leds {
        compatible = "gpio-leds";
        user1 {
                label = "v2m:green:user1";
                gpios = <&v2m_led_gpios 0 0>;
                linux,default-trigger = "heartbeat";
        };
        user2 {
                label = "v2m:green:user2";
                gpios = <&v2m_led_gpios 1 0>;
                linux,default-trigger = "mmc0";
        };
```

- 使用gpio管脚作为中断

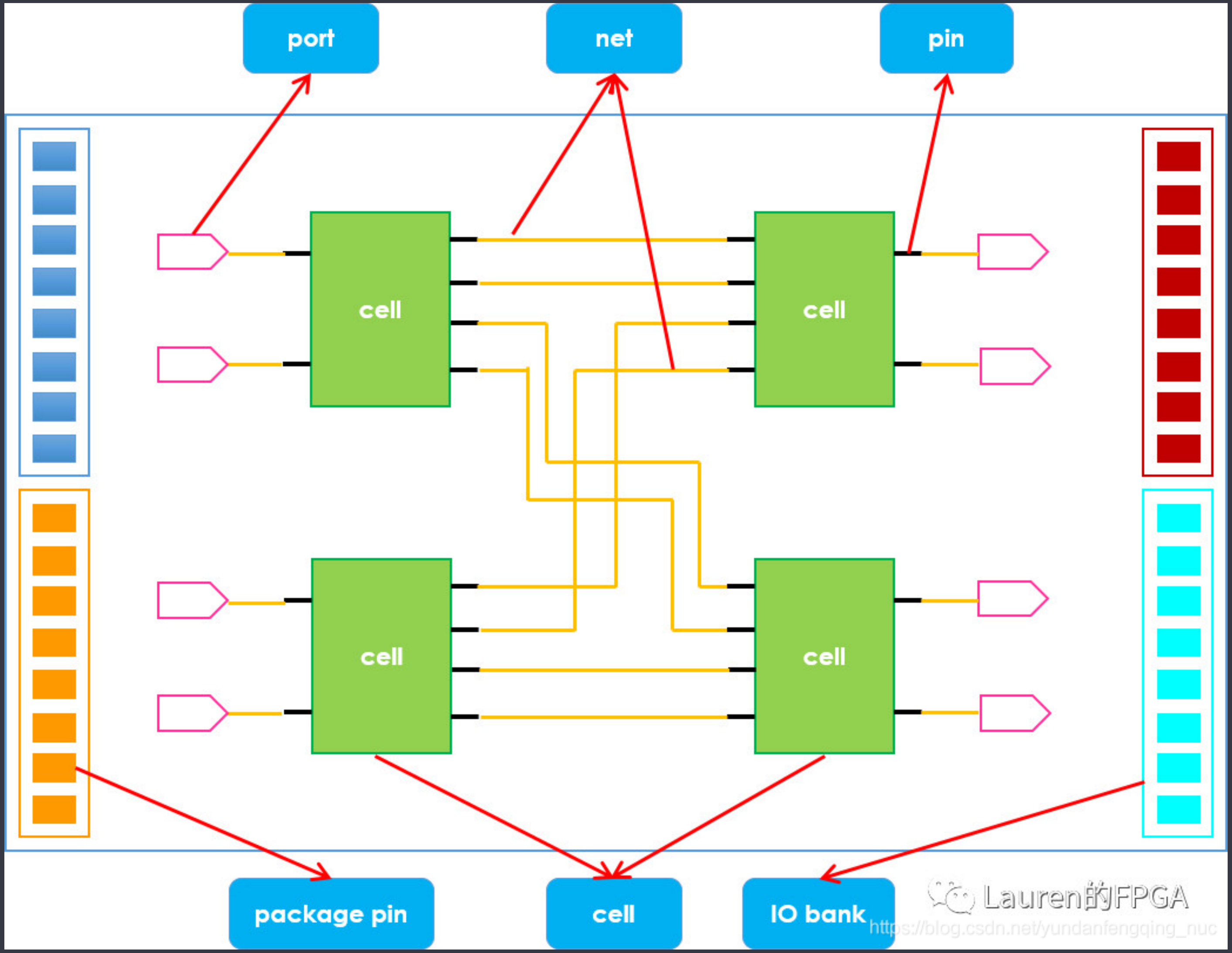
```
captouch: touchscreen@38 {
    compatible = "edt,edt-ft5406";
    reg = <0x38>;
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_irq_touch2 &pinctrl_emcon_gpio4>;
    interrupt-parent = <&gpio6>;
    interrupts = <31 IRQ_TYPE_EDGE_FALLING>;
    wake-gpios = <&gpio2 3 GPIO_ACTIVE_HIGH>;
    wakeup-source;
};
```

```
    gpio2: gpio@20a0000 {
        compatible = "fsl,imx6q-gpio", "fsl,imx35-gpio";
        reg = <0x020a0000 0x4000>;
        interrupts = <0 68 IRQ_TYPE_LEVEL_HIGH>,
                     <0 69 IRQ_TYPE_LEVEL_HIGH>;
        gpio-controller;
        #gpio-cells = <2>;
        interrupt-controller;
        #interrupt-cells = <2>;
    };
imx6qdl.dtsi                                          unix < utf-8 < c
```

18 设备树实例分析：pinmux

- 本节主要知识点
    - SoC 芯片术语：pin、pad、cell、net、port
    - pinmux的基本概念
    - IO 管脚复用的电路实现

- 芯片的一些术语



图片来源：Lauren的FPGA

- pinmux的基本概念

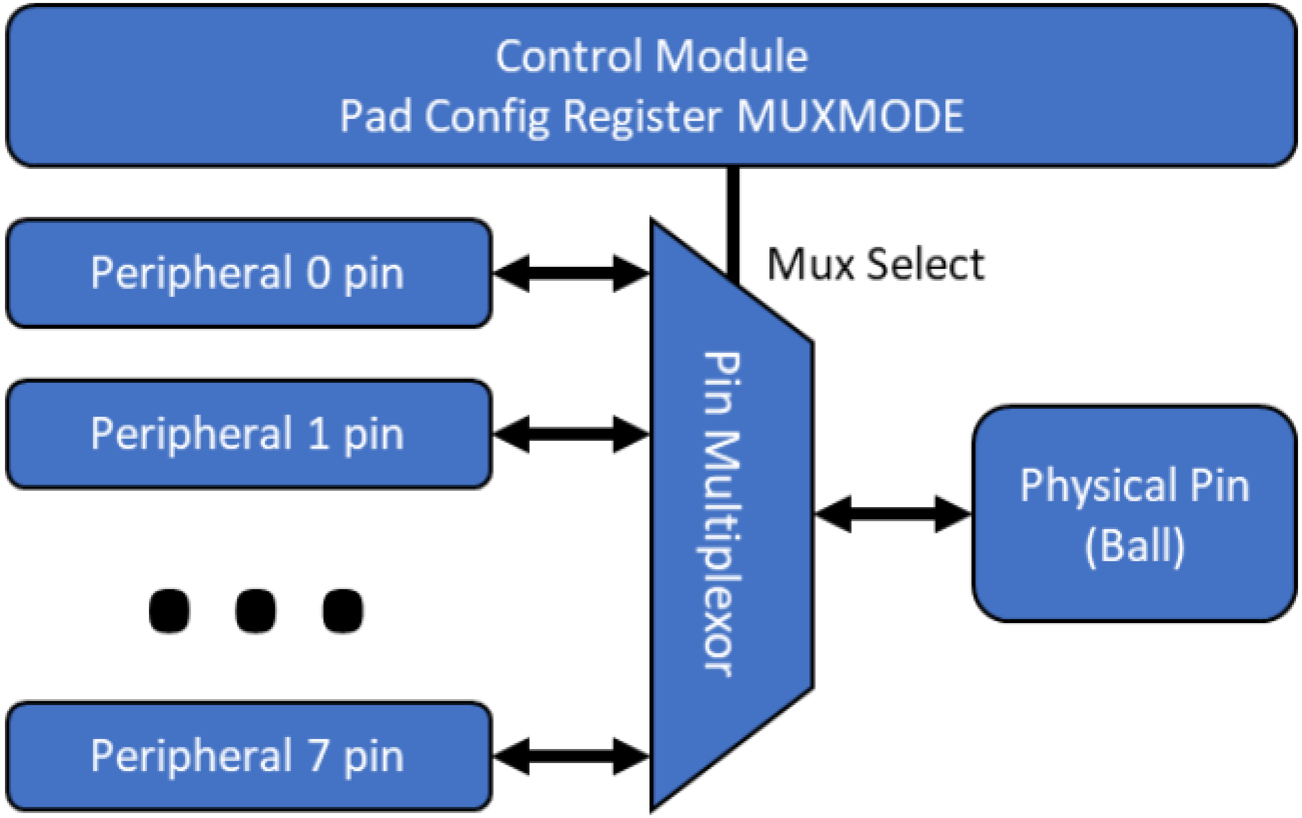


Figure 1:Pin Mux Block Diagram

- IO管脚复用的电路实现

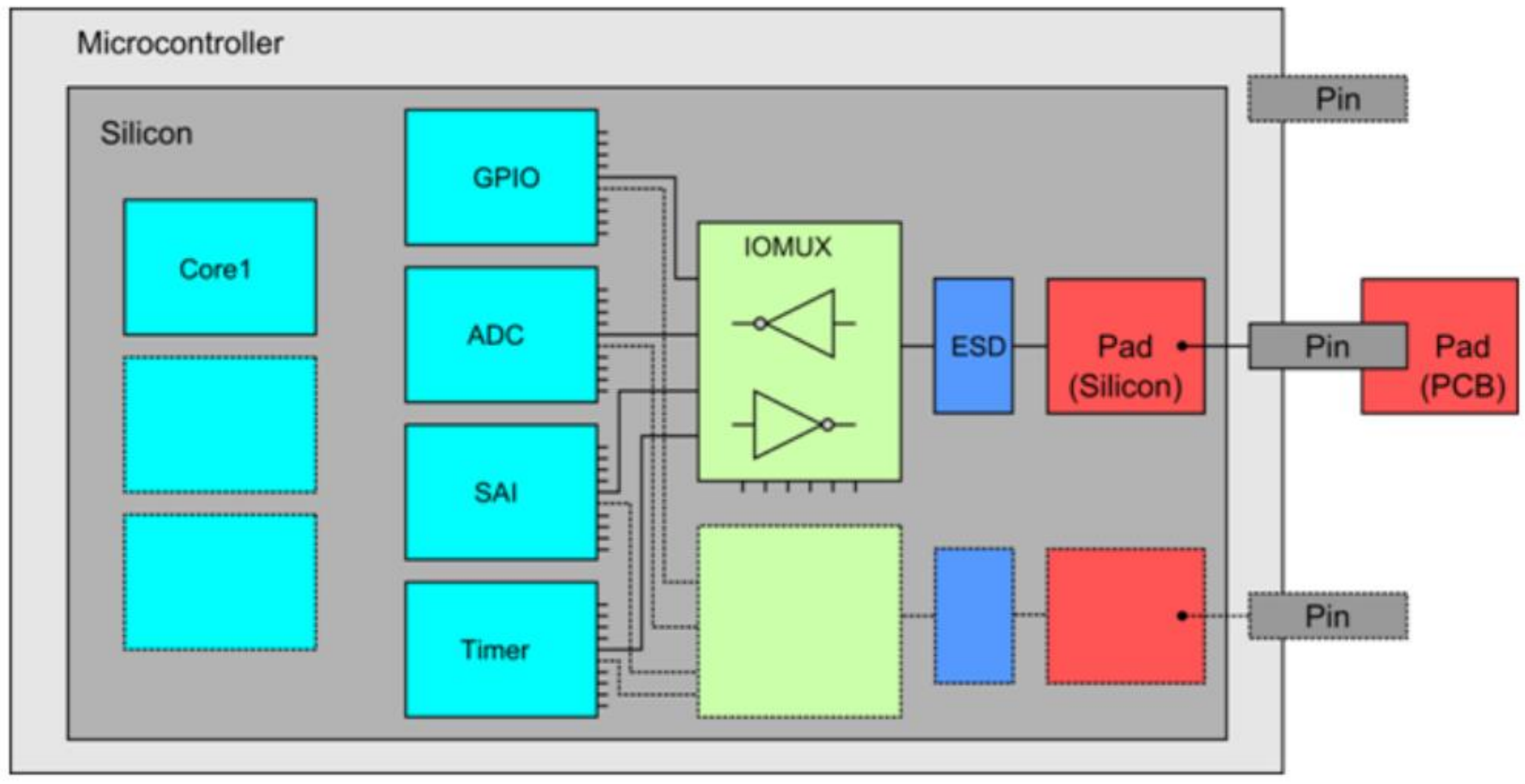
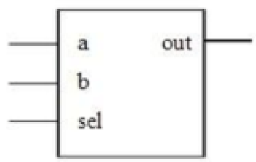
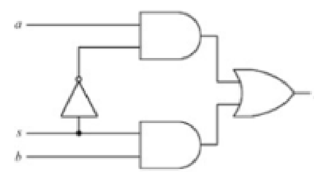


Figure 5. Alternate functions of a pin

• IO管脚复用的电路实现





$$Y = \bar{S}A + SB$$

S	A	B	Y
0	1	1	1
	1	0	1
	0	1	0
	0	0	0
1	1	1	1
	1	0	0
	0	1	1
	0	0	0

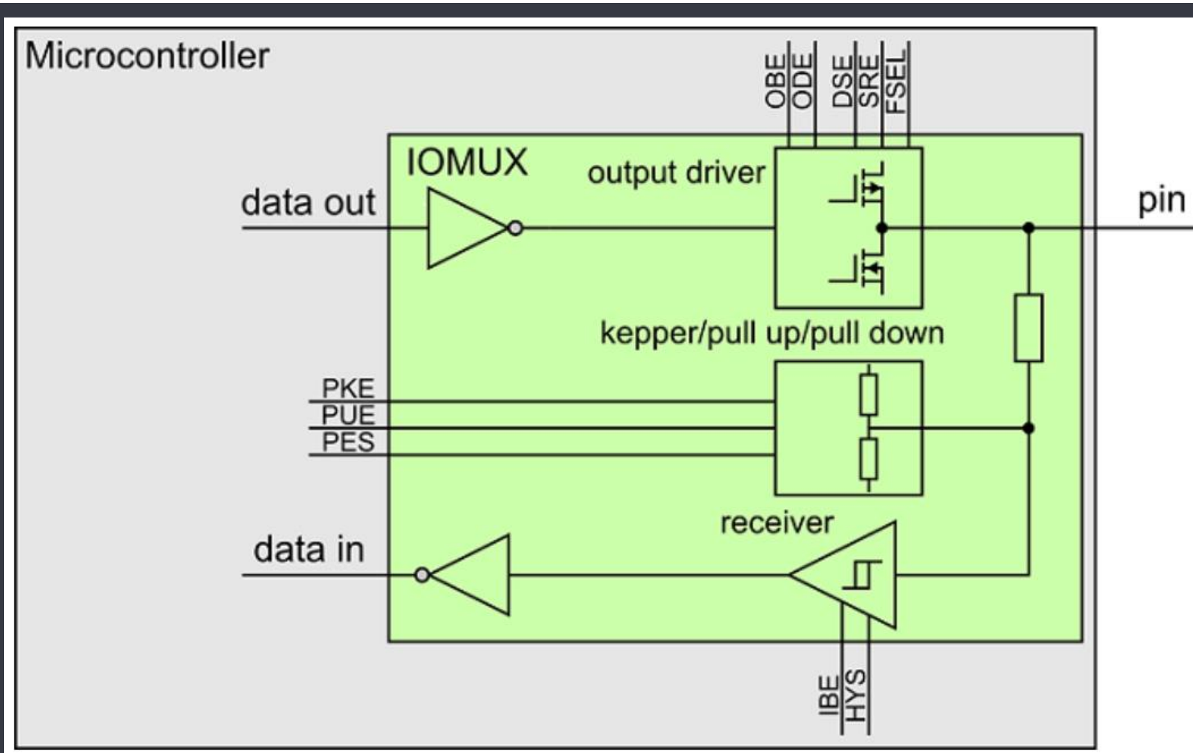


Figure 6. Simplified internal structure of the IOMUX for each pin

- 管脚的命名
  - 物理管脚：pin(pad) name、ball name
  - 逻辑管脚：signal name

Table 1. Assign pad, pin and alternate function in a datasheet

364 MAP BGA	176 LQFP	Pin name	Default	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	EzPort
T6	49	PTB0	–	PTB0	FTM0_CH0	ADC0_SE2	TRACECTL	LCD34	SAI2_RX_BCLK	VIU_DATA18	QSPI1_A_CS0	–

For example, the BGA package with 364 pads and the component datasheet information in Table 1. The physical pad T6 in the BGA package has the logical pad name PTB0.

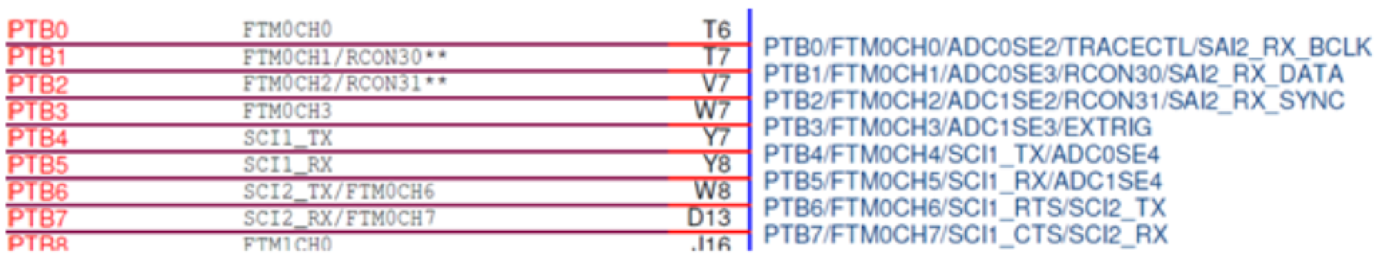


Figure 4. Assign pad, pin and alternate function in a schematic

19 设备树实例分析：pinmux(下)

- 本节主要知识点
  - 如何在设备树中描述pinmux
  - Linux中的pinctrl驱动作用
  - 如何配置PIN
  - 设备如何引用pinmux

- 设备树中的pin 配置

```
//am335x-baltos-leds.dtsi
am33xx_pinmux: pinmux@800 {
        compatible = "pinctrl-single";
        reg = <0x800 0x238>;
        #pinctrl-cells = <2>;
        pinctrl-single,register-width = <32>;
        pinctrl-single,function-mask = <0x7f>;
};

&am33xx_pinmux {
    user_leds: pinmux_user_leds {      //配置一个PIN state(configuration)
            pinctrl-single,pins = <
        AM33XX_PADCONF(AM335X_PIN_MII1_COL,  PIN_OUTPUT_PULLDOWN, MUX_MODE7) /* mii1_col.gpio3_0 PWR LED*/
          //  0x108(264)    PIN_OUTPUT_PULLDOWN    MUX_MODE7
        AM33XX_PADCONF(AM335X_PIN_MII1_TXD3, PIN_OUTPUT_PULLDOWN, MUX_MODE7) /* mii1_txd3.gpio0_16 WLAN LED */
        AM33XX_PADCONF(AM335X_PIN_MII1_TXD2, PIN_OUTPUT_PULLDOWN, MUX_MODE7) /* mii1_txd2.gpio0_17 APP LED */>;
    };

    // 一个设备所需要的PIN配置，单独一个node，add here
};
```

- 设备树中的pinctl和gpio的关联

```
//am335x-baltos-leds.dtsi
leds {
	pinctrl-names = “default”;  // 指明状态名字
	pinctrl-0 = <&user_leds>;   // 引用这个PIN state(configuration)
	compatible = "gpio-leds";
	power {
		label = "onrisc:red:power";
		linux,default-trigger = "default-on";
		gpios = <&gpio3 0 GPIO_ACTIVE_LOW>;
		default-state = "on";
	};
	wlan {
		label = "onrisc:blue:wlan";
		gpios = <&gpio0 16 GPIO_ACTIVE_HIGH>;
		default-state = "off";
	};
	app {
		label = "onrisc:green:app";
		gpios = <&gpio0 17 GPIO_ACTIVE_HIGH>;
		default-state = "off";
	};
};
```

- 多个pin state的配置方法

```
//am335x-evm.dts
cpsw_default: cpsw_default {
		pinctrl-single,pins = <
		AM33XX_PADCONF(AM335X_PIN_MII1_TX_EN, PIN_OUTPUT_PULLDOWN, MUX_MODE2)	/* mii1_txen.rgmii1_tctl */
		AM33XX_PADCONF(AM335X_PIN_MII1_RX_DV, PIN_INPUT_PULLDOWN, MUX_MODE2)	/* mii1_rxdv.rgmii1_rctl */>;
		};

cpsw_sleep: cpsw_sleep {
		pinctrl-single,pins = <
		AM33XX_PADCONF(AM335X_PIN_MII1_TX_EN, PIN_INPUT_PULLDOWN, MUX_MODE7) /* mii1_tx_en.gpio3_3 */
		AM33XX_PADCONF(AM335X_PIN_MII1_RX_DV, PIN_INPUT_PULLDOWN, MUX_MODE7)>;
		};

&mac {
		pinctrl-names = "default", "sleep";
		pinctrl-0 = <&cpsw_default>;
		pinctrl-1 = <&cpsw_sleep>;
		status = "okay";
		slaves = <1>;
};
```

- Linux 中的 pinctrl 子系统
    - pinctrl 在系统中的作用:drivers/pinctrl/
    - 和gpio子系统的关联
    - gpio 如何引用PIN的配置
    - pin管脚在什么时候配置的？

```
__device_attach
    bus_for_each_drv(dev->bus, NULL, &data, __device_attach_driver);
        __device_attach_driver
            driver_match_device(drv, dev);
            driver_probe_device(struct device_driver *drv, struct device *dev)
                really_probe
                    pinctrl_bind_pins    //读取设备树节点中的pin state，对pin管脚进行配置
                        drv->probe(dev) // 对设备进行初始化
```

20 dts和dtsi文件的分离

- 本节主要知识点
  - Vexpress FPGA motherboard：底板
  - Vexpress FPGA daughterboard：核心板
  - 对应的dts文件和dtsi文件分析
  - 好处
  - node的合并

21 property的overwrite

- 本节主要知识点
  - node的合并
  - property的overwrite
  - overwrite原则
    - 同层次节点的属性
    - 子节点的属性
  - 删除一个节点或属性

```
# make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- dtbs
# ./scripts/dtc/dtc –I dtb –O dts –o xxx.dts arch/arm/boot/dts/xxx.dtb
# ./scripts/dtc/dtc –I dts  –O dtb –o xxx.dtb  xxx.dts
```

- 实验
  - 节点属性的合并
  - 同级属性的覆盖
  - 子节点的属性

```
memory@30000000 {
        device_type = "memory";
        reg = <0x30000000 0x20000000>;
};

memory@30000000 {
        reg = <0x30000000 0x10000000>;
};
```

```
memory@30000000 {
        device_type = "memory";
        reg = <0x30000000 0x10000000>;
   };
```

```
xusbxti: oscillator@1 {
    compatible = "fixed-clock";
    reg = <1>;
    clock-frequency = <0>;
    clock-output-names = "xusbxti";
    #clock-cells = <0>;
};
```

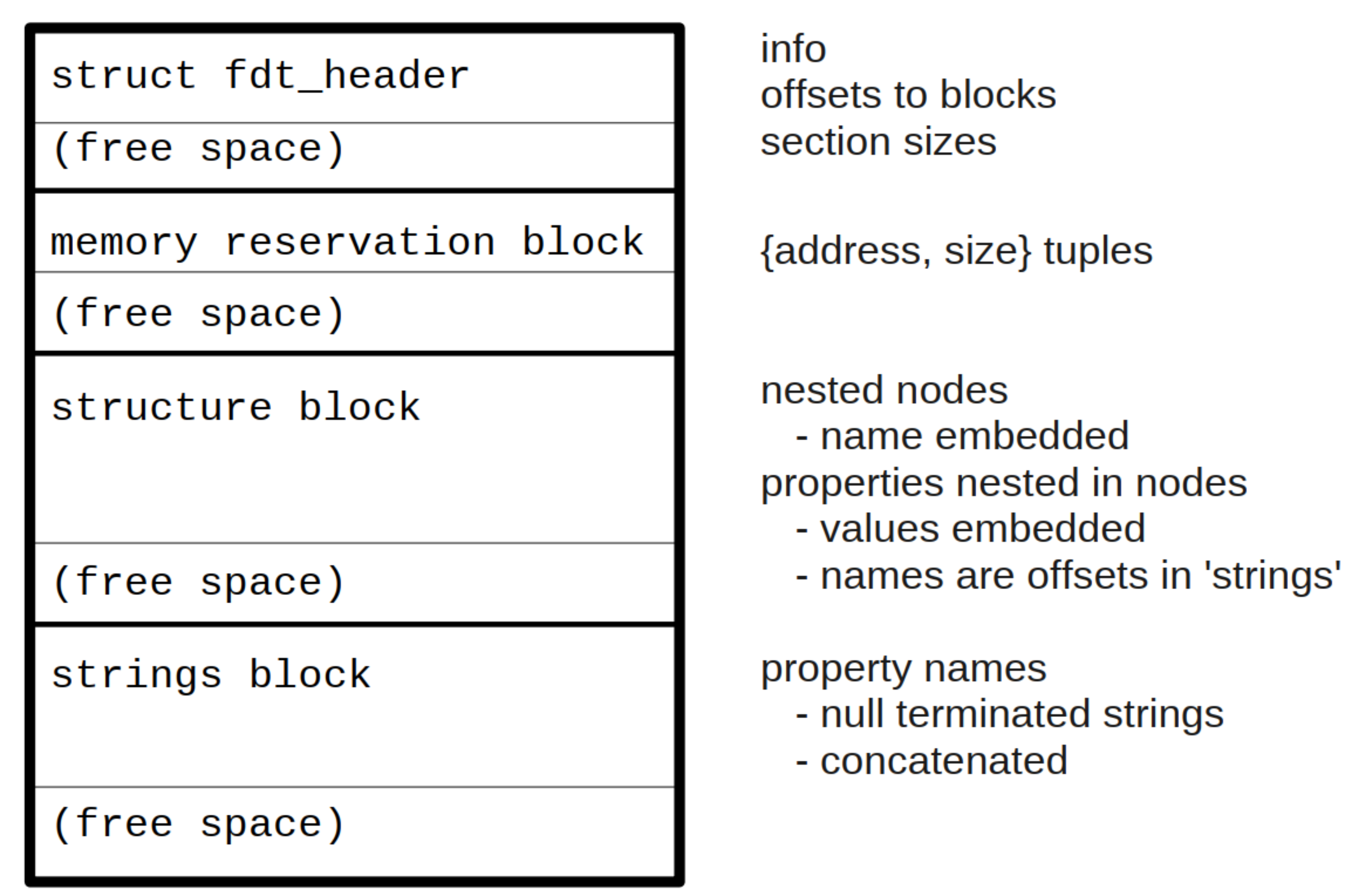
```
&xusbxti {
    clock-frequency = <24000000>;
};
```

- 实验
    - 删除一个node
    - 删除一个property

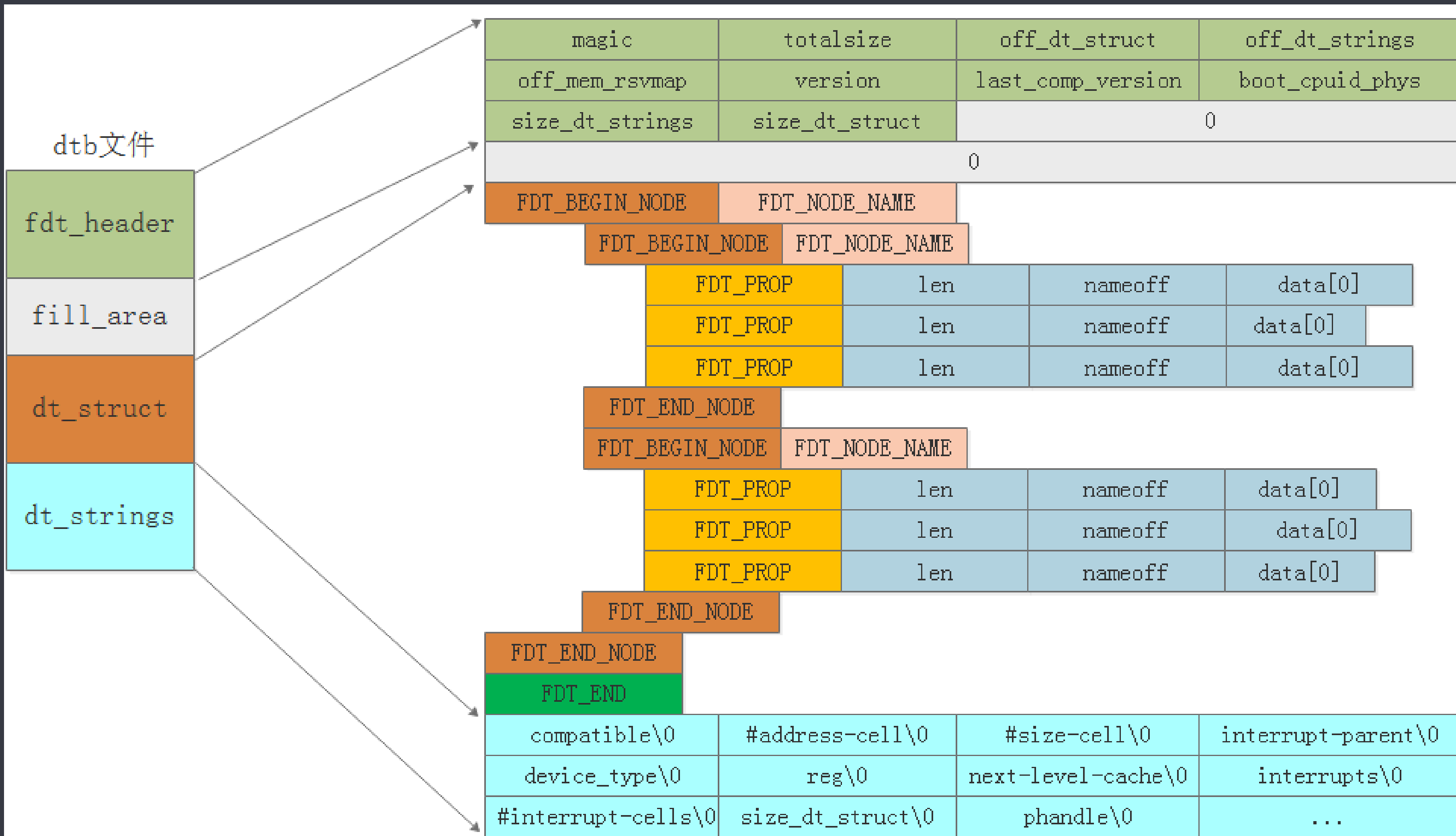
```
/ {
        label1: node1 {
                compatible = “zhaixue,node1";
                reg = <1>;
        };
        label2: node2{
                compatible = “zhaixue,node2";
                 reg = <1>;
        };
        &label2 {
                /delete-property/ reg;
        };

         /delete-node/ &label1    /node1
};
```

22 dtb文件格式

- 本节主要知识点
  - 从dts到dtb文件
  - dtb文件格式
  - 相关结构体



- dtb文件格式



参考：蜗窝科技/Device Tree(四)：文件结构解析

- 和dtb文件相关结构体

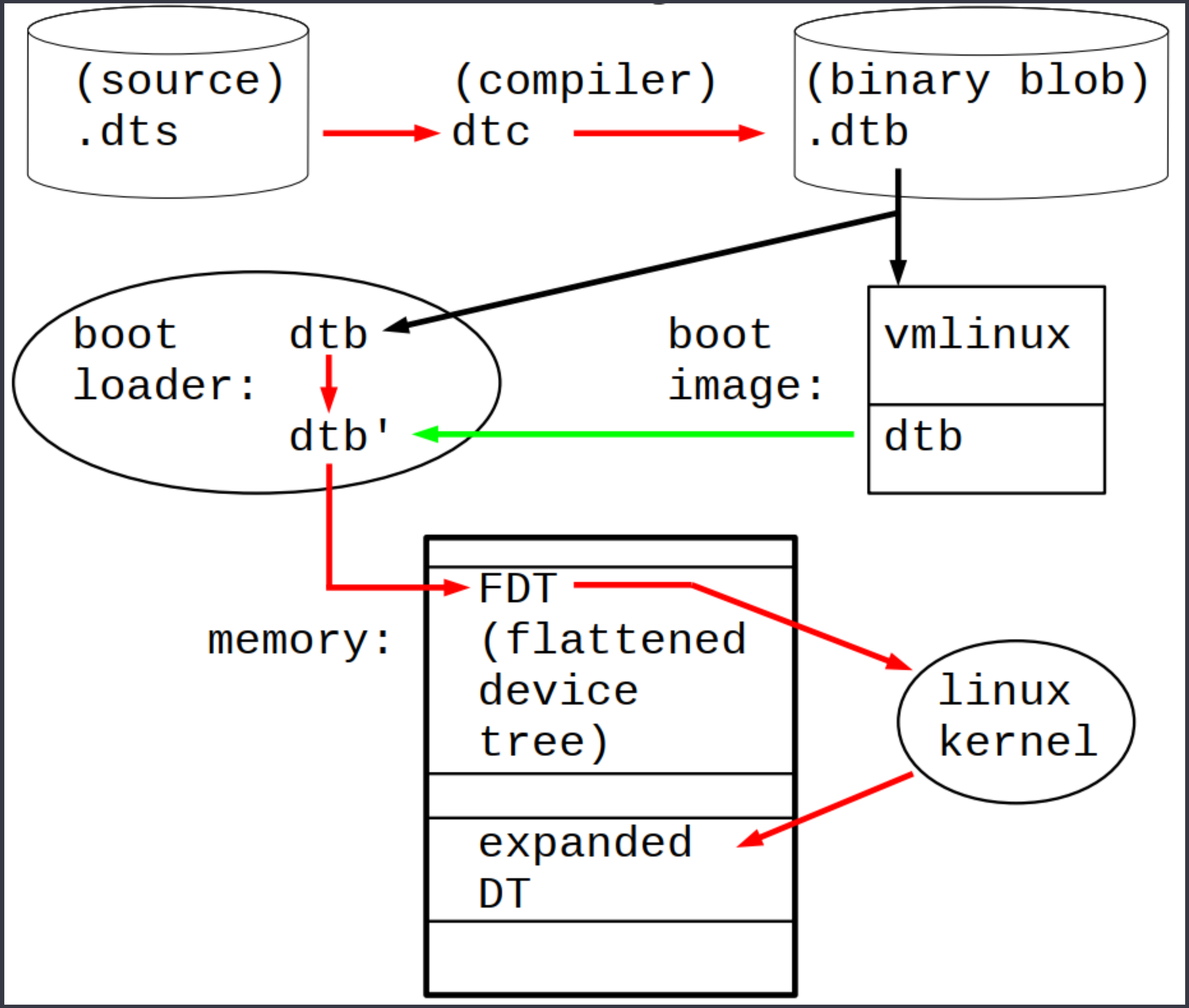
```
struct fdt_header {
        fdt32_t magic;                        /* magic word FDT_MAGIC */
        fdt32_t totalsize;                    /* total size of DT block */
        fdt32_t off_dt_struct;                /* offset to structure */
        fdt32_t off_dt_strings;               /* offset to strings */
        fdt32_t off_mem_rsvmap;               /* offset to memory reserve map */
        fdt32_t version;                      /* format version */
        fdt32_t last_comp_version;            /* last compatible version */

        /* version 2 fields below */
        fdt32_t boot_cpuid_phys;      /* Which physical CPU id we're booting on */
        /* version 3 fields below */
        fdt32_t size_dt_strings;      /* size of the strings block */
        /* version 17 fields below */
        fdt32_t size_dt_struct;               /* size of the structure block */
};
```

- 和dtb文件相关的宏定义

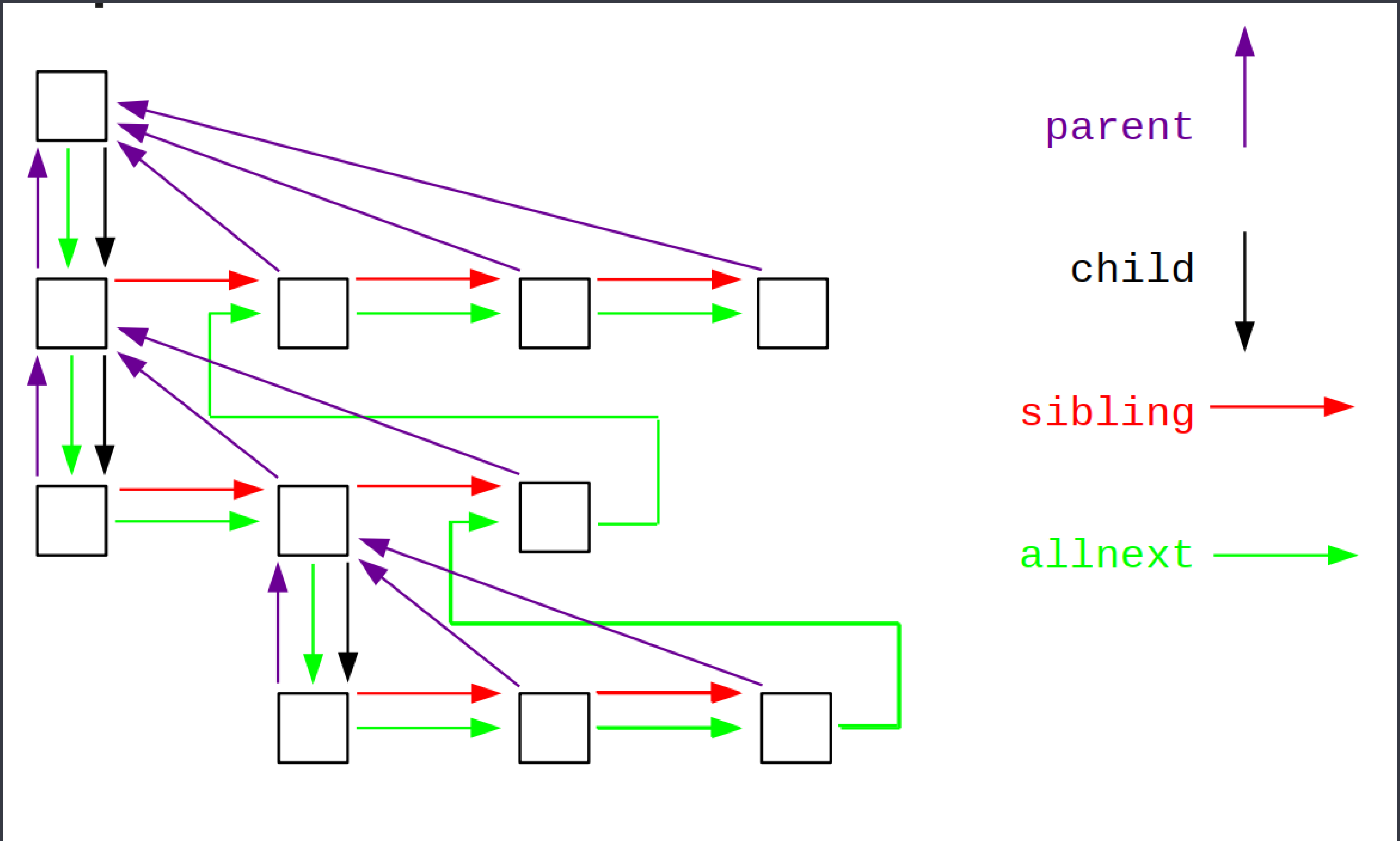
```
#define FDT_MAGIC          0xd00dfeed      /* 4: version, 4: total size */
#define FDT_TAGSIZE        sizeof(fdt32_t)
#define FDT_BEGIN_NODE     0x1             /* Start node: full name */
#define FDT_END_NODE       0x2             /* End node */
#define FDT_PROP           0x3             /* Property: name off, size, content */
#define FDT_NOP            0x4             /* nop */
#define FDT_END            0x9
```

23 platform device自动展开分析(上)

- 内核源码分析
  - 内核如何解析dtb文件
  - dtb文件如何展开为device_node



• 展开后的设备树(tree of struct device_node)



来源：dt_internals.pdf

- 解析过程分析01
  - start_kernel/setup_arch

```
setup_arch(&command_line);
        mdesc = setup_machine_fdt(__atags_pointer);//获取 machine_desc 描述符
        arm_memblock_init(mdesc); //保存设备树所在内存区域,以及其它保留的内存区域
        unflatten_device_tree();        //解析 dtb 文件,  创建 device_node 设备树
            __unflatten_device_tree(initial_boot_params, NULL, &of_root,
                early_init_dt_alloc_memory_arch, false);
                /* 第一次扫描 for size,扫描设备节点,统计总的设备树需要的内存大小 */
                size = unflatten_dt_nodes(blob, NULL, dad, NULL);
                  /* Allocate memory for the expanded device tree,申请内存 */
                mem = dt_alloc(size + 4, __alignof__(struct device_node));
                  /* 第二次扫描, do actual unflattening,初始化并建立组织树关系 */
                unflatten_dt_nodes(blob, mem, dad, mynodes);
```

- 解析过程分析02
  - 核心函数：unflatten_dt_nodes

```
static int unflatten_dt_nodes(const void *blob, void *mem,    struct device_node *dad,
                                                  struct device_node **nodepp)
{
        struct device_node *nps[FDT_MAX_DEPTH];        //设备树最大深度 64
                        //注意这里传的 mem 为 NULL 时, dryrun  为 1,表示光扫描,
                        //不初始化,不建立父子关系,只做统计
        offset = fdt_next_node(blob, offset, &depth)) {        //遍历 dtb 的所有节点
                      //分配并初始化设备树节点
        if (!populate_node(blob, offset, &mem, nps[depth], &nps[depth+1], dryrun))


        reverse_nodes(root);        //建立节点之间的关系,同父节点
}
```

- 解析过程分析03
  - 创建device_node并添加到设备树：populate_node

```
static bool populate_node(const void *blob, int offset, void **mem, struct device_node *dad,
                    struct device_node **pnp, bool dryrun)
{
    np = unflatten_dt_alloc(mem, sizeof(struct device_node) + allocl,
                    __alignof__(struct device_node)); //创建 device_node 结构体
    if (!dryrun) {
        char *fn;
        of_node_init(np);
        np->full_name = fn = ((char *)np) + sizeof(*np);
        memcpy(fn, pathp, l);
        if (dad != NULL) {   //将新创建的 device_node 结构体添加到内核设备树中
            np->parent = dad;
            np->sibling = dad->child;
            dad->child = np;
        }
    }
    populate_properties(blob, offset, mem, np, pathp, dryrun); //解析该节点的属性
    if (!dryrun) {
        np->name = of_get_property(np, "name", NULL);
    }
    *pnp = np;
    return true;
}
```

- # 解析过程分析04
  - ## 解析node的属性并添加到链表：populate_node

```
static void populate_properties(const void *blob, int offset, void **mem, struct device_node
                                *np, const char *nodename, bool dryrun)
{
	struct property *pp, **pprev = NULL;
	pprev = &np->properties;
	for (cur = fdt_first_property_offset(blob, offset);
	     cur >= 0;
	     cur = fdt_next_property_offset(blob, cur)) {
		const __be32 *val;
		const char *pname;
		u32 sz;
		val = fdt_getprop_by_offset(blob, cur, &pname, &sz);
		pp = unflatten_dt_alloc(mem, sizeof(struct property),__alignof__(struct property));
		if (dryrun)
			continue;
		pp->name     = (char *)pname;
		pp->length = sz;
		pp->value    = (__be32 *)val;    //核心代码：将解析到的 property 依次添加到链表
		*pprev       = pp;
		pprev        = &pp->next;
	}
...
	if (!dryrun)
		*pprev = NULL;
}
```

24 platform device自动展开分析(下)

- 内核源码分析
  - 如何从设备树中创建platform device
  - 如何初始化resource资源
  - platform device如何添加到platform 总线

• 内核源码分析

```
start_kernel->rest_init->kernel_init->kernel_init_freeable->do_basic_setup->do_initcalls->do_initcall_level

arch_initcall_sync(of_platform_default_populate_init);
of_platform_default_populate_init
   of_platform_default_populate
                of_platform_populate
                   for_each_child_of_node(root, child) {               //遍历根节点下的所有子节点
                                  rc = of_platform_bus_create(child, matches, lookup, parent, true); //递归处理子节点
                                     of_platform_device_create_pdata  //展开为对应的platform_device

 of_platform_device_create_pdata
                of_device_alloc(np, bus_id, parent); //创建初始化platform_device及resources， 关联device_node
                of_device_add                                     //添加platform device到platform总线上
```

25 I2C设备的自动展开

- 本节主要知识点
    - extend bus：主从设备
    - I2C总线与I2C设备(i2c_client)
    - SPI总线与SPI设备(spi_device)

- 总线型设备驱动
  - IIC设备： i2c_client->irq
  - SPI设备： spi_device->irq
  - API接口： of_irq_get

```
drivers/i2c/busses/i2c_versatile.c
struct i2c_client {
        unsigned short flags;                /* div., see below    */
        unsigned short addr;                 /* chip address - NOTE: 7bit   */
        char name[I2C_NAME_SIZE];
        struct i2c_adapter *adapter;         /* the adapter we sit on*/
        struct device dev;                   /* the device structure*/
        int init_irq;                        /* irq set at initialization*/
        int irq;                             /* irq issued by device*/
        struct list_head detected;
#if IS_ENABLED(CONFIG_I2C_SLAVE)
        i2c_slave_cb_t slave_cb;             /* callback for slave mode*/
#endif
};
```

26 哪些node会自动展开为platform_device？

- 设备树的哪些node会自动展开？
  - 不是所有的node都自动展开
  - 展开规则
    - 根节点下含有compatible属性的子节点：AHB/AXI
    - 包含以下bus标识的node的所有子节点(有compatible属性)
      - » PCI、SMB

```
// drivers/of/ platform.c
of_platform_populate(root, of_default_bus_match_table, lookup,parent);

const struct of_device_id of_default_bus_match_table[] = {
        { .compatible = “simple-bus”, },   //simple-bus是什么意思？
        { .compatible = “simple-mfd”, },  //子节点中为什么需要compatible？
        { .compatible = "isa", },
#ifdef CONFIG_ARM_AMBA
        { .compatible = "arm,amba-bus", },
#endif /* CONFIG_ARM_AMBA */
        {} /* Empty terminated list */
};
```

- amba_device
  - Primecells IP控制器
  - compatible = "arm,amba-primecell“
  - platform_device & amba_device

```
of_platform_populate -> of_platform_bus_create:

if (of_device_is_compatible(bus, "arm,primecell")) {
		of_amba_device_create(bus, bus_id, platform_data, parent);
		return 0;
}

dev = of_platform_device_create_pdata(bus, bus_id, platform_data, parent);
		if (!dev || !of_match_node(matches, bus))
		return 0;
```

27 设备树节点解析示例：CPU node

- 本节主要知识点
    - 如何去解析一个node？
    - 如何去解析一个property？
    - 设备树编程接口
    - 设备树相关的头文件

28 设备树节点解析示例：memory node

- 本节主要知识点
  - 如何读写property属性值
  - 如何访问memory node
  - 内核对memory mode的解析
  - U-boot对memory node的读写

- U-boot对device tree的支持

```
spl.c /board_init_r
        ->spl_fixup_fdt
            ->arch_fixup_fdt
                ->fdt_fixup_memory_banks：根据 memory banks 来设置 reg 属性大小
/* common/fdt_support.c */
int fdt_fixup_memory_banks(void *blob, u64 start[], u64 size[], int banks)
{
	int err, nodeoffset;
	int len, i;
	u8 tmp[MEMORY_BANKS_MAX * 16]; /* Up to 64-bit address + 64-bit size */
	if (banks > MEMORY_BANKS_MAX) {
		printf("%s: num banks %d exceeds hardcoded limit %d."
		       " Recompile with higher MEMORY_BANKS_MAX?\n",
		       __FUNCTION__, banks, MEMORY_BANKS_MAX);
		return -1;
	}
```

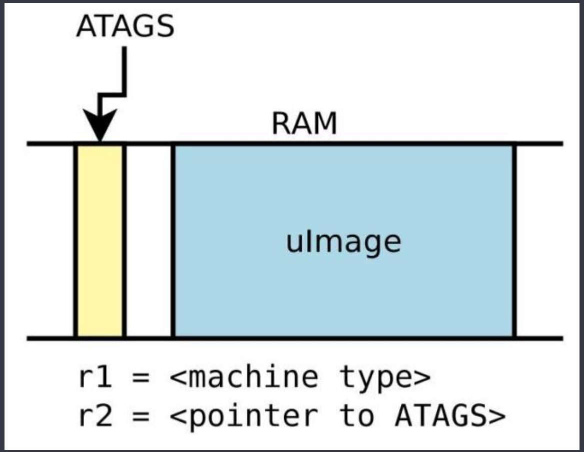
- U-boot对device tree的支持

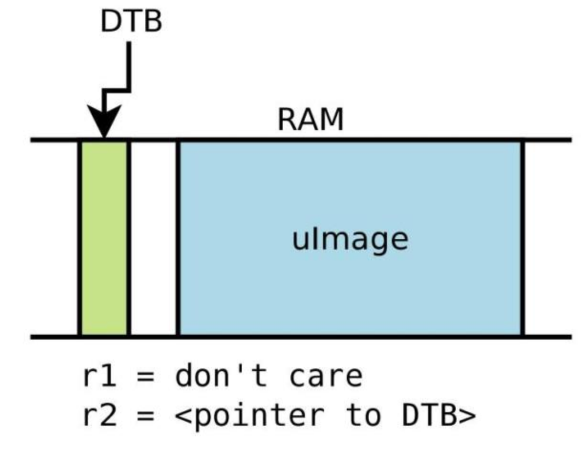
- bootm fdt： relocates the flattened device tree
- bootm go： performs fix-up actions and boots the operating system
- fdt addr <addr> [<length>]： sets the FDT location to <addr>
- fdt boardsetup： performs board-specific setup
- fdt move <fdt> <newaddr> <length>： copies the FDT to <addr> and makes it active
- fdt resize： resizes the FDT to size + padding to 4 K address
- fdt print <path> [<prop>]： recursive print starting at <path>
- fdt set <path> <prop> [<val>]： sets <property> [to <val>]
- fdt mknode <path> <node>： creates a new node after <path>
- fdt rm <path> [<prop>]： deletes the node or <property>
- fdt header： displays header information
- fdt chosen [<start> <end>]： adds/updates the /chosen branch in the tree
    - <start>/<end>： initrd the start/end address

29 设备树节点解析示例：chosen node(上)

- 本节主要知识点
  - chosen node的作用
  - U-boot对choose node的支持
  - 引入DT后，内核启动方式的变化
  - 内核对chosen node的解析过程

```
chosen {
        bootargs = "console=ttyS0,115200 loglevel=8";
        initrd-start = <0xc8000000>;
        initrd-end = <0xc8200000>;
};
```



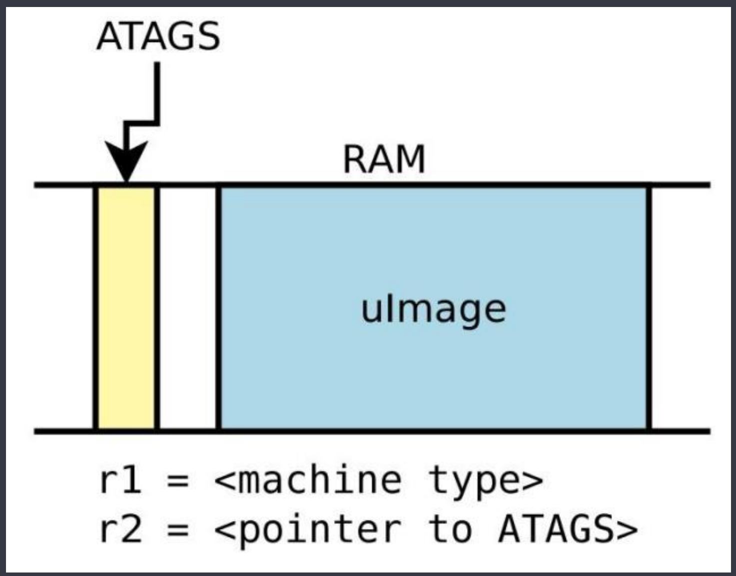


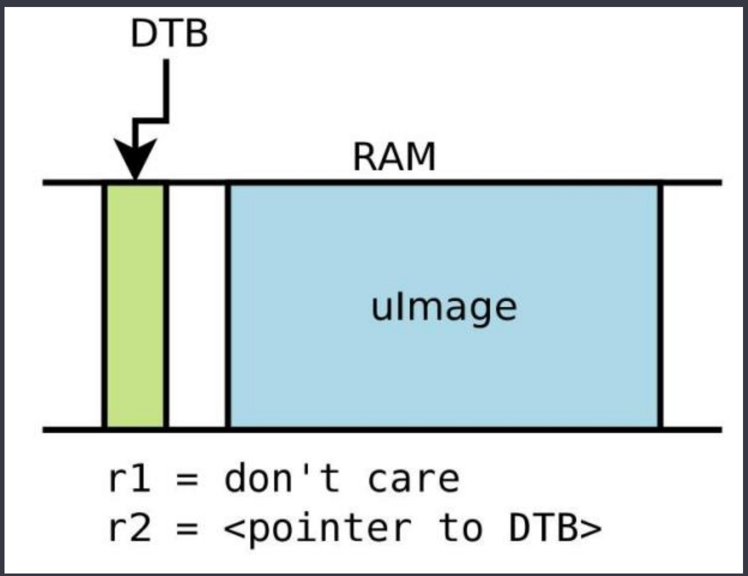
- U-boot对chosen节点的修改

```
u-boot/cmd/ bootm.c:
U_BOOT_CMS( bootm, CONFIG_SYS_MAXARGS, 1, do_bootm)
do_bootm->do_bootm_states
                        boot_fn = bootm_os_get_boot_func(images->os.os);
                        boot_fn(BOOTM_STATE_OS_PREP, argc, argv, images);
arch/arm/lib/bootm.c:
                do_bootm_linux
                    boot_prep_linux
                            image_setup_linux
                                    image_setup_libfdt
                                          fdt_chosen
                    boot_jump_linux
                            r2 = images->fdt_addr
                            kernel_entry(0,machid,r2);
```

30设备树节点解析示例：chosen node(下)

- 引入DT前后内核启动方式的变化





Linux uses DT data for three major purposes:

1) platform identification,
2) runtime configuration,
3) device population.

• 引入DT前后内核启动方式的变化

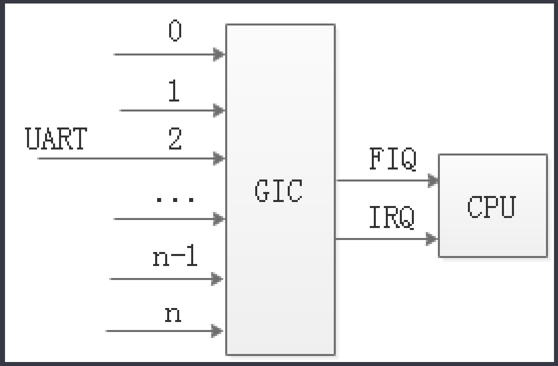
```
struct machine_desc {
	unsigned int		nr;			/* architecture number	*/
	const char		*name;			/* architecture name	*/
	unsigned long		atag_offset;		/* tagged list (relative) */
	const char *const	*dt_compat;		/* array of device tree  * 'compatible' strings */
	unsigned int		nr_irqs;		/* number of IRQs */
	unsigned int		video_start;		/* start of video RAM	*/
	unsigned int		video_end;		/* end of video RAM	*/

	unsigned char		reserve_lp0 :1;		/* never has lp0	*/
	unsigned char		reserve_lp1 :1;		/* never has lp1	*/
	unsigned char		reserve_lp2 :1;		/* never has lp2	*/
	enum reboot_mode	reboot_mode;		/* default restart mode	*/
	unsigned		l2c_aux_val;		/* L2 cache aux value	*/
	unsigned		l2c_aux_mask;		/* L2 cache aux mask	*/
	void			(*l2c_write_sec)(unsigned long, unsigned);
	const struct smp_operations	*smp;	/* SMP operations	*/
	bool			(*smp_init)(void);
	void			(*fixup)(struct tag *, char **);
	void			(*dt_fixup)(void);
	long long		(*pv_fixup)(void);
	void			(*reserve)(void);/* reserve mem blocks	*/
	void			(*map_io)(void);/* IO mapping function	*/
	void			(*init_early)(void);
	void			(*init_irq)(void);
	void			(*init_time)(void);
	void			(*init_machine)(void);
	void			(*init_late)(void);
	void			(*restart)(enum reboot_mode, const char *);
};
```

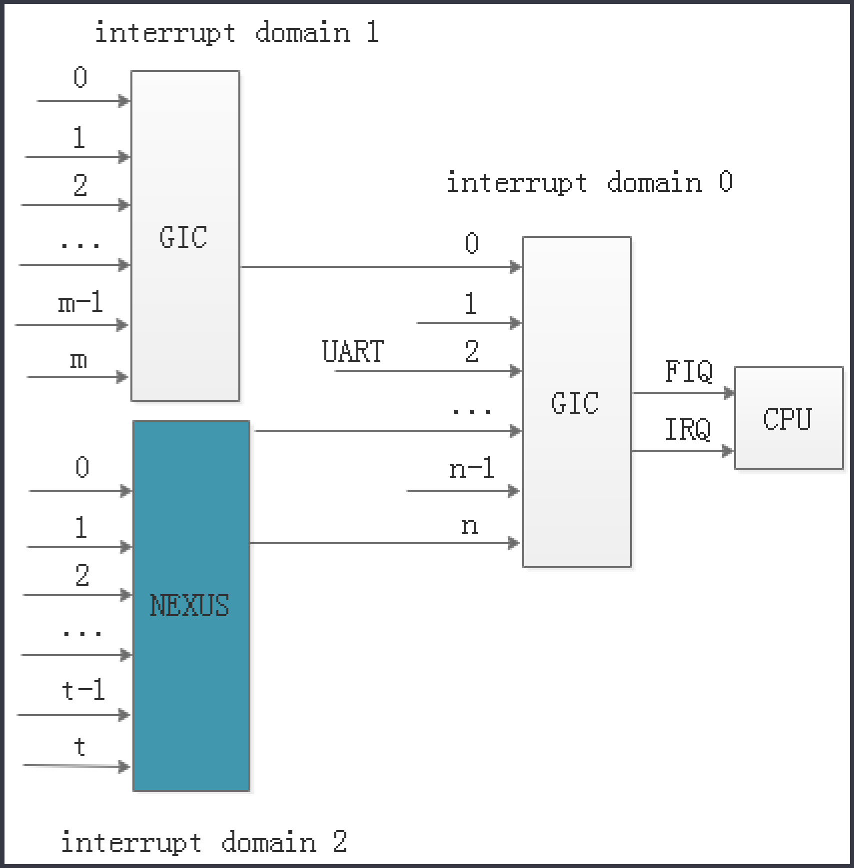
31设备树节点解析示例：aliases node

- 本节主要知识点
  - aliases node的作用
  - 如何对aliases node进行访问

32设备树节点解析：获取IRQ number

- 本节主要知识点
  - 区别：IRQ number 和 HW interrupt ID
  - 如何获取IRQ number？
  - 内核对中断源hwirq的统一管理





- 内核对中断源hwirq的统一管理
    - IRQ domain
    - 线性映射
    - Radix Tree map
    - no map
    - 映射分析：of_irq_get

33设备树节点解析：获取register地址

- 本节主要知识点
    - 如何搜索指定的device node节点
    - 获取寄存器起始地址
    - 寄存器地址的映射

34设备树节点解析：GPIO

- 本节主要知识点
  - 如何解析gpio节点和属性
  - 读写gpio的内核API编程接口
  - 读写gpio的内核新接口
  - 驱动如何读写gpio管脚

- 操作gpio的两套接口
  - 老接口(legency)
    - API 接口函数以”gpio_”为前缀
    - 使用一个整数来表示一个管脚
    - 编程示例
  - 基于描述符(descriptor-based)
    - API 接口函数以”gpiod_”为前缀
    - 使用gpio_desc表示一个管脚
    - 编程示例

35 device bingdings & guidelines

- 本节主要知识点
  - 本期课程的主要内容
  - 关于device tree的知识框架
  - Device Bindings & Binding Guidelines
  - 关于device tree的未来展望
  - Device tree ABI接口

- Binding Guidelines
  - Documentation/devicetree/bindings/submitting-patches.rst

```
Overall design
==============

- DO attempt to make bindings complete even if a driver doesn't support some
  features. For example, if a device has an interrupt, then include the
  'interrupts' property even if the driver is only polled mode.

- DON'T refer to Linux or "device driver" in bindings. Bindings should be
  based on what the hardware has, not what an OS and driver currently support.

- DO use node names matching the class of the device. Many standard names are
  defined in the DT Spec. If there isn't one, consider adding it.

- DO check that the example matches the documentation especially after making
  review changes.

- DON'T create nodes just for the sake of instantiating drivers. Multi-function
  devices only need child nodes when the child nodes have their own DT
  resources. A single node can be multiple providers (e.g. clocks and resets).

- DON'T use 'syscon' alone without a specific compatible string. A 'syscon'
  hardware block should have a compatible string unique enough to infer the
  register layout of the entire block (at a minimum).
```

Binding Guidelines

```
Properties
==========

- DO make 'compatible' properties specific. DON'T use wildcards in compatible
  strings. DO use fallback compatibles when devices are the same as or a subset
  of prior implementations. DO add new compatibles in case there are new
  features or bugs.

- DO use a vendor prefix on device specific property names. Consider if
  properties could be common among devices of the same class. Check other
  existing bindings for similar devices.

- DON'T redefine common properties. Just reference the definition and define
  constraints specific to the device.

- DO use common property unit suffixes for properties with scientific units.
  See property-units.txt.

- DO define properties in terms of constraints. How many entries? What are
  possible values? What is the order?


Board/SoC .dts Files
====================

- DO put all MMIO devices under a bus node and not at the top-level.

- DO use non-empty 'ranges' to limit the size of child buses/devices. 64-bit
  platforms don't need all devices to have 64-bit address and size.
```