

Linux总线、设备、驱动模型

讲解时间：2017年7月5日晚8时
Barry Song <21cnbao@gmail.com>

观看录播：
<http://edu.csdn.net/course/detail/5329>

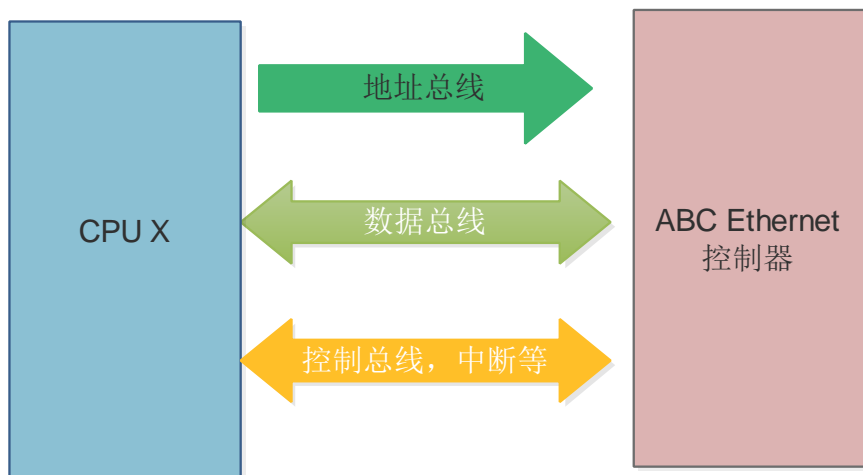
麦当劳喜欢您来，喜欢您再来



扫描关注
Linuxer



一个想象的Ethernet控制器 / 一块板



```
#define ABC_BASE 0x100000
```

```
#define ABC_IRQ 10
```

```
int abc_send(...)
```

```
{
```

```
    writel(ABC_BASE + REG_X, 1);
```

```
    writel(ABC_BASE + REG_Y, 0x3);
```

```
    ...
```

```
}
```

```
int abc_init(...)
```

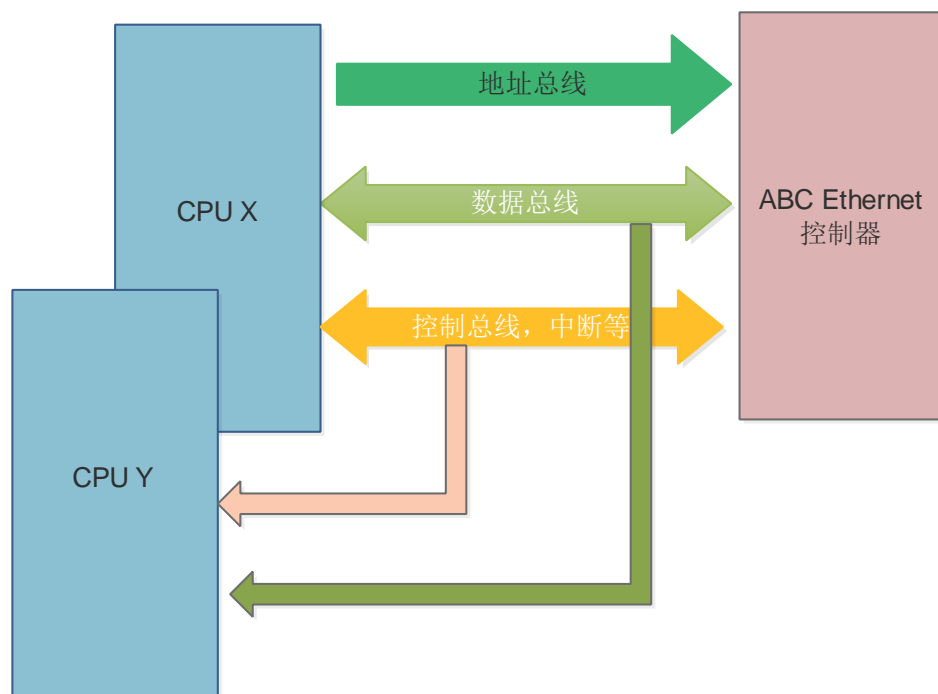
```
{
```

```
    request_irq(ABC_IRQ,...);
```

```
}
```

一个想象的Ethernet控制器/N块板

可以这样写代码吗？



```
#ifdef BOARD_A
#define ABC_BASE 0x100000
#define ABC_IRQ 10

#elif defined(BOARD_B)
#define ABC_BASE 0x110000
#define ABC_IRQ 20

#elif defined(BOARD_C)
#define ABC_BASE 0x120000
#define ABC_IRQ 10
...
#endif
```

一个想象的Ethernet控制器/1板N卡

可以这样写代码吗？

```
#ifdef BOARD_A
#define ABC1_BASE 0x100000
#define ABC1_IRQ 10
#define ABC2_BASE 0x101000
#define ABC2_IRQ 11

#elif defined(BOARD_B)
#define ABC1_BASE 0x110000
#define ABC1_IRQ 20
...
#endif
```

```
int abc1_send(...)
{
    writel(ABC1_BASE + REG_X, 1);
    writel(ABC1_BASE + REG_Y, 0x3);
    ...
}

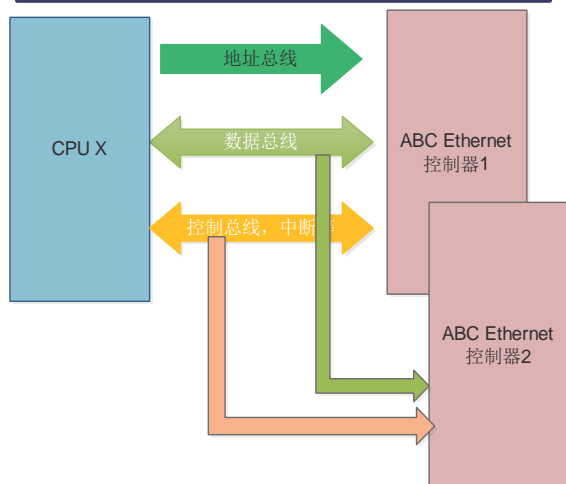
int abc1_init(...)
{
    request_irq(ABC1_IRQ,...);
}

int abc2_send(...)
{
    writel(ABC2_BASE + REG_X, 1);
    writel(ABC2_BASE + REG_Y, 0x3);
    ...
}

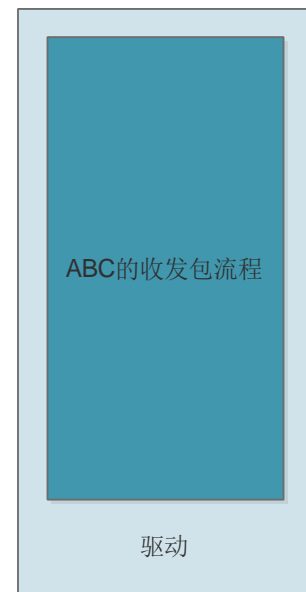
int abc2_init(...)
{
    request_irq(ABC2_IRQ,...);
}
...
```

还是这样？

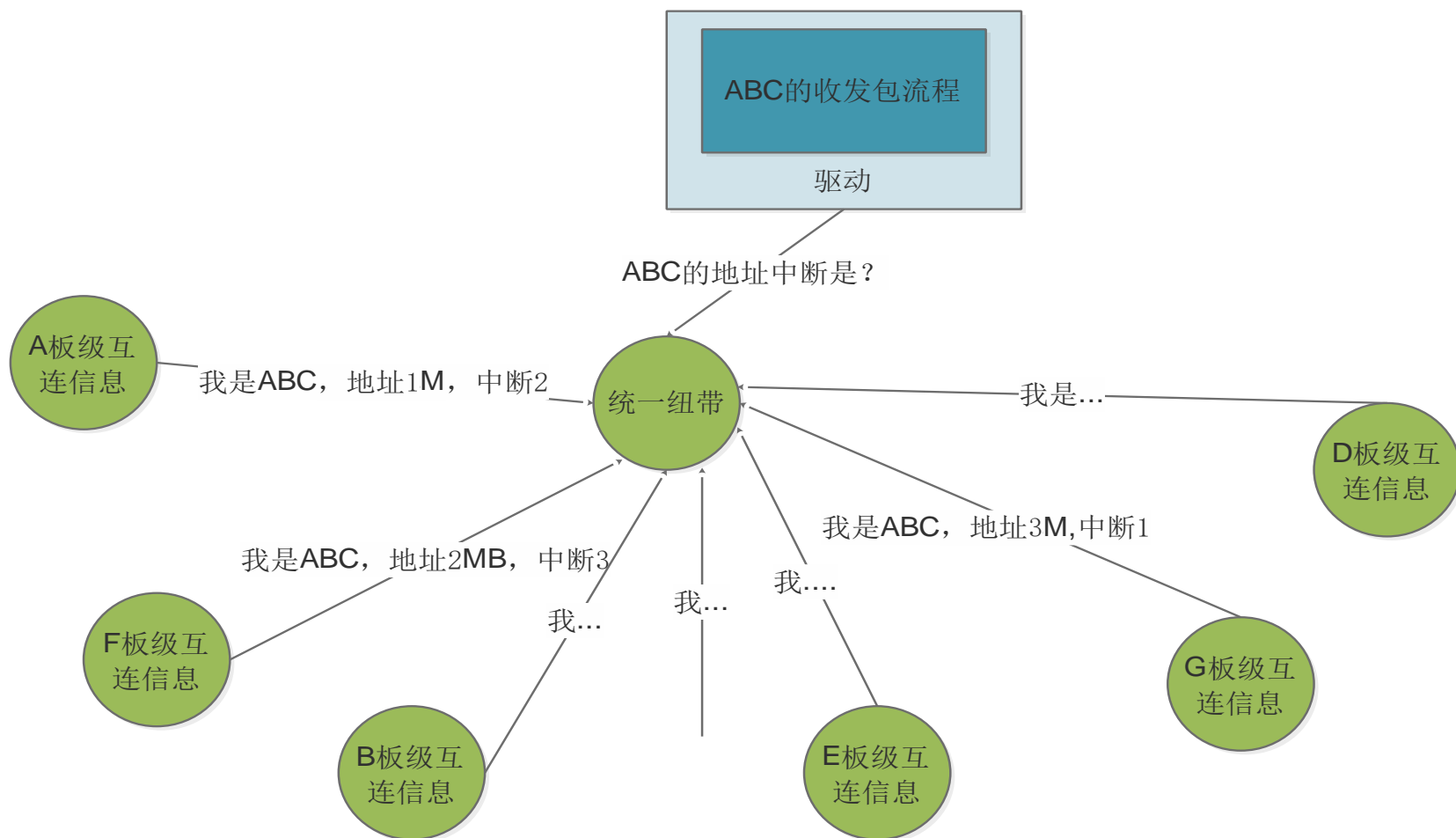
```
int abc_send(int id, ...)
{
    if (id == 0) {
        writel(ABC1_BASE + REG_X, 1);
        writel(ABC1_BASE + REG_Y, 0x3);
    } else if (id == 1) {
        writel(ABC2_BASE + REG_X, 1);
        writel(ABC2_BASE + REG_Y, 0x3);
    }
    ...
}
```



去耦合



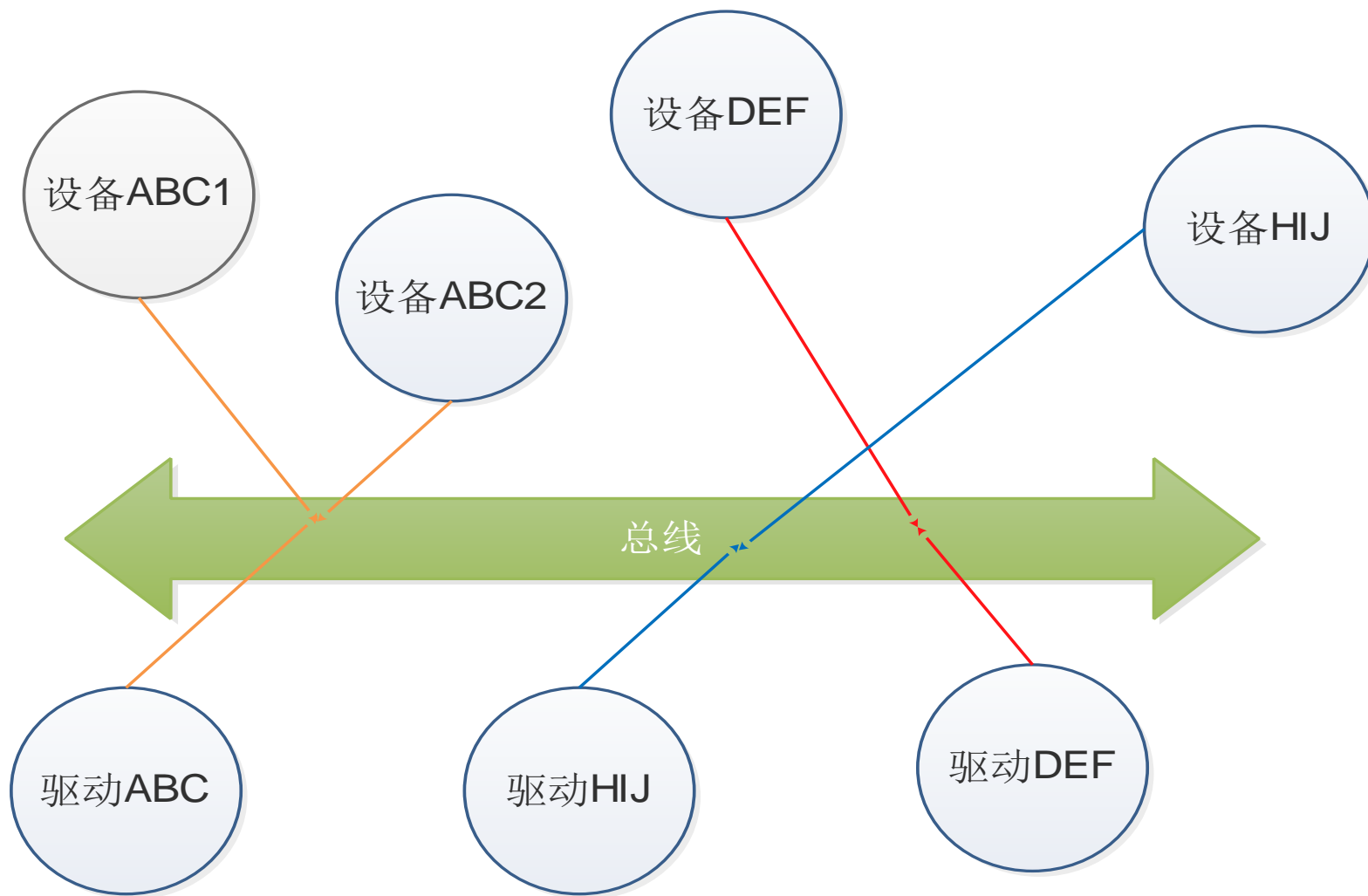
统一纽带



总线、设备、驱动

实体	功能	代码
设备	描述基地址、中断号、时钟、DMA、复位等信息	arch/arm arch/blackfin arch/xxx 等目录
驱动	完成外设的功能，如网卡收发包，声卡录放，SD卡读写...	drivers/net sound drivers/mmc 等目录
总线	完成设备和驱动的关联	drivers/base/platform.c drivers/pci/pci-driver.c ...

匹配



驱动、设备注册demo: globalfifo



设备是设备，驱动是驱动

arch/arm/mach-xxx/board-a.c - abc1设备注册

arch/arm/mach-xxx/board-a.c - abc2设备注册

arch/arm/mach-yyy/board-a.c - abc设备注册

arch/blackfin/mach-yyy/board-a.c - abc设备注册

arch/blackfin/mach-yyy/board-c.c - abc设备注册

...

总线

drivers/net/ethernet/abc.c

设备端代码: arch/xxx/

```
static struct resource dm9000_resource1[] = {
    {
        .start = 0x20100000,
        .end   = 0x20100000 + 1,
        .flags = IORESOURCE_MEM
    },
    ...
    {
        .start = IRQ_PF15,
        .end   = IRQ_PF15,
        .flags = IORESOURCE_IRQ | IORESOURCE_IRQ_HIGHEDGE
    }
};

static struct platform_device dm9000_device1 = {
    .name      = "dm9000",
    .id        = 0,
    .num_resources = ARRAY_SIZE(dm9000_resource1),
    .resource   = dm9000_resource1,
};

static struct platform_device *ip0x_devices[] __initdata = {
    &dm9000_device1,
    &dm9000_device2,
};

static int __init ip0x_init(void)
{
    platform_add_devices(ip0x_devices, ARRAY_SIZE(ip0x_devices));
}
```

驱动端代码: drivers/xxx/

```
static int dm9000_probe(struct platform_device *pdev)
{
    ...
    db->addr_res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
    db->data_res = platform_get_resource(pdev, IORESOURCE_MEM, 1);
    db->irq_res = platform_get_resource(pdev, IORESOURCE_IRQ, 0);
    ...
}

static struct platform_driver dm9000_driver = {
    .driver = {
        .name = "dm9000",
        .pm = &dm9000_drv_pm_ops,
        .of_match_table = of_match_ptr(dm9000_of_matches),
    },
    .probe = dm9000_probe,
    .remove = dm9000_drv_remove,
};
```

总线match函数

```
static int platform_match(struct device *dev, struct device_driver *drv)
{
    struct platform_device *pdev = to_platform_device(dev);
    struct platform_driver *pdrv = to_platform_driver(drv);

    /* When driver_override is set, only bind to the matching driver */
    if (pdev->driver_override)
        return !strcmp(pdev->driver_override, drv->name);

    /* Attempt an OF style match first */
    if (of_driver_match_device(dev, drv))
        return 1;

    /* Then try ACPI style match */
    if (acpi_driver_match_device(dev, drv))
        return 1;

    /* Then try to match against the id table */
    if (pdrv->id_table)
        return platform_match_id(pdrv->id_table, pdev) != NULL;

    /* fall-back to driver name match */
    return (strcmp(pdev->name, drv->name) == 0);
}
```

1个驱动多个设备

Linux习惯使用alloc

Linux习惯使用私有数据

```
static int dm9000_probe(struct platform_device *pdev)
{
    struct board_info *db;    /* Point a board information structure */
    ...
    ndev = alloc_etherdev(sizeof(struct board_info));
    if (!ndev)
        return -ENOMEM;

    ...
    /* setup board info structure */
    db = netdev_priv(ndev);
    ...
    db->addr_res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
    db->data_res = platform_get_resource(pdev, IORESOURCE_MEM, 1);
    db->irq_res = platform_get_resource(pdev, IORESOURCE_IRQ, 0);
    ...
    ret = register_netdev(ndev);
    ...
    return ret;
}

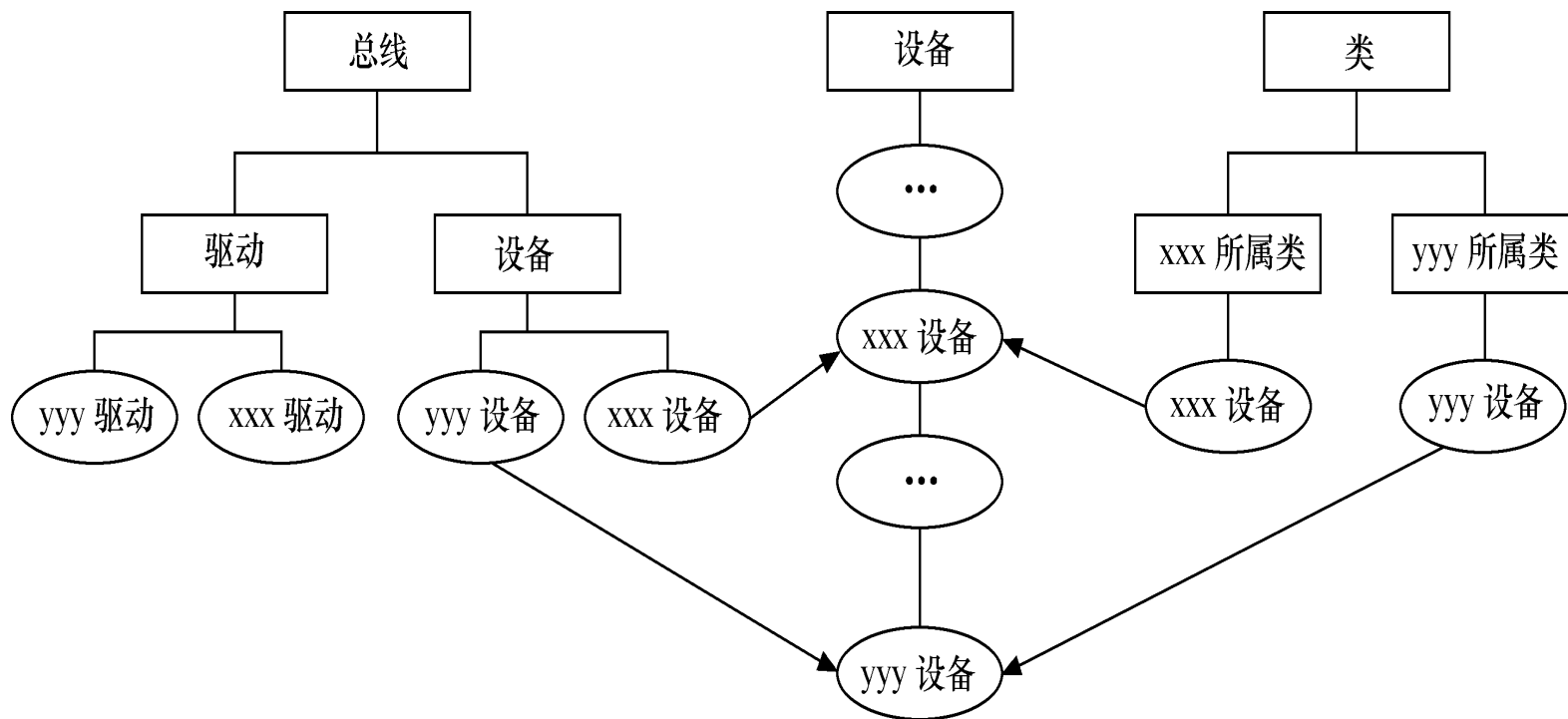
static int dm9000_start_xmit(struct sk_buff *skb, struct net_device *dev)
{
    struct board_info *db = netdev_priv(dev);
    ...
    return NETDEV_TX_OK;
}
```

/sys

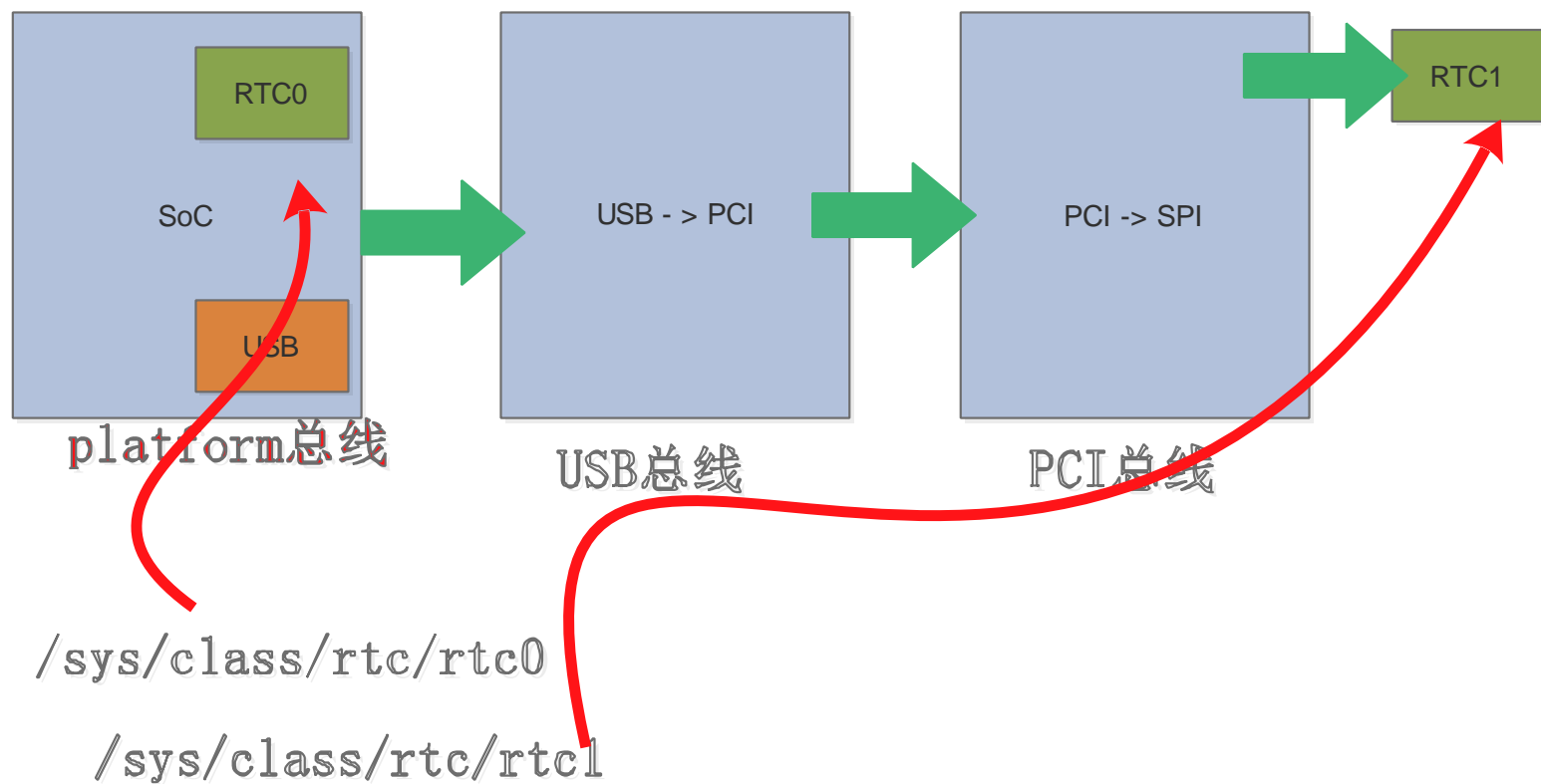
/sys/bus

/sys/devices

/sys/class



class的视角



一个LED在 /sys/devices 和 /sys/class

/sys/class/leds/v2m:green:user1 简洁直观的路径



/sys/devices/platform/smb/smb:motherboard/smb:motherboard:leds/leds/vm:green:user1

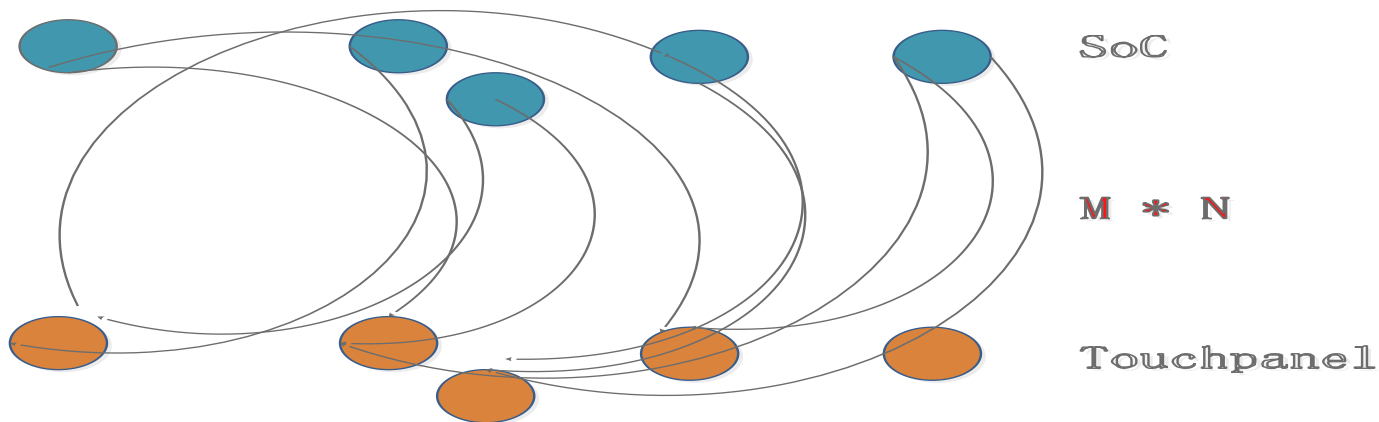
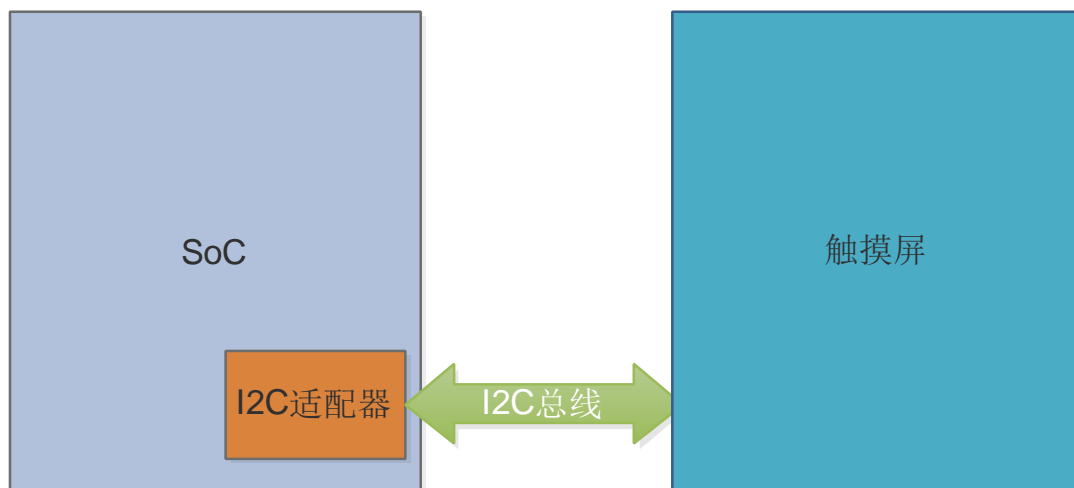
等级化的硬件互连

总线级联 - 单片机编码思维

可以这样写代码吗？

```
cpu_x_i2c_reg...  
cpu_x_i2c_reg...  
tp_y_reg...  
tp_y_reg...  
cpu_x_i2c_reg...  
cpu_x_i2c_reg...  
tp_y_reg...  
tp_y_reg...
```

这是M*N强耦合



总线级联 - 适配器单独驱动

每一级透过自己挂的总线枚举进来；
尽管它自己可能是总线适配器

可以这样写代码吗？

`cpu_x_i2c_reg...`

`cpu_x_i2c_reg...`

`tp_y_reg...`

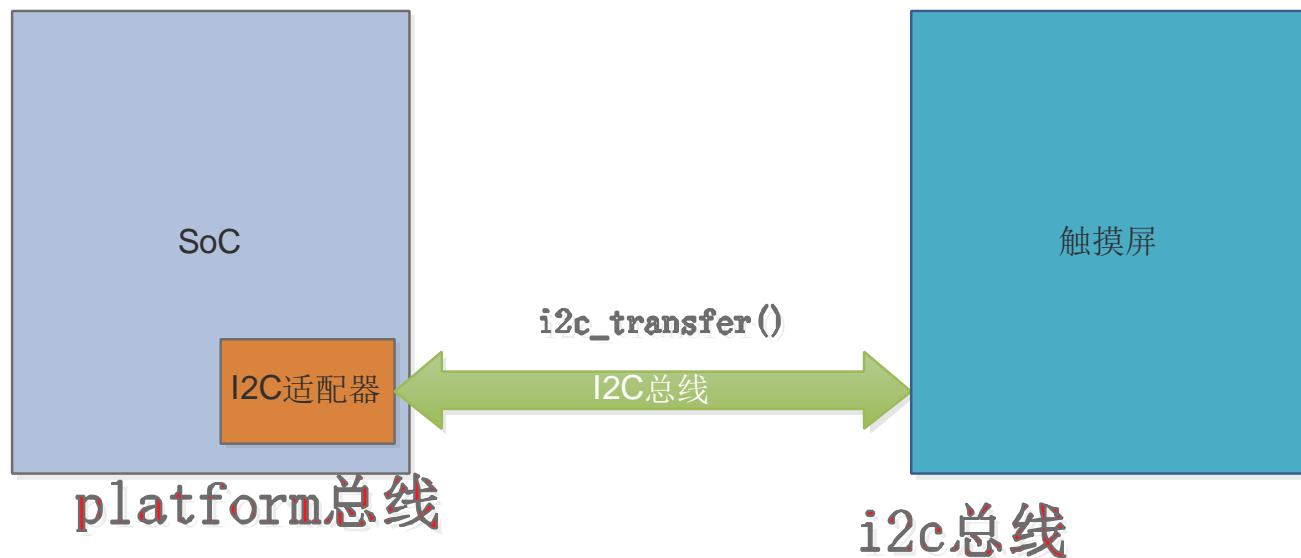
`tp_y_reg...`

`cpu_x_i2c_reg...`

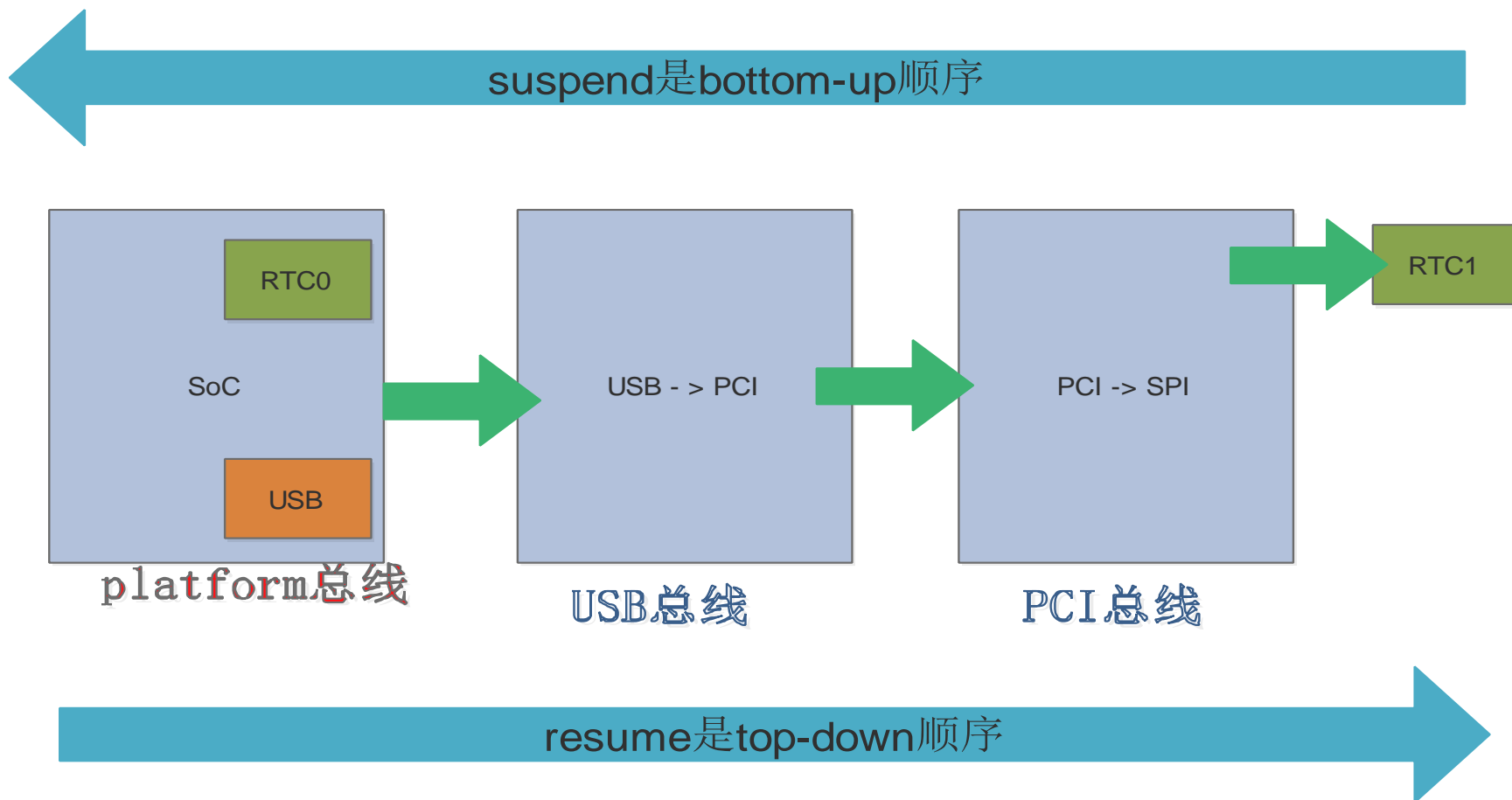
`cpu_x_i2c_reg...`

`tp_y_reg...`

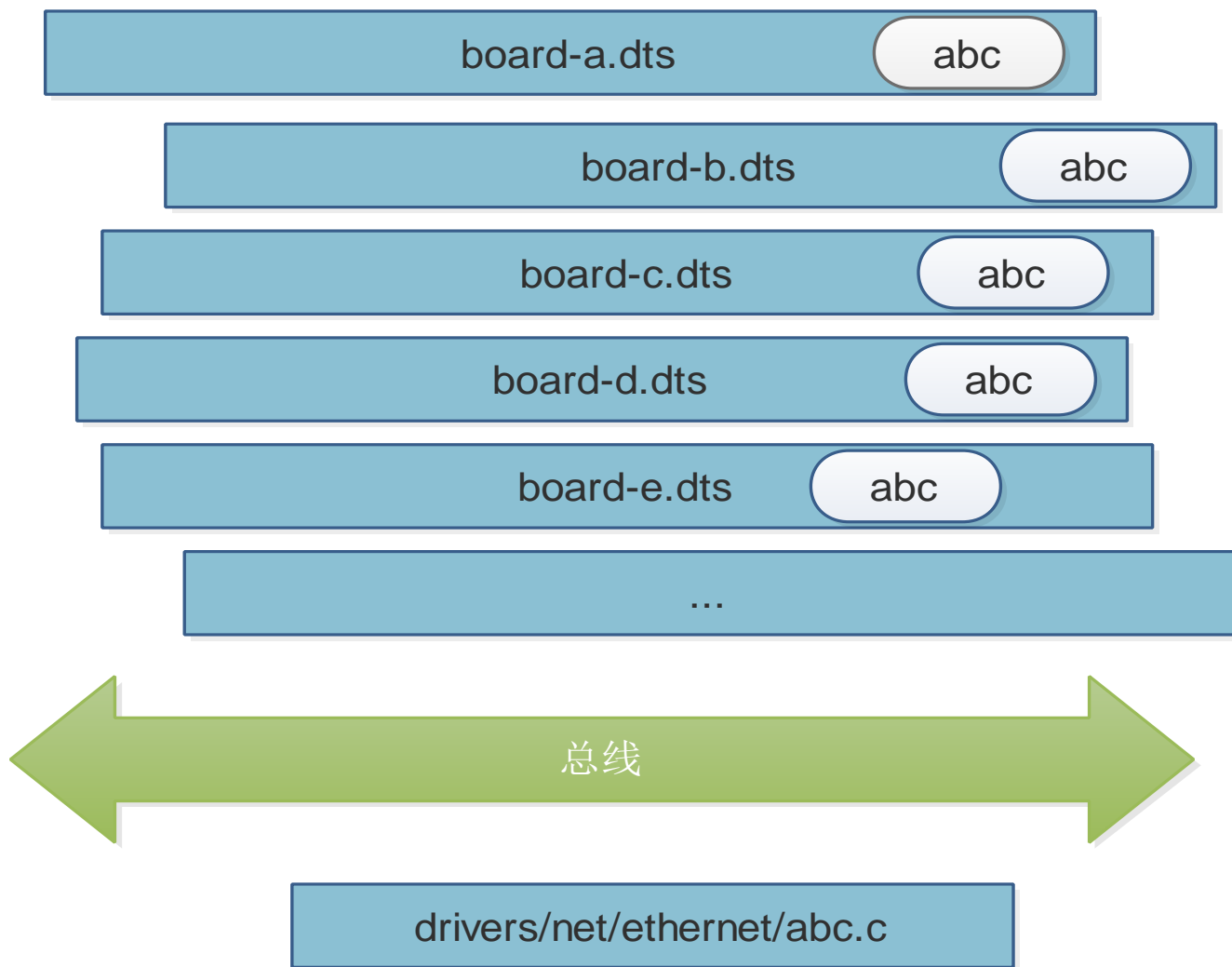
`tp_y_reg...`



总线级联与电源管理



设备在脚本，驱动在C里



DTS

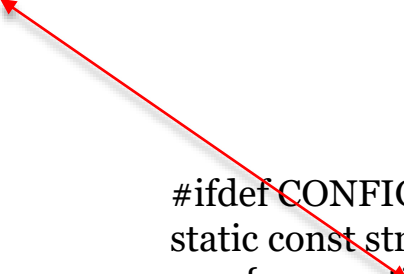
开机过程中执行的类似语句会帮忙从dts节点生成platform_device:
of_platform_populate(NULL, of_default_bus_match_table,
NULL, NULL);

```
eth: eth@4,c00000 {  
    compatible = "davicom,dm9000";  
    reg = <  
        4 0x00c00000 0x2  
        4 0x00c00002 0x2  
    >;  
    interrupt-parent = <&gpio2>;  
    interrupts = <14 IRQ_TYPE_LEVEL_LOW>;  
    ...  
};
```

dts和driver的匹配

```
eth: eth@4,c00000 {  
    compatible = "davicom,dm9000";  
    ...  
};
```

```
#ifdef CONFIG_OF  
static const struct of_device_id dm9000_of_matches[] = {  
    { .compatible = "davicom,dm9000", },  
    { /* sentinel */ }  
};  
MODULE_DEVICE_TABLE(of, dm9000_of_matches);  
#endif  
  
static struct platform_driver dm9000_driver = {  
    .driver = {  
        .name   = "dm9000",  
        .pm     = &dm9000_drv_pm_ops,  
        .of_match_table = of_match_ptr(dm9000_of_matches),  
    },  
    .probe = dm9000_probe,  
    .remove = dm9000_drv_remove,  
};
```



Of populate 展开 device

of_platform_populate函数会最终生成和展开platform_device

```
struct platform_device *of_device_alloc(struct device_node *np,
                                         const char *bus_id,
                                         struct device *parent)
{
    struct platform_device *dev;
    int rc, i, num_reg = 0, num_irq;
    struct resource *res, temp_res;

    dev = platform_device_alloc("", -1);
    if (!dev)
        return NULL;

    /* count the io and irq resources */
    while (of_address_to_resource(np, num_reg, &temp_res) == 0)
        num_reg++;
    num_irq = of_irq_count(np);

    /* Populate the resource table */
    if (num_irq || num_reg) {
        res = kzalloc(sizeof(*res) * (num_irq + num_reg), GFP_KERNEL);
        ...
        dev->num_resources = num_reg + num_irq;
        dev->resource = res;
        for (i = 0; i < num_reg; i++, res++) {
            rc = of_address_to_resource(np, i, res);
            WARN_ON(rc);
        }
        if (of_irq_to_resource_table(np, res, num_irq) != num_irq)
            ...
    }
    ...
}
```

设备驱动模型连本质都没有变!

- 2017.8.14

CSDN 《深入理解Linux的设备树》直播

讲解时间：2017年8月14日晚8时

也可扫描二维码报名

报名直播或者录播：

<http://edu.csdn.net/huiyiCourse/detail/465>



谢谢！