PCIE简单学习文档(linux3.0.8)

——何良斌

一、系统配置：

1、kernel添加对PCI总线的支持

2、涉及驱动

PCIE底层驱动：hi35xx\_dev\_host.ko pcit\_dma\_host.ko

PCIE消息驱动：mcc\_drv\_host.ko mcc\_usrdev\_host.ko

PCIV模块驱动：hi3531\_pciv.ko hi3531\_pciv\_fmw.ko

加载uboot需要的驱动：boot\_device.ko

二、内核对pci支持driver/pci

1、device driver bus，PCI子系统更具initcall确定调用顺序

probe.c pci-drvier.c ——postcore\_initcall

slot.c hipcie/pcie.c ——subsys\_initcall

proc.c ——device\_initcall

pci.c pci\_sysfs.c ——late\_initcall

初始化加载顺序：

Probe.c —— pcibus\_class\_init();

pci-drvier.c —— pci\_driver\_init()

slot.c —— pci\_slot\_init()

hipcie/pcie.c —— hisi\_pcie\_init()

proc.c —— pci\_proc\_init()

pcie/portdrv\_pci.c —— pcie\_portdrv\_init()

pci.c —— pci\_resource\_alignment\_sysfs\_init()

pci\_sysfs.c —— pci\_sysfs\_init()

hipcie/pcie.c —— hisi\_pcie\_init()

该初始化对PCIE总线进行了扫描

2、配置寄存器头

PCIe控制器有4K字节的配置寄存器空间，其中前256字节为PCI兼容的配置空间，有两种类型的PCI配置寄存器头PCIe Header Type0和PCIe Header Type1。当偏移0x0E地址段 head\_type为0时代表Header Type0：表示当前设备是一个EP设备即PCI设备；当head\_type为1：代表Header Type1：表示当前设备是一个RC设备即PCI桥or switch。

pci常用三种：pci i/o、pci memory、pci conf，三种都可以被CPU存储。设备驱动程序使用pci i/o pci mem，pci初始化使用pci conf。

三、hi35xx\_dev\_host.ko

1、probe调用流程

hi35xx\_pcie\_module\_init —> pci\_register\_driver（& pci\_hidev\_driver） —> \_\_pci\_register\_driver —> driver\_register —> bus\_add\_driver —> driver\_attach —> \_\_driver\_attach —> driver\_probe\_device —> really\_probe —> dev->bus->probe（pci\_bus\_type. pci\_device\_probe）

pci\_device\_probe—> \_\_pci\_device\_probe —> pci\_call\_probe —> local\_pci\_probe —> ddi->drv->probe（pci\_hidev\_driver. pci\_hidev\_probe）

四、drivers/pci/hipcie/pcie.c

1、函数hisi\_pcie\_plat\_driver\_probe()

pcie0\_mem\_space\_size = 0x7800000

pcie0\_cfg\_space\_size = 0x800000

pcie\_sys\_base\_virt = ioremap\_nocache(0x20050000, 0x1000);

2、函数init\_pcie\_controller\_info()

config\_base\_addr = 0x20800000;

dbi\_base\_addr = 0x40000000;

dbi\_base\_size = pcie0\_cfg\_space\_size;

pcie\_info[0].controller = 0

pcie\_info[0].conf\_base\_addr = ioremap\_nocache(config\_base\_addr, 0x1000);

pcie\_info[0].base\_addr = ioremap\_nocache(dbi\_base\_addr, dbi\_base\_size);

hipcie.nr\_controllers = 1

3、函数pcie\_sys\_init() RC模式

0x200500ac: bit[11]=0, disable pcie0

0x20030078: bit[5]=1, 打开pcie0时钟

0x200500b4: 0x05605001，参考时钟

0x200500b8: 0x2005006

0x20050090: bit[31:28]=4，RC设备

0x20030078: 复位pcie控制器

0x20080008: 0x60400<<8， pci->pcie桥设备

0x200500ac: bit[11]=1，enable pcie

0x20080004: bit[2:0]=0b111，使能

4、结构体

static struct hw\_pci hipcie \_\_initdata = {

.nr\_controllers = 1,

.preinit = pcie\_preinit,

.swizzle = pci\_std\_swizzle,

.setup = pcie\_setup,

.scan = pcie\_scan\_bus,

.map\_irq = pcie\_map\_irq,

};

5、pci\_common\_init函数

Pci总线扫描枚举，资源分配，桥过滤串口等皆由该函数实现。

preinit()、pcibios\_init\_hw()、pci\_bus\_size\_bridges()、pci\_bus\_assign\_resources()

6、函数pcie\_preinit()

pcie0\_mem\_space.start = 0x30000000

pcie0\_mem\_space.end = 0x30000000 + 0x7800000 – 1

pcie0\_io\_space.start = pcie0\_mem\_space.end + 1

pcie0\_io\_space.end = 0x30000000 + 0x10000000 -1

得到：

mem: 0x30000000 ~ 0x37800000

io: 0x37800000 ~ 0x40000000

cfg: 0x40000000 ~ 0x40800000

7、函数pcibios\_init\_hw()

struct pci\_sys\_data \*sys = NULL;

sys->hw = hw; hw = hipcie

sys->busnr = busnr;

sys->swizzle = hw->swizzle;

sys->map\_irq = hw->map\_irq;

sys->resource[0] = &ioport\_resource;

sys->resource[1] = &iomem\_resource;

ret = hw->setup(nr, sys);

这个执行之后：

sys->resource[0] = & pcie0\_io\_space;

sys->resource[1] = & pcie0\_mem\_space;

进行host ATU设置：config\_controller\_iatu\_table

总线枚举

sys->bus = hw->scan(nr, sys);

8、函数config\_controller\_iatu\_table

Viewport: 0~5 6个ATU

类型0配置事务输出转换区域

Viewport: 0

BAR: low:0x40100000 uper:0x0

LAR: 0x401fffff

TAR: low:0x01000000 uper:0x0

Region\_ctr1: 0x4 Region\_ctrl2:0x90000000

类型1配置事务输出转换区域

Viewport: 1

BAR: low:0x40200000 uper:0x0

LAR: 0x4fffffff

TAR: low:0x02000000 uper:0x0

Region\_ctr1: 0x5 Region\_ctrl2:0x90000000

IO操作输出转换区域

Viewport: 2

BAR: low:0x37800000 uper:0x0

LAR: 0x37800000 - 1

TAR: low:0x0 uper:0x0

Region\_ctr1: 0x2 Region\_ctrl2:0x80000000

9、函数pcie\_scan\_bus

pci读写操作函数：static struct pci\_ops pcie\_ops = {

.read = pcie\_read\_conf,

.write = pcie\_write\_conf,

};

中断引脚：表示设备使用了哪个中断引脚，1~3：INTA~INTD

中断号：配置时需要将中断号写入该寄存器

pci bridge设备的7、8、9项属于过滤窗口，对应总线资源。

pci bus0地址分布：

bar0 mem pref: 0x30000000~0x307fffff

bar1 mem: 0x31600000~0x3160ffff

bar6 mem pref: 0x31610000~0x3161ffff

pci bridge地址分布：

bar9 mem pref: 0x30800000~0x313fffff

bar8 mem: 0x31400000~0x315fffff

bar7 io: 0x37800000~0x37800fff

pci bus1地址分布：

bar0 mem pref: 0x30800000~0x30ffffff

bar2 mem: 0x31400000~0x314fffff

bar1 mem: 0x31500000~0x3150ffff

bar6 mem pref: 0x31000000~0x3100fffff

bar3 io: 0x37800000~0x380000ff

配置空间寄存器

Base Addres Register(BAR) 三个作用：

存放mem/io访问空间的起始地址

存放mem/io访问空间的大小

往寄存器里写0xffffffff，读出寄存器，并取反+1，就是大小

定制访问类型mem方式、IO方式

PCI设备：6个Base Address寄存器 PCI桥 2个Base Address

根总线链表头：pci\_root\_buses

一个PCI域支持256个pci总线，每个pci总线最大支持32个设备（单功能和多功能设备）。对于多功能设备最多支持8个功能。

本地配置如下: bit[31-28]: 4

bit[27-20]: BUS ID总线好

bit[19-15]： DEV ID每个总线设备号

bit[14-12]: FUN ID每个设备功能

bit[11-2]: REG\_NUM

bit[1-0]: 0

pci设备打印数字结构： 0000： 00： 00： 0

domain pcibus devid funid

五、hi35xx\_dev\_host.c

1、函数pci\_hidev\_probe

hi\_dev->bar0 = pdev->resorce[0].start = 0x30800000

hi\_dev->bar1 = pdev->resorce[1].start = 0x31500000

hi\_dev->bar2 = pdev->resorce[2].start = 0x31400000

hi\_dev->device\_id = 0x353219e5

hi\_dev->pci\_bar0\_virt = (unsigned int)ioremap\_nocache(hi\_dev->bar0, 0x800000);

hi\_dev->pci\_bar1\_virt = (unsigned int)ioremap\_nocache(hi\_dev->bar1, 0x1000);

pdev->dev->p->driver\_data = hi\_dev

2、函数g\_local\_handler->init\_hidev

配置总线山的设备配置空间