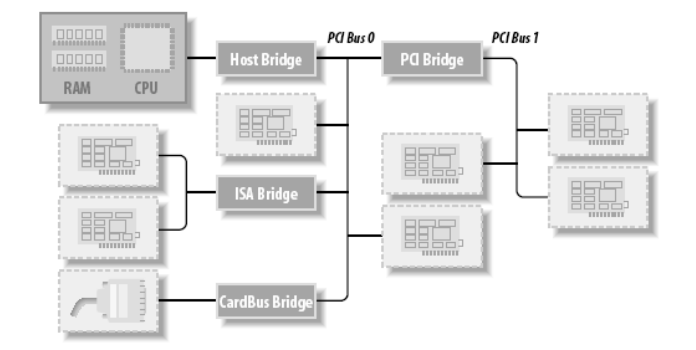
PCI（外围设备互连标准）被设计为ISA的替代品

包括我们现在使用的PC的显卡，网卡，声卡都属于PCI设备

1. **PCI寻址**



PCI外设由 域、总线编号、设备编号、功能编号 标识

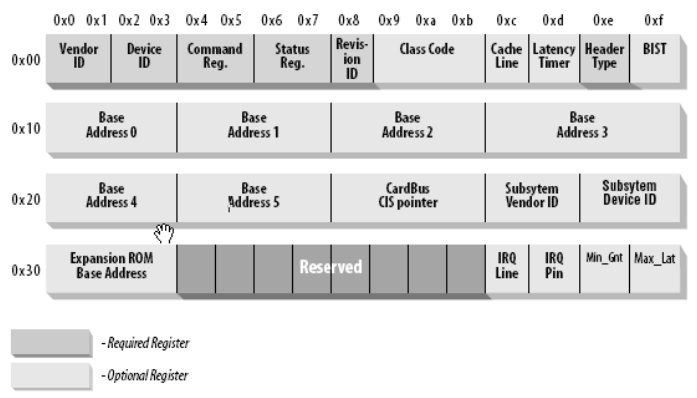
使用lspci可查看PCI外设标识



1. **PCI 配置寄存器**

PCI的配置在系统引导阶段被映射到cpu地址空间上

PCI的配置至少256个字节，其中前64个是标准化的，如下



其中verdorID, deviceID, 和 class三个寄存器可标识一个设备

vendorID

制造商ID. 例如, 每个 Intel 设备都标有相同的供应商号, 0x8086.

deviceID

设备ID，由供应商自定义

class

每个设备都属于某个类，由供应商自定义

1. **驱动支持的设备列表**

一个struct pci\_device\_id 结构代表驱动程序所支持的PCI设备，使用下面的宏初始化pci\_device\_id 结构

PCI\_DEVICE(vendor, device)

这个创建一个 struct pci\_device\_id , 它只匹配特定的供应商和设备 ID. PCI\_DEVICE\_CLASS(device\_class, device\_class\_mask)

这个创建一个 struct pci\_device\_id, 它匹配一个特定的 PCI 类.

使用数组表示驱动程序所支持的设备列表

static struct pci\_device\_id i810\_ids[] = {

    {PCI\_DEVICE(PCI\_VENDOR\_ID\_INTEL, PCI\_DEVICE\_ID\_INTEL\_82810\_IG1)},

    {PCI\_DEVICE(PCI\_VENDOR\_ID\_INTEL, PCI\_DEVICE\_ID\_INTEL\_82810\_IG3)},

    {PCI\_DEVICE(PCI\_VENDOR\_ID\_INTEL, PCI\_DEVICE\_ID\_INTEL\_82810E\_IG)},

    {PCI\_DEVICE(PCI\_VENDOR\_ID\_INTEL, PCI\_DEVICE\_ID\_INTEL\_82815\_CGC)},

    {PCI\_DEVICE(PCI\_VENDOR\_ID\_INTEL, PCI\_DEVICE\_ID\_INTEL\_82845G\_IG)},

    { 0, },

};

**导出驱动支持的设备列表**

对应热插拔设备，当我们插入拔出插入拔出时，系统就会装载卸载装载卸载我们的驱动程序，那系统如何知道插入哪些设备就装载我们的驱动呢？使用MODULEDEVICETABLE 宏，例：

MODULE\_DEVICE\_TABLE(pci, i810\_ids);

Pci：我们支持的设备类型

I810\_ids：我们前面定义的驱动程序所支持的设备列表

1. **注册驱动程序**

**struct pci\_driver 结构**

要注册驱动程序需要实现struct pci\_driver 结构，该结构字段如下

const char \*name;

驱动程序的名字. 在内核中所有 PCI 驱动里面它必须是唯一的

const struct pci\_device\_id \*id\_table;

指向 struct pci\_device\_id 表的指针（所支持的设备列表）

int (\*probe) (struct pci\_dev \*dev, const struct pci\_device\_id \*id);

指向 PCI 驱动探测函数的指针. 当正确初始化这个设备并且返回 0. 如果这个驱动不想拥有这个设备, 或者产生一个错误, 它应当返回一个负的错误值.

void (\*remove) (struct pci\_dev \*dev);

指向在 struct pci\_dev 从系统中移除或PCI 驱动从内核中卸载时调用的函数的指针

int (\*suspend) (struct pci\_dev \*dev, u32 state);

当 struct pci\_dev 被挂起时调用的函数的指针，这个函数是可选的

int (\*resume) (struct pci\_dev \*dev);

当 pci\_dev 被恢复时调用的函数的指针这个函数时可选的

**注册PCI驱动程序**

pci\_register\_driver(&pci\_driver);

注册成功返回0，否则返回负值

**卸载PCI驱动程序**

pci\_unregister\_driver(&pci\_driver);

**激活PCI设备**

在访问任何设备资源之前，需要先激活设备

int pci\_enable\_device(struct pci\_dev \*dev);

1. **访问配置空间**

在探测到由PCI设备后，驱动程序便可以向IO内存或IO端口读写数据了，但IO内存或IO端口的位置在哪？这就需要访问配置空间获取了

读配置空间函数

int pci\_read\_config\_byte(struct pci\_dev \*dev, int where, u8 \*val);

int pci\_read\_config\_word(struct pci\_dev \*dev, int where, u16 \*val);

int pci\_read\_config\_dword(struct pci\_dev \*dev, int where, u32 \*val);

写配置空间函数

int pci\_write\_config\_byte(struct pci\_dev \*dev, int where, u8 val);

int pci\_write\_config\_word(struct pci\_dev \*dev, int where, u16 val);

int pci\_write\_config\_dword(struct pci\_dev \*dev, int where, u32 val);

1. **访问IO内存或IO端口**

PCI设备可以有至多6个IO地址区域（IO内存或端口），这6个区域的配置寄存器为Base Address 0 ~ 5中，我们使用如下函数来获取这些区域的地址

unsigned long pci\_resource\_start(struct pci\_dev \*dev, int bar);

// bar指定区域号（0-5），该函数返回IO端口号或IO内存的开始地址

unsigned long pci\_resource\_end(struct pci\_dev \*dev, int bar);

// bar指定区域号（0-5），该函数返回IO内存的结束地址

unsigned long pci\_resource\_flags(struct pci\_dev \*dev, int bar);

// 这个函数返回和这个区域相关的标识.

// 所有的资源标志都定义在 <linux/ioport.h>; 最重要的是:

// IORESOURCE\_IO

// IORESOURCE\_MEM

1. **PCI中断**

要安装中断需要获取中断号，中断号保存在配置寄存器的PCI\_INTERRUPT\_LINE中，通过如下函数获取中断号

result = pci\_read\_config\_byte(dev, PCI\_INTERRUPT\_LINE, &myirq);

if (result)

{

    /\* deal with error \*/

}