PCI 设备详解一

2016-10-09

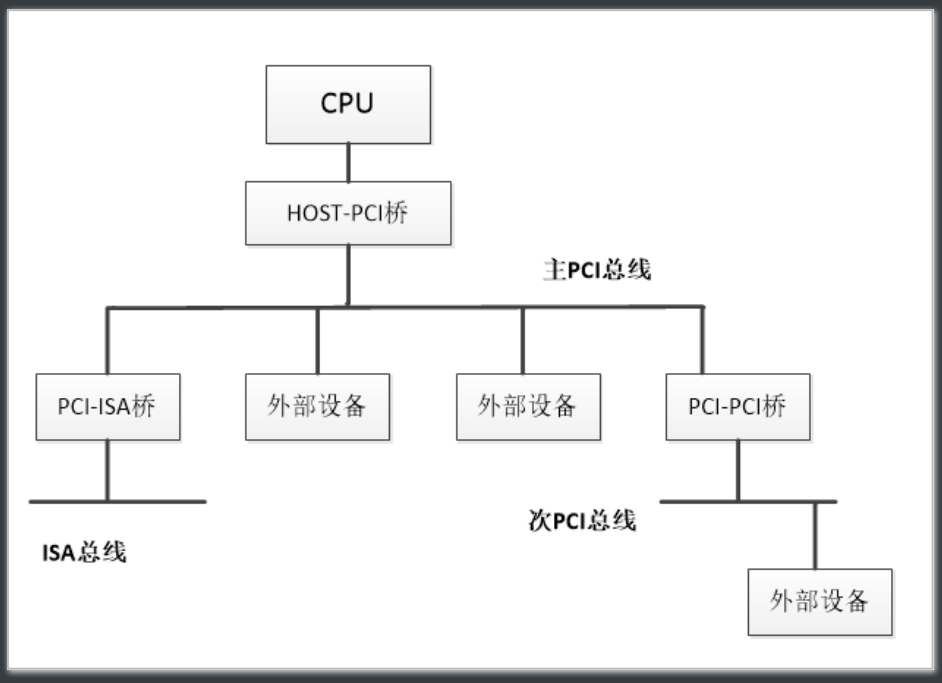
其实之前是简单学习过 PCI 设备的相关知识，但是总感觉自己的理解很函数，很多东西说不清楚，正好今天接着写这篇文章自己重新梳理一下，文章想要分为三部分，首先介绍 PCI 设备硬件相关的知识，然后介绍 LINux 内核中对 PCI 设备的支持。本节讲第一部分。

PCI 总线在目前计算机总线系统中占据举足轻重的地位，其良好的扩展性，地址统一分配和总线竞争的处理相对于其他总线而言都具有绝对优势。

扩展性：

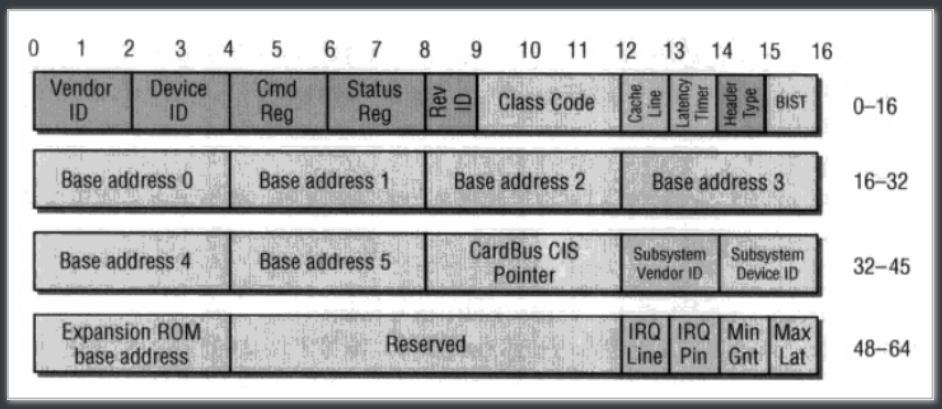
先说其扩展性，PCI 总线上存在若干 PCI 设备插槽，当 PCI 插槽无法满足需求，就可以通过 PCI 桥扩展 PCI 设备，一个 PCI 桥把一个 PCI 总线连在一个 PCI 插槽上，作为 PCI 的一个设备。例如 CPU 通过 “宿主 - PCI 桥与一条 PCI 总线相连，此总线成为“主 PCI 总线”，当通过 PCI 桥扩展 PCI 总线时，扩展的总线成为“从总线”，当然还可以通过其他的桥比如“PCI-ISA” 桥扩展 ISA 总线，所以这样通过 PCI-PCI 桥可以构筑起一个层次的、树状的 PCI 系统结构，对于上层的总线而言，连接在这条总线上的 PCI 桥也是一个设备，但是这是一种特殊的设备。

其 PCI 树状结构如图所示：



一条 PCI 总线一般有 32 个接口，即可以连接 32 个 PCI 接口卡，而一个接口卡对应一个外部设备，注意这里的外部设备可以有多个功能（最多八个），每一个功能称为逻辑设备。每一个逻辑设备对应一个 PCI 配置空间。对于逻辑设备后面还会详细解释，这里先说配置空间的问题。PCI 配置空间可以说是记录了关于此设备的详细信息。PCI 配置空间最大 256 个字节，其中起先的 64 个字节的格式是预定义好的。当然并非所有的项都必须填充，位置是固定了，没有用到可以填充 0。而前 16 个字节的格式是一定的。包含头部的类型、设备的总类、设备的性质以及制造商等。

PCi 配置空间前 64 个字节格式如下：



需要注意的有一下几项：

ClassCode 用于将设备分到具体的功能组，该字段分为两部分，前 8 个 bit 表示基类即大类别，后 8 个比特表示基类的一个子类。比如 PCI_BASE_CLASS_STORAGE 表示大类大容量存储器，而 PCI_CLASS_STORAGE_IDE 表明这个 IDE 控制器。

HeaderType 表明头部类型。一般区分为 0 型头部（PCI 设备）1 型头部（PCI 桥），注意不同头部的配置空间格式有差异。这里我们主要先描述 0 型头部即普通 PCi 设备的配置空间。

PCI HeaderType 为一个字节的大小，最高位为 0 表示单功能，最高位为 1 表示多功能（即前面描述的逻辑设备），单功能情况下一个 PCI 设备就是一个逻辑设备。低 7 位表示头部类型。

前 16 个字节都是一些基本的信息就不在多说，重点看下

接下来的 6 个 BAR 空间。每个 BAR 记录了该设备映射的一段地址空间。为了区分 IO 空间和 IO 内存，这里我们分开描述：

当 BAR 最后一位为 0 表示这是映射的 IO 内存，为 1 是表示这是 IO 端口，当是 IO 内存的时候 1-2 位表示内存的类型，bit 2 为 1 表示采用 64 位地址，为 0 表示采用 32 位地址。bit1 为 1 表示区间大小超过 1M，为 0 表示不超过 1M.bit3 表示是否支持可预取。

而相对于 IO 内存，当最后一位为 1 时表示映射的 IO 地址空间。IO 地址空间一般不支持预取，所以这里是 29 位的地址。

一般情况下，6 个 BAR 是足够使用的，大部分情况都是 3-4 个 BAR。而这些 BAR 映射的空间是连续的，即这些 BAR 共同描述设备的地址空间范围

除了 6 个基本的 BAR 空间们还有一个额外的配置 ROM 区间，区间最低位表示是否使用 ROM 区间，高 21 位表示地

址。中间的是保留项。其余原理和上面类似。

接下来需要注意的就是中断，因为大部分外设和系统交互就是通过中断的方式。。

由配置空间中的 IRQ Pin 决定设备是否支持中断，1 表示支持，0 表示不支持。假如支持中断，IRQ Line 表记录下中断号。

PCI 桥的配置空间

PCI 桥同样是连接在 PCI 总线接口卡上的一个设备，只不过是一个桥设备，连接一条 PCI 总线。既然同属于设备，那么它同样也就有设备的配置空间，只是它的配置空间和普通设备的有些差异。PCI 桥的配置空间在系统软件遍历 PCI 总线树的时候配置，并不需要专门的 PCI 驱动，故称为透明桥。PCI 桥连接两条总线，和 HOst 桥近的称为上游总线（Primary Bus ）, 远的一条称为下游总线（Secondry Bus）。PCI 桥的配置空间在前 16 个字节的格式和普通 PCI 设备并无区别，另外，桥还保留了普通设备的前两个 BAR 空间。所以从配置空间的 0x18 开始有了桥设备自身的配置格式。如前所述，桥设备记录了上游总线和下游总线，以及桥下最大的总线号。这里还有一个比较重要的概念就是窗口。在 PCI 桥的配置空间有三个窗口：IO 地址区间窗口、存储器区间窗口、可预取存储器地址窗口。实际上每个窗口都是一段地址区间，就像一个门，规定了桥下设备映射的区间。从北桥出来的地址，如果在该区间内，就可以穿过该桥到达次级总线，这样依次寻找设备。反过来，从南桥出来的地址及由设备发出的，只有地址不在该区间范围内才可以穿过该桥。因为同一条总线上的设备交互不需要外部空间，就像是内网传输和公网传输一样的道

理。

内核中关于桥配置空间的定义如下：

```
/* Header type 1 (PCI-to-PCI bridges) */
#define PCI_PRIMARY_BUS        0x18    /* Primary bus number
#define PCI_SECONDARY_BUS    0x19    /* Secondary bus number
#define PCI_SUBORDINATE_BUS    0x1a    /* Highest bus number
#define PCI_SEC_LATENCY_TIMER    0x1b    /* Latency timer for

#define PCI_IO_BASE        0x1c    /* I/O range behind the br
#define PCI_IO_LIMIT        0x1d

#define  PCI_IO_RANGE_TYPE_MASK    0x0fUL    /* I/O bridging
#define  PCI_IO_RANGE_TYPE_16    0x00
#define  PCI_IO_RANGE_TYPE_32    0x01
#define  PCI_IO_RANGE_MASK    (~0x0fUL) /* Standard 4K I/O wi
#define  PCI_IO_1K_RANGE_MASK    (~0x03UL) /* Intel 1K I/O wi
#define PCI_SEC_STATUS        0x1e    /* Secondary status reg

#define PCI_MEMORY_BASE        0x20    /* Memory range behind
#define PCI_MEMORY_LIMIT    0x22

#define  PCI_MEMORY_RANGE_TYPE_MASK 0x0fUL
#define  PCI_MEMORY_RANGE_MASK    (~0x0fUL)

#define PCI_PREF_MEMORY_BASE    0x24    /* Prefetchable memor
#define PCI_PREF_MEMORY_LIMIT    0x26

#define  PCI_PREF_RANGE_TYPE_MASK 0x0fUL
#define  PCI_PREF_RANGE_TYPE_32    0x00
#define  PCI_PREF_RANGE_TYPE_64    0x01
#define  PCI_PREF_RANGE_MASK    (~0x0fUL)
#define PCI_PREF_BASE_UPPER32    0x28    /* Upper half of pre
#define PCI_PREF_LIMIT_UPPER32    0x2c
#define PCI_IO_BASE_UPPER16    0x30    /* Upper half of I/O a
#define PCI_IO_LIMIT_UPPER16    0x32
/* 0x34 same as for htype 0 */
/* 0x35-0x3b is reserved */
#define PCI_ROM_ADDRESS1    0x38    /* Same as PCI_ROM_ADDRES
```

```
/* 0x3c-0x3d are same as for htype 0 */

#define PCI_BRIDGE_CONTROL    0x3e
#define  PCI_BRIDGE_CTL_PARITY    0x01    /* Enable parity de
#define  PCI_BRIDGE_CTL_SERR    0x02    /* The same for SERR
#define  PCI_BRIDGE_CTL_ISA    0x04    /* Enable ISA mode */
#define  PCI_BRIDGE_CTL_VGA    0x08    /* Forward VGA address
#define  PCI_BRIDGE_CTL_MASTER_ABORT    0x20  /* Report maste
#define  PCI_BRIDGE_CTL_BUS_RESET    0x40    /* Secondary bus
#define  PCI_BRIDGE_CTL_FAST_BACK    0x80    /* Fast Back2Bac
```

下面说说 PCI 设备的地址空间：

前面也简单介绍了下 PCI 设备的地址空间支持 PIO 和 MMIO，即 IO 端口和 IO 内存。下面详细分析下这两种方式：

PIO

IO 端口的编址是独立于系统的地址空间，其实就是一段地址区域，所有外设的地址都映射到这段区域中。就像是一个进程内部的各个变量，公用进程地址空间一样。不同外设的 IO 端口不同。访问 IO 端口需要特殊的 IO 指令，OUT/IN，OUT 用于 write 操作，in 用于 read 操作。在此基础上，操作系统实现了读写不同大小端口的函数。为什么说是不同大小呢？？因为前面也说到，IO 端口实际上是一段连续的区域，每个端口理论上是字节为单位即 8bit, 那么要想读写 16 位的端口只能把相邻的端口进行合并，32 位的端口也是如此。

例如下面的汇编指令：

OUT 21h,al

0x21 是 8259A 中断控制器的中断屏蔽寄存器，该指令将 Intel X86 处理器的 al 寄存器中的值写到中断寄存器的控制寄存器中，从而达到屏蔽某些中断信号的目的。类似的，在下面的汇编指令中：

IN al,20h

0x20 是 8259A 中断控制器的正在服务寄存器，该指令将此寄存器中的状态值传递到处理器的 al 寄存器中。

无论是 windows 还是 Linux 都会上述指令做了封装以满足读写不同长度端口的需要。不过这种方式缺点也比较明显，一般情况 CPU 分配给 IO 端口的空间都比较小，在当前外设存储日益增大的情况下很难满足需要。另一方面，

MMIO

IO 内存是直接把寄存器的地址空间直接映射到系统地址空间，系统地址空间往往会保留一段内存区用于这种 MMIO 的映射（当然肯定是位于系统内存区），这样系统可以直接使用普通的访存指令直接访问设备的寄存器，随着计算机内存容量的日益增大，这种方式更是显出独特的优势，在性能至上的理念下，使用 MMIO 可以最大限度满足日益增长的系统和外设存储的需要。所以当前其实大多数外设都是采用 MMIO 的方式。

还有一种方式是把 IO 端口空间映射到内存空间，这样依然可以通过正常的访存指令访问 IO 端口，但是这种方式下依然受到 IO 空间大小的制约，可以说并没有解决实际问题。

但是上述方案只适用于在外设和内存进行小数据量的传输时，假如进行大数据量的传输，那么 IO 端口这种以字节为单位的传输就不用说了，IO 内存虽然进行了内存映射，但是其映射的范围大小相对于大量的数据，仍然不值一提，所以即使采用 IO 内存仍然是满足不了需要，会让 CPU 大部分时间处理繁琐的映射，极大的浪费了 CPU 资源。那么这种情况就引入了 DMA，直接内存访问。这种方式的传输由 DMA 控制器控制，CPU 给 DMA 控制器下达传输指令后就转而处理其他的事务，然后 DMA 控制器就开始进行数据的传输，在完成数据的传输后通过中断的方式通知 CPU，这样就可以极大的解放 CPU。当然本次讨论的重点不在 DMA，所以对于 DMA 的讨论仅限于此。

那么 CPU 是怎么访问这些配置寄存器的呢？要知道配置寄存器指定了设备存储寄存器的映射方式以及地址区间，但是配置寄存器本身的访问就是一个问题。为每个设备预留 IO 端口或者 IO 内存都是不现实的。暂且不说 x86 架构下 IO 端口地址空间只有区区 64K, 就是内存，虽然现在随着科技的发展，内存空间越来越大，但是也不可能为每个设备预留空间。那么折中的方式就是为所有设备的配置寄存器使用同一个 IO 端口，系统在 IO 地址空间预留了一段地址就是 0xCF8~0xCFF 一共八个字节，前四个字节做地址端口，后四个字节做数据端口。CPU 访问某个设备的配置寄存器时，先向地址端口写入地址，然后从数据端口读写数据。这里的地址是一个综合地址，结构如下：

最高位为1	保留不用（7位）	总线号（8位）	设备号（5位）	功能号（3位）	寄存器地址（8位）最低两位为0

这里就可以解释我们总线数量和逻辑设备数量的限制了。在寻找某一个逻辑设备时，先根据总线号找到总线，然后根据设备号找到总线上的某个接口，最后在根据功能号定位某一个逻辑设备。

到此，PCI 总线架构以及设备的配置空间已经描述的差不多了，下面就是看系统如何探测和初始化设备了。

下篇文章会结合 LInux 源代码分析下 Linux 内核中对 PCI 设备的配置过程

全文完