kernel hacker 修炼之道之 PCI subsystem(三)

作者 李万鹏

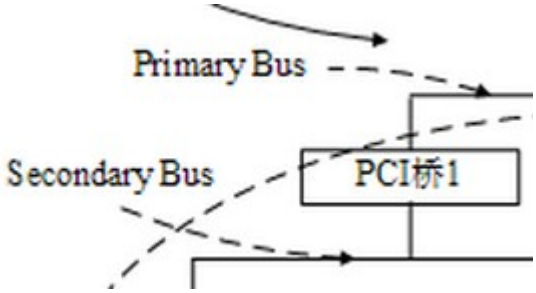
这里主要分析 PCI core 核心的数据结构：

pci_bus，这个是 PCI core 中用来描述 pci 总线的结构体：

```
1. 413struct pci_bus {
2. 414        struct list_head node;          /* node in list of buses */
3. 415        struct pci_bus  *parent;        /* parent bus this bridge is on */
4. 416        struct list_head children;      /* list of child buses */
5. 417        struct list_head devices;       /* list of devices on this bus */
6. 418        struct pci_dev  *self;          /* bridge device as seen by parent */
7. 419        struct list_head slots;         /* list of slots on this bus */
8. 420        struct resource *resource[PCI_BRIDGE_RESOURCE_NUM];
9. 421        struct list_head resources;     /* address space routed to this bus */
10.422
11.423        struct pci_ops  *ops;           /* configuration access functions */
12.424        void            *sysdata;       /* hook for sys-specific extension */
13.425        struct proc_dir_entry *procdir; /* directory entry in /proc/bus/pci */
14.426
15.427        unsigned char   number;         /* bus number */
16.428        unsigned char   primary;        /* number of primary bridge */
17.429        unsigned char   secondary;      /* number of secondary bridge */
18.430        unsigned char   subordinate;    /* max number of subordinate buses */
19.431        unsigned char   max_bus_speed; /* enum pci_bus_speed */
20.432        unsigned char   cur_bus_speed;  /* enum pci_bus_speed */
21.433
22.434        char            name[48];
23.435
24.436        unsigned short  bridge_ctl;     /* manage NO_ISA/FBB/et al behaviors */
25.437        pci_bus_flags_t bus_flags;      /* Inherited by child busses */
26.438        struct device           *bridge;
27.439        struct device           dev;
28.440        struct bin_attribute    *legacy_io; /* legacy I/O for this bus */
29.441        struct bin_attribute    *legacy_mem; /* legacy mem */
30.442        unsigned int            is_added:1;
31.443};
```

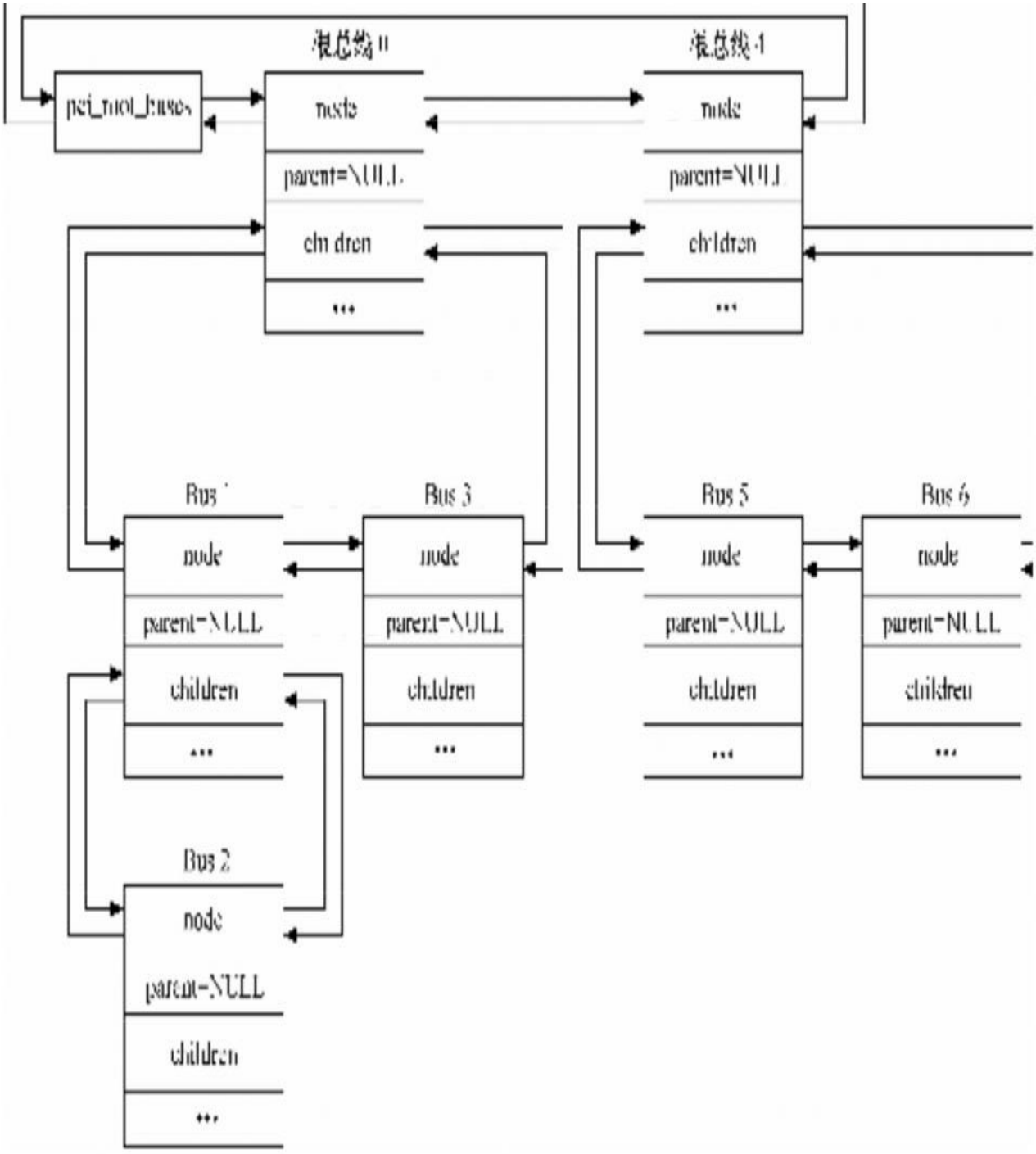
这里的 node bit field 是一个链表，如果是根总线，就是 host-to-pci bridge 引出的那条总线的话，bus 通过 node bit field 链入 pci_root_buses 链表，pci_root_buses 链表头定义于 drivers/pci/probe.c 文件中。如果是普通 bus 那么它通过 node bit field 链入 parent bus 的 children 链表。这样内核中所有 pci_bus 就形成了一个倒置的树状结构。 parrent bit field 指明了这条 bus 的 parent bus，也就是引出这条 bus 的 bridge 所在的 bus。devices bit field 又是一个链表头，这条 bus 上的所有 device 都链接到这条链表上，注意这里的 device 是 logical device，也就是 domain:bus:device:function 中的 function，这里的 device 会有一个专门的

结构 struct slot 来表示。 self bit field 指向了引出这条 bus 的 bridge。number bit field 是 bus 的号，primary 实际是这个 bus 坐在的桥的上游总线和下游总线，如下图：



primary 是引出这条 bus 所在的 bridge 所在的 bus，secondary 是这条 bus 所在的 bridge 引出的 bus，所以应该和 number bit field 相等。subordinate bit field 是这条 bus 下的 pci sub tree 中 bus number 最大的，这个在分配 bus 号的时候非常有用的。

下面是 pci_bus 的 tree：



pci_device，这个是 PCI core 中用来描述 pci 总线的结构体：

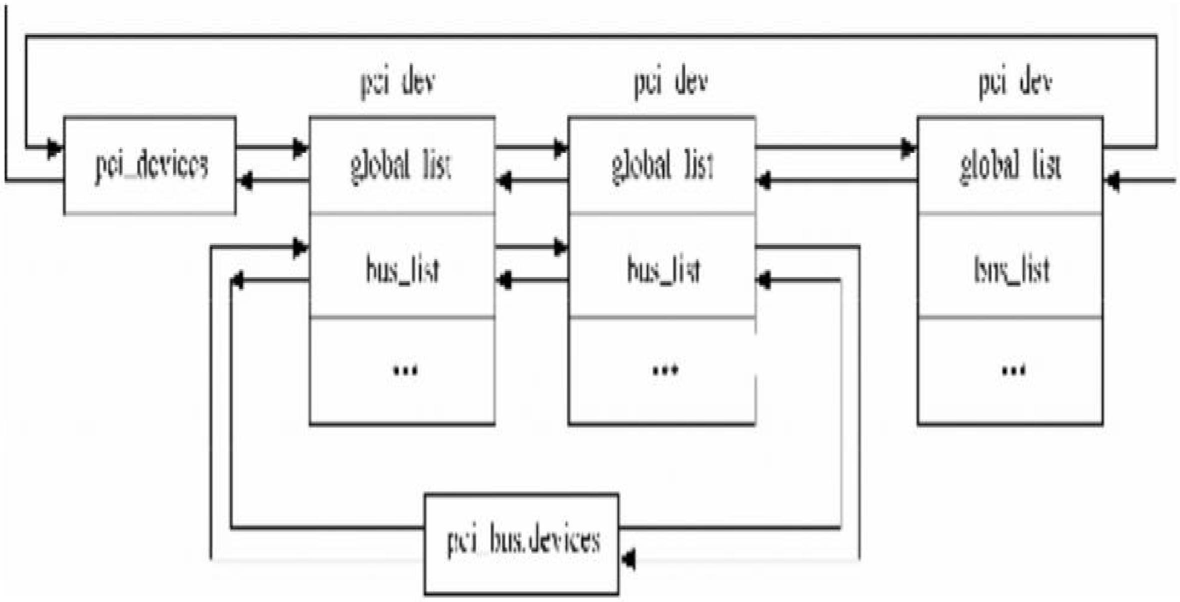
```
238struct pci_dev {
239        struct list_head bus_list;      /* node in per-bus list */
240        struct pci_bus  *bus;           /* bus this device is on */
241        struct pci_bus  *subordinate;   /* bus this device bridges to */
242
243        void            *sysdata;       /* hook for sys-specific extension */
244        struct proc_dir_entry *procent; /* device entry in /proc/bus/pci */
245        struct pci_slot *slot;          /* Physical slot this device is in */
246
```

```
10.247      unsigned int    devfn;          /* encoded device & function index */
11.248      unsigned short  vendor;
12.249      unsigned short  device;
13.250      unsigned short  subsystem_vendor;
14.251      unsigned short  subsystem_device;
15.252      unsigned int    class;          /* 3 bytes: (base,sub,prog-if) */
16.253      u8              revision;       /* PCI revision, low byte of class word */
17.254      u8              hdr_type;       /* PCI header type (`multi' flag masked out) */
18.255      u8              pcie_cap;       /* PCI-E capability offset */
19.256      u8              pcie_type:4;    /* PCI-E device/port type */
20.257      u8              pcie_mpss:3;    /* PCI-E Max Payload Size Supported */
21.258      u8              rom_base_reg;   /* which config register controls the ROM */
22.259      u8              pin;            /* which interrupt pin this device uses */
23.260
24.261      struct pci_driver *driver;      /* which driver has allocated this device */
25.262      u64             dma_mask;       /* Mask of the bits of bus address this
26.263                                     device implements.  Normally this is
27.264                                     0xffffffff.  You only need to change
28.265                                     this if your device has broken DMA
29.266                                     or supports 64-bit transfers.  */
30.267
31.268      struct device_dma_parameters dma_parms;
32.269
33.270      pci_power_t     current_state;  /* Current operating state. In ACPI-speak,
34.271                                     this is D0-D3, D0 being fully functional,
35.272                                     and D3 being off. */
36.273      int             pm_cap;         /* PM capability offset in the
37.274                                     configuration space */
38.275      unsigned int    pme_support:5;  /* Bitmask of states from which PME#
39.276                                     can be generated */
40.277      unsigned int    pme_interrupt:1;
41.278      unsigned int    pme_poll:1;     /* Poll device's PME status bit */
42.279      unsigned int    d1_support:1;   /* Low power state D1 is supported */
43.280      unsigned int    d2_support:1;   /* Low power state D2 is supported */
44.281      unsigned int    no_d1d2:1;      /* Only allow D0 and D3 */
45.282      unsigned int    mmio_always_on:1;       /* disallow turning off io/mem
46.283                                     decoding during bar sizing */
47.284      unsigned int    wakeup_prepared:1;
48.285      unsigned int    d3_delay;       /* D3->D0 transition time in ms */
49.286
50.287#ifdef CONFIG_PCIEASPM
51.288      struct pcie_link_state  *link_state;    /* ASPM link state. */
52.289#endif
53.290
54.291      pci_channel_state_t error_state;        /* current connectivity state */
55.292      struct  device  dev;            /* Generic device interface */
56.293
57.294      int             cfg_size;       /* Size of configuration space */
58.295
59.296      /*
60.297       * Instead of touching interrupt line and base address registers
61.298       * directly, use the values stored here. They might be different!
62.299       */
```

```
        unsigned int    irq;
        struct resource resource[DEVICE_COUNT_RESOURCE]; /* I/O and memory regions + expansion ROMs */
        resource_size_t fw_addr[DEVICE_COUNT_RESOURCE]; /* FW-assigned addr */

        /* These fields are used by common fixups */
        unsigned int    transparent:1;  /* Transparent PCI bridge */
        unsigned int    multifunction:1;/* Part of multi-function device */
        /* keep track of device state */
        unsigned int    is_added:1;
。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。
#ifdef CONFIG_PCI_ATS
        union {
                struct pci_sriov *sriov;        /* SR-IOV capability related */
                struct pci_dev *physfn; /* the PF this VF is associated with */
        };
        struct pci_ats  *ats;   /* Address Translation Service */
#endif
};
```

首先说明 pci bridge 与真正的 pci device 都是 pci device。通过 bus_list bit field 可以将 pci_dev 链入 pci_dev 所在的 pci_bus 的 devices 链表上。



bus bit field 则直接指向了 pci_dev 所在的 bus。subordinate bit field 是当 pci_dev 是一个 bridge 的时候，这个 bridge 所引出的 bus。slot bit field 是这个 pci_dev 所在的 slot。devfn bit field 是将 5 位 device 号和 3 位 function 合到一起的逻辑功能号。vendor，
device，subsystem_vendor，subsystem_device，class，revision，hdr_type 等都是可以从 PCI device 的 configuration space 读出来的。multifunction:1 bit field 表明这是一个单功能设备还是多功能设备，如果是多功能设备则 bit field 为 1。struct resource resource[DEVICE_COUNT_RESOURCE]是 pci_dev 所拥有

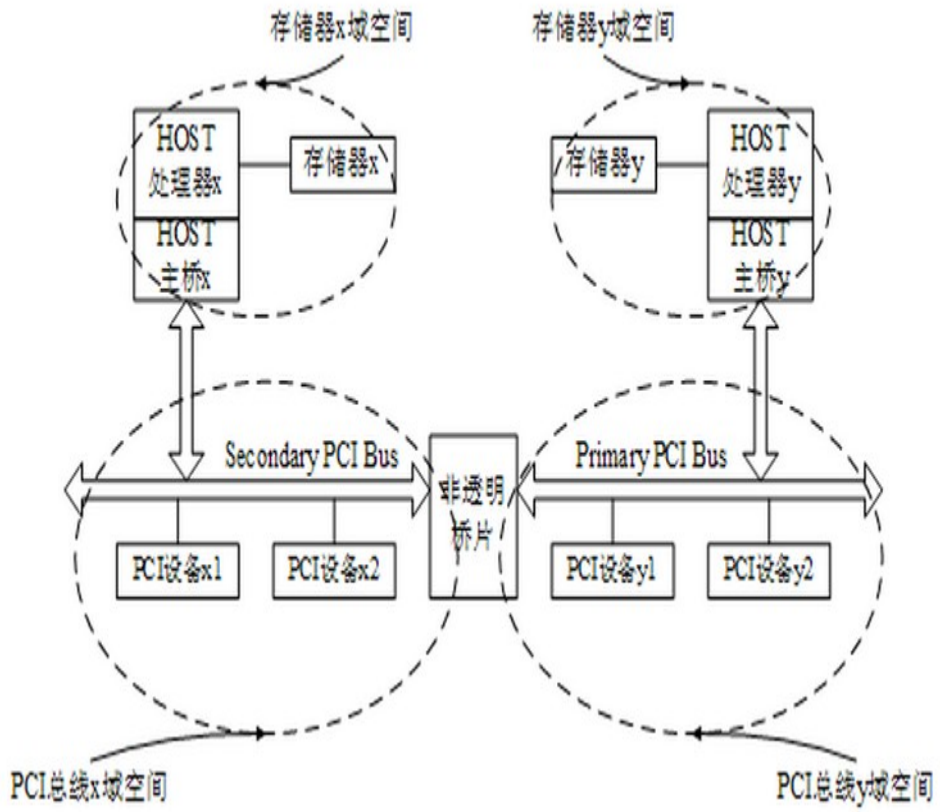
的资源的数组。在 PCI subsystem 中还有一棵 resource 树滴。在 include/linux/pci.h 中定义了一个枚举：

```
1. 89enum {
2. 90        /* #0-5: standard PCI resources */
3. 91        PCI_STD_RESOURCES,
4. 92        PCI_STD_RESOURCE_END = 5,
5. 93
6. 94        /* #6: expansion ROM resource */
7. 95        PCI_ROM_RESOURCE,
8. 96
9. 97        /* device specific resources */
10. 98#ifdef CONFIG_PCI_IOV
11. 99        PCI_IOV_RESOURCES,
12.100        PCI_IOV_RESOURCE_END = PCI_IOV_RESOURCES + PCI_SRIOV_NUM_BARS - 1,
13.101#endif
14.102
15.103        /* resources assigned to buses behind the bridge */
16.104#define PCI_BRIDGE_RESOURCE_NUM 4
17.105
18.106        PCI_BRIDGE_RESOURCES,
19.107        PCI_BRIDGE_RESOURCE_END = PCI_BRIDGE_RESOURCES +
20.108                                  PCI_BRIDGE_RESOURCE_NUM - 1,
21.109
22.110        /* total resources associated with a PCI device */
23.111        PCI_NUM_RESOURCES,
24.112
25.113        /* preserve this for compatibility */
26.114        DEVICE_COUNT_RESOURCE
27.115};
```

0～5 表示 pci device 的 6 个 resource，可以包括：I/O，MMIO，prefetch MMIO，6 是 ROM。所以一个普通的 pci device 会有这 4 类资源。如果是 pci bridge 还有 4 个过滤窗口，其实也就是这个 pci bridge 下的 pci sub tree 所拥有的全部资源。当 CPU 要访问 pci device 的时候，会与 pci bridge 上的 4 个过滤窗口进行匹配，当资源的范围在这个地址区间内的话就会将这个 CPU 的 request 向下 forward。过滤窗口是枚举里的：7 是 I/O，8 是 MMIO，9 是 prefetch，10 是 ROM。

如果 pci_dev 是一个 pci bridge，那么如果 transparent:1 bit field 为 1 则这个 bridge 是透明桥，否则是非透明桥。

一般的 PCI bridge 被称为透明桥，主要原因是 PCI bridge 的配置空间在系统软件遍历 PCI 总线树时配置，系统软件不需要专门的驱动程序设置 PCI bridge 的使用方法。在某些处理器系统中，还有一类桥 PCI bridge，叫做非透明桥。非透明桥不是 PCI 总线定义的标准桥片，但是在使用 PCI 总线挂接另外一个处理器时非常有用，非透明桥片主要是连接连个不同的 PCI 总线域，进而连接两个处理器系统。



pci_driver，这个是 PCI core 中用来描述 pci 设备驱动的结构体：

```
1. 551struct pci_driver {
2. 552        struct list_head node;
3. 553        const char *name;
4. 554        const struct pci_device_id *id_table;   /* must be non-NULL for probe to be called */
5. 555        int  (*probe)  (struct pci_dev *dev, const struct pci_device_id *id);   /* New device inserted
   */
6. 556        void (*remove) (struct pci_dev *dev);   /* Device removed (NULL if not a hot-plug capable
   driver) */
7. 557        int  (*suspend) (struct pci_dev *dev, pm_message_t state);      /* Device suspended */
8. 558        int  (*suspend_late) (struct pci_dev *dev, pm_message_t state);
9. 559        int  (*resume_early) (struct pci_dev *dev);
10.560        int  (*resume) (struct pci_dev *dev);                   /* Device woken up */
11.561        void (*shutdown) (struct pci_dev *dev);
12.562        struct pci_error_handlers *err_handler;
13.563        struct device_driver    driver;
14.564        struct pci_dynids dynids;
15.565};
```

pci_slot，这个是 PCI core 中用来描述 pci slot 的结构体：

```
1. 61struct pci_slot {
2. 62        struct pci_bus *bus;           /* The bus this slot is on */
3. 63        struct list_head list;         /* node in list of slots on this bus */
4. 64        struct hotplug_slot *hotplug;   /* Hotplug info (migrate over time) */
5. 65        unsigned char number;          /* PCI_SLOT(pci_dev->devfn) */
6. 66        struct kobject kobj;
```

```
7. 67};
```

一个 slot 可以有 8 个 function，其中每个 function 在内核中会有一个 pci_slot 对应。

还有一个与体系结构相关的 pci_controller，这个结构用来描述 host bridge，在 arch/powerpc/include/asm/pci-bridge.h 中定义：

```
1. 20struct pci_controller {
2. 21        struct pci_bus *bus;
3. 22        char is_dynamic;
4. 23#ifdef CONFIG_PPC64
5. 24        int node;
6. 25#endif
7. 26        struct device_node *dn;
8. 27        struct list_head list_node;
9. 28        struct device *parent;
10.29
11.30        int first_busno;
12.31        int last_busno;
13.32        int self_busno;
14.33
15.34        void __iomem *io_base_virt;
16.35#ifdef CONFIG_PPC64
17.36        void *io_base_alloc;
18.37#endif
19.38        resource_size_t io_base_phys;
20.39        resource_size_t pci_io_size;
21.40
22.41        /* Some machines (PReP) have a non 1:1 mapping of
23.42         * the PCI memory space in the CPU bus space
24.43         */
25.44        resource_size_t pci_mem_offset;
26.45
27.46        /* Some machines have a special region to forward the ISA
28.47         * "memory" cycles such as VGA memory regions. Left to 0
29.48         * if unsupported
30.49         */
31.50        resource_size_t isa_mem_phys;
32.51        resource_size_t isa_mem_size;
33.52
34.53        struct pci_ops *ops;
35.54        unsigned int __iomem *cfg_addr;
36.55        void __iomem *cfg_data;
37.56
38.57        /*
39.58         * Used for variants of PCI indirect handling and possible quirks:
40.59         *  SET_CFG_TYPE - used on 4xx or any PHB that does explicit type0/1
41.60         *  EXT_REG - provides access to PCI-e extended registers
42.61         *  SURPRESS_PRIMARY_BUS - we suppress the setting of PCI_PRIMARY_BUS
43.62         *   on Freescale PCI-e controllers since they used the PCI_PRIMARY_BUS
44.63         *   to determine which bus number to match on when generating type0
45.64         *   config cycles
```

```
46.65         *  NO_PCIE_LINK - the Freescale PCI-e controllers have issues with
47.66         *   hanging if we don't have link and try to do config cycles to
48.67         *   anything but the PHB.  Only allow talking to the PHB if this is
49.68         *   set.
50.69         *  BIG_ENDIAN - cfg_addr is a big endian register
51.70         *  BROKEN_MRM - the 440EPx/GRx chips have an errata that causes hangs on
52.71         *   the PLB4.  Effectively disable MRM commands by setting this.
53.72         */
54.73#define PPC_INDIRECT_TYPE_SET_CFG_TYPE          0x00000001
55.74#define PPC_INDIRECT_TYPE_EXT_REG               0x00000002
56.75#define PPC_INDIRECT_TYPE_SURPRESS_PRIMARY_BUS  0x00000004
57.76#define PPC_INDIRECT_TYPE_NO_PCIE_LINK          0x00000008
58.77#define PPC_INDIRECT_TYPE_BIG_ENDIAN            0x00000010
59.78#define PPC_INDIRECT_TYPE_BROKEN_MRM            0x00000020
60.79        u32 indirect_type;
61.80        /* Currently, we limit ourselves to 1 IO range and 3 mem
62.81         * ranges since the common pci_bus structure can't handle more
63.82         */
64.83        struct resource io_resource;
65.84        struct resource mem_resources[3];
66.85        int global_number;              /* PCI domain number */
67.86
68.87        resource_size_t dma_window_base_cur;
69.88        resource_size_t dma_window_size;
70.89
71.90#ifdef CONFIG_PPC64
72.91        unsigned long buid;
73.92
74.93        void *private_data;
75.94#endif  /* CONFIG_PPC64 */
76.95};
```

这里的 first_busno bit field 表示 phb 直接链接的总线号，last_busno bit field 是这个 PCI domain 中最大的总线号。