Linux Device Drivers – PCI Drivers

Jernej Vičič

Overview

1. Introduction
2. PCI Interface
   - PCI addressing
3. Booting
4. PCI driver registration
5. Other buses

Introduction

- bus,
- The most common is the PCI (in the PC world),
- PCI - Peripheral Component Interconnect,
- bus consists of two components:
    - electrical interface
    - programming interface,
- other buses.

PCI Interface

- PCI is more than just a bunch of wires,
- it is a complete set of specifications,
- defines how different parts of the computer should work together,
- differs from other simpler buses: *autodetection*:
  - PCI devices do not have a jumper (older buses and devices require it),
  - devices are configured during boot time,
  - driver must read the configuration information on the device itself.

PCI addressing

- each PCI device is represented by:
  - bus number,
  - device number,
  - function number.

PCI addressing

- PCI specification allows up to 256 buses on one system,
- Linux combines buses into domains,
- most of today's systems have at least two PCI buses,
- they are connected with a bridge.

PCI addressing

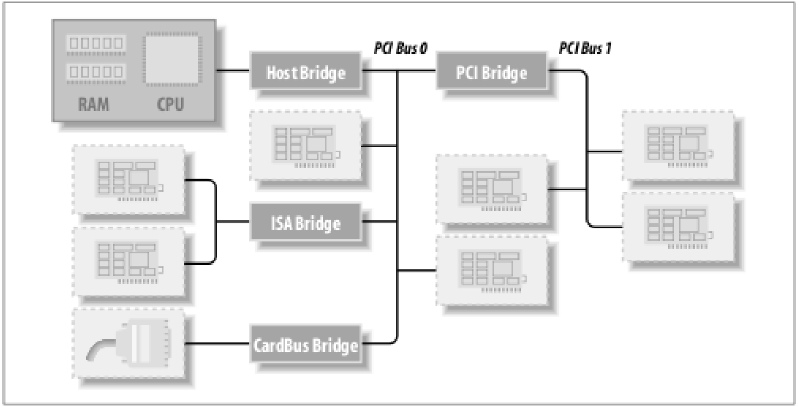


Figure: A typical PCI system.

PCI addressing

```
$ lspci | cut -d: -f1-3
0000:00:00.0 Host bridge
0000:00:00.1 RAM memory
0000:00:00.2 RAM memory
0000:00:02.0 USB Controller
0000:00:04.0 Multimedia audio controller
0000:00:06.0 Bridge
0000:00:07.0 ISA bridge
0000:00:09.0 USB Controller
0000:00:09.1 USB Controller
0000:00:09.2 USB Controller
0000:00:0c.0 CardBus bridge
0000:00:0f.0 IDE interface
0000:00:10.0 Ethernet controller
0000:00:12.0 Network controller
0000:00:13.0 FireWire (IEEE 1394)
0000:00:14.0 VGA compatible controller
```

PCI addressing

```
$ cat /proc/bus/pci/devices | cut -f1
0000
0001
0002
0010
0020
0030
0038
0048
0049
004a
0060
0078
0080
0090
0098
00a0
```

PCI addressing

```
$ tree /sys/bus/pci/devices/
/sys/bus/pci/devices/
|-- 0000:00:00.0 -> ../../../devices/pci0000:00/0000:00:00.0
|-- 0000:00:00.1 -> ../../../devices/pci0000:00/0000:00:00.1
|-- 0000:00:00.2 -> ../../../devices/pci0000:00/0000:00:00.2
|-- 0000:00:02.0 -> ../../../devices/pci0000:00/0000:00:02.0
|-- 0000:00:04.0 -> ../../../devices/pci0000:00/0000:00:04.0
|-- 0000:00:06.0 -> ../../../devices/pci0000:00/0000:00:06.0
|-- 0000:00:07.0 -> ../../../devices/pci0000:00/0000:00:07.0
|-- 0000:00:09.0 -> ../../../devices/pci0000:00/0000:00:09.0
|-- 0000:00:09.1 -> ../../../devices/pci0000:00/0000:00:09.1
|-- 0000:00:09.2 -> ../../../devices/pci0000:00/0000:00:09.2
|-- 0000:00:0c.0 -> ../../../devices/pci0000:00/0000:00:0c.0
|-- 0000:00:0f.0 -> ../../../devices/pci0000:00/0000:00:0f.0
|-- 0000:00:10.0 -> ../../../devices/pci0000:00/0000:00:10.0
|-- 0000:00:12.0 -> ../../../devices/pci0000:00/0000:00:12.0
|-- 0000:00:13.0 -> ../../../devices/pci0000:00/0000:00:13.0
‘-- 0000:00:14.0 -> ../../../devices/pci0000:00/0000:00:14.0
```

PCI addressing

- The example explained:
  - VGA video,
  - 0x00a0 = 0000:00:14.0,
  - domain (16 bits),
  - bus (8 bits).
  - device (5 bits),
  - function (3 bits),

PCI addressing

- The hardware of each peripheral device responds to queries:
  - memory locations,
  - I/O ports,
  - configuration registers
- the first two are shared among all devices,
- configuration registers use *geographical addressing*,
- the configuration queries address a single slot, never collide.

PCI addressing

- access to memory and I/O regions is well known,
- *inb*, *readb*,
- configuration transactions are performed by calling specific kernel functions for accessing configuration registers,
- Each PCI slot has four interrupting pins,
- device is defined by an n-touple:
  - domain number,
  - bus number,
  - device number,
  - function number.

Booting

- boot time,
- at that time all devices will be configured,
- when the device gets electricity, it remains inactive,
- each motherboard is equipped with firmware,
- this equipment prepares the configuration part of the devices,
- reads and writes to device registers,

Booting

- at the boot time configuration transactions for each device are triggered,
- to do the firmware or the Linux kernel (depending on the configuration),
- I/O regions and device memory are already mapped into the memory of the processor when the device driver accesses the device.

Booting

- the user can view the list of PCI devices:
    - /proc/bus/pci/devices (text file, device information),
    - /proc/bus/pci/*/* (binary file, configuration registers for each device, one file per device),
    - each device has a directory in sysfs: */sys/bus/pci/devices*.

Booting

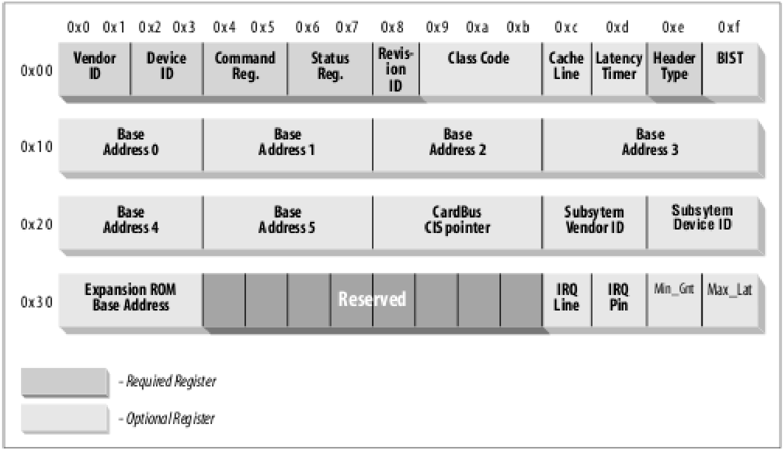
- PCI device directory:

```
$ tree /sys/bus/pci/devices/0000:00:10.0
/sys/bus/pci/devices/0000:00:10.0
|-- class
|-- config
|-- detach_state
|-- device
|-- irq
|-- power
|  `-- state
|-- resource
|-- subsystem_device
|-- subsystem_vendor
`-- vendor
```

Booting

- *config* - binary file configuration information,
- *vendor, device, subsystem _device, subsystem _vendor, class* - values for a particular device,
- *irq* - the current irq,
- *resource* - the current memory resources of this device.

Booting PCI



	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
0x00	Vendor ID		Device ID		Command Reg.		Status Reg.		Revision ID	Class Code			Cache Line	Latency Timer	Header Type	BIST
0x10	Base Address 0				Base Address 1				Base Address 2				Base Address 3			
0x20	Base Address 4				Base Address 5				CardBus CIS pointer				Subsytem Vendor ID		Subsytem Device ID	
0x30	Expansion ROM Base Address				Reserved								IRQ Line	IRQ Pin	Min_Gnt	Max_Lat

▒ - Required Register

░ - Optional Register

Figure: PCI configuration registers

As the figure shows, some
of the PCI configuration registers are required and some are optional

Booting

- some registers are mandatory,
- other optional,
- PCI registers are always little-endian,

PCI driver registration

- main structure: *struct pci_driver*,
- consists of:
  - a set of callback functions,
  - a set of variables describing the driver.

PCI driver registration

-
- const char *name; – driver name, unique for all PCI drivers,
- const struct pci_device_id *id_table; – pointer to the *pci_device_id* structure.
- int (*probe) (struct pci_dev *dev, const struct pci_device_id *id); – pointer to a probing function,
- void (*remove) (struct pci_dev *dev); – pointer to a function called by PCI kernel when *pci_dev* is being removed from the system,
- int (*suspend) (struct pci_dev *dev, u32 state); – pointer to a function called by PCI kernel at suspend
- int (*resume) (struct pci_dev *dev); – pointer to a function called by PCI kernel at resume.

PCI driver registration

- this is the least needed:

```
static struct pci_driver pci_driver = {
  .name = "pci_skel",
  .id_table = ids,
  .probe = probe,
  .remove = remove,
};
```

PCI driver registration

- registration of the structure *pci_driver* in PCI kernel:

```
static int __init pci_skel_init(void)
{
  return pci_register_driver(&pci_driver);
}
```

Other buses

- *Industry Standard Architecture (ISA)* – old in design and is a notoriously poor performer,
- *Micro Channel Architecture (MCA)* – IBM v PS/2 računalnikih,
- *Extended ISA (EISA)* – 32 bitno ISA vodilo,
- *VESA Local Bus (VLB)* – Mac computers,
- *SBus* – SPARC-based workstations,
- *NuBus* – Mac computers M68k.
- *USB* – zunanje vodilo.