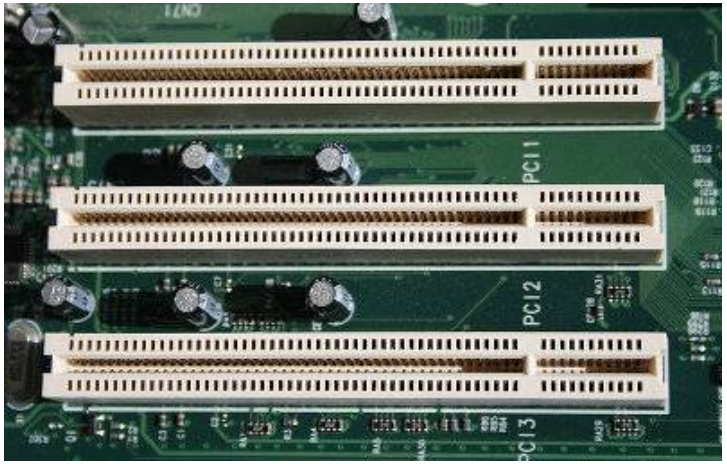
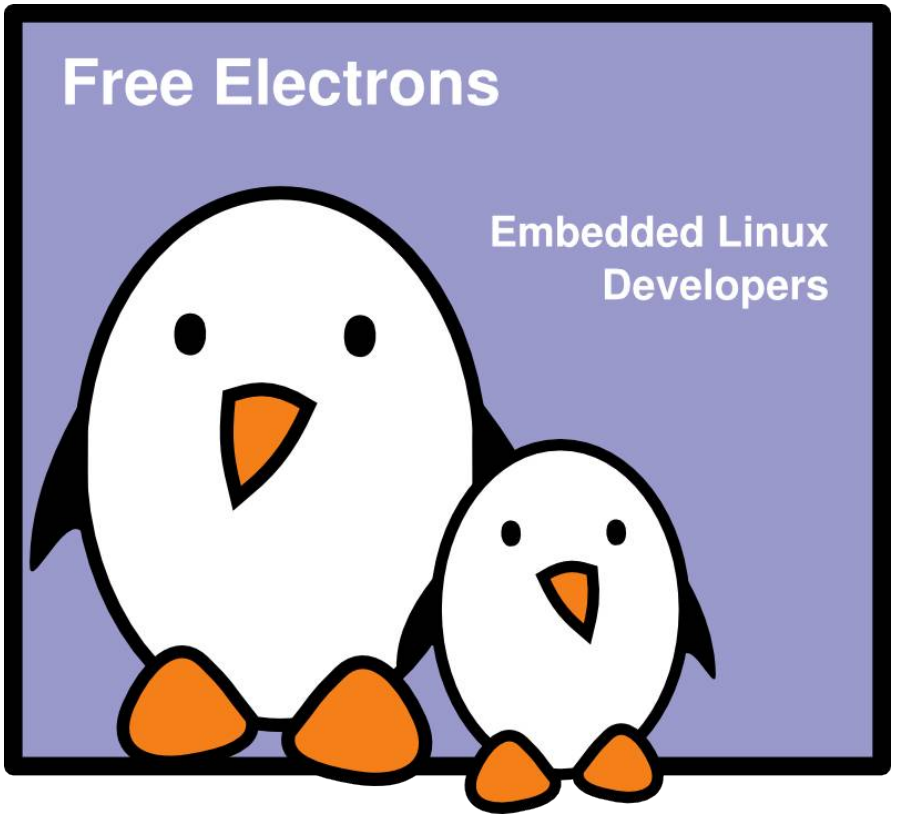
Linux PCI drivers

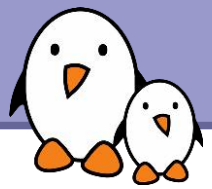
Michael Opdenacker
Free Electrons



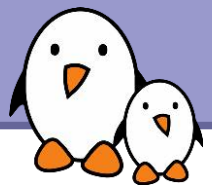
http://en.wikipedia.org/wiki/Image:PCI_Slots_Digon3.JPG



Latest update: Jul 13, 2010,
Document sources, updates and translations:
http://free-electrons.com/docs/pci-drivers
Corrections, suggestions, contributions and translations are welcome!



Understanding PCI

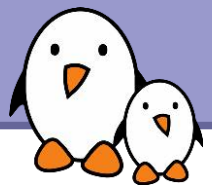


PCI bus family

The following buses belong to the PCI family:

- PCI
  32 bit bus, 33 or 66 MHz
- MiniPCI
  Smaller slot in laptops
- CardBus
  External card slot in laptops
- PIX Extended (PCI-X)
  Wider slot than PCI, 64 bit, but can accept a standard PCI card
- PCI Express (PCIe or PCI-E)
  Current generation of PCI. Serial instead of parallel.
- PCI Express Mini Card
  Replaces MiniPCI in recent laptops
- Express Card
  Replaces CardBus in recent laptops

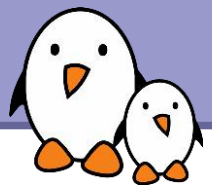
These technologies are compatible and can be handled by the same kernel drivers. The kernel doesn't need to know which exact slot and bus variant is used.



PCI device types

Main types of devices found on the PCI bus

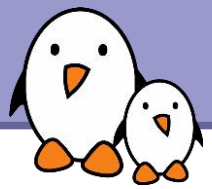
- Network cards (wired or wireless)
- SCSI adapters
- Bus controllers: USB, PCMCIA, I2C, FireWire, IDE
- Graphics and video cards
- Sound cards



PCI features

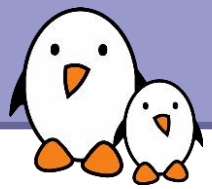
For device driver developers

- Device resources (I/O addresses, IRQ lines) automatically assigned at boot time, either by the BIOS or by Linux itself (if configured).
- The device driver just has to read the corresponding configurations somewhere in the system address space.
- Endianism: PCI device configuration information is Little Endian. Remember that in your drivers
(conversion taken care of by some kernel functions).



PCI device list (1)

```
lscpi
00:00.0 Host bridge: Intel Corporation Mobile 945GM/PM/GMS, 943/940GML and 945GT Express Memory Controller Hub (rev 03)
00:02.0 VGA compatible controller: Intel Corporation Mobile 945GM/GMS, 943/940GML Express Integrated Graphics Controller (rev 03)
00:02.1 Display controller: Intel Corporation Mobile 945GM/GMS/GME, 943/940GML Express Integrated Graphics Controller (rev 03)
00:1b.0 Audio device: Intel Corporation 82801G (ICH7 Family) High Definition Audio Controller (rev 01)
00:1c.0 PCI bridge: Intel Corporation 82801G (ICH7 Family) PCI Express Port 1 (rev 01)
00:1c.1 PCI bridge: Intel Corporation 82801G (ICH7 Family) PCI Express Port 2 (rev 01)
00:1c.2 PCI bridge: Intel Corporation 82801G (ICH7 Family) PCI Express Port 3 (rev 01)
00:1d.0 USB Controller [...]
00:1e.0 PCI bridge: Intel Corporation 82801 Mobile PCI Bridge (rev e1)
00:1f.0 ISA bridge: Intel Corporation 82801GBM (ICH7-M) LPC Interface Bridge (rev 01)
00:1f.1 IDE interface: Intel Corporation 82801G (ICH7 Family) IDE Controller (rev 01)
00:1f.3 SMBus: Intel Corporation 82801G (ICH7 Family) SMBus Controller (rev 01)
02:01.0 CardBus bridge: Ricoh Co Ltd RL5c476 II (rev b4)
02:01.1 FireWire (IEEE 1394): Ricoh Co Ltd R5C552 IEEE 1394 Controller (rev 09)
02:01.2 SD Host controller: Ricoh Co Ltd R5C822 SD/SDIO/MMC/MS/MSPro Host Adapter (rev 18)
09:00.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5752 Gigabit Ethernet PCI Express (rev 02)
0c:00.0 Network controller: Intel Corporation PRO/Wireless 4965 AG or AGN Network Connection (rev 61)

lspci -tv
-[0000:00]-+-00.0  Intel Corporation Mobile 945GM/PM/GMS, 943/940GML and 945GT Express Memory Controller Hub
           +-02.0  Intel Corporation Mobile 945GM/GMS, 943/940GML Express Integrated Graphics Controller
           +-02.1  Intel Corporation Mobile 945GM/GMS/GME, 943/940GML Express Integrated Graphics Controller
           +-1b.0  Intel Corporation 82801G (ICH7 Family) High Definition Audio Controller
           +-1c.0-[0000:0b]--
           +-1c.1-[0000:0c]----00.0  Intel Corporation PRO/Wireless 4965 AG or AGN Network Connection
           +-1c.2-[0000:09]----00.0  Broadcom Corporation NetXtreme BCM5752 Gigabit Ethernet PCI Express
           +-1d.0  Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #1 [...]
           +-1e.0-[0000:02-06]--+-01.0  Ricoh Co Ltd RL5c476 II
           |                    +-01.1  Ricoh Co Ltd R5C552 IEEE 1394 Controller
           |                    \-01.2  Ricoh Co Ltd R5C822 SD/SDIO/MMC/MS/MSPro Host Adapter
           +-1f.0  Intel Corporation 82801GBM (ICH7-M) LPC Interface Bridge
           +-1f.1  Intel Corporation 82801G (ICH7 Family) IDE Controller
           \-1f.3  Intel Corporation 82801G (ICH7 Family) SMBus Controller
```



PCI device list (2)

- `lspci` enumerates all PCI devices

```
02:01.0 CardBus bridge: Ricoh Co Ltd RL5c476 II (rev b4)
02:01.1 FireWire (IEEE 1394): Ricoh Co Ltd R5C552 IEEE 1394 Controller
02:01.2 SD Host controller: Ricoh Co Ltd R5C822 SD/SDIO/MMC/MS/MSPro
```

Function number

PCI device number

PCI bus number

- `lscpi -t` shows the bus device tree

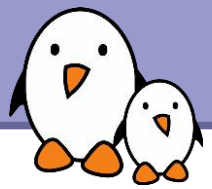
```
-[0000:00]-+-00.0  Intel Corporation Mobile 945GM/PM/GMS, 943/940GML and 945GT Express Memory Controller Hub
           +-02.0  Intel Corporation Mobile 945GM/GMS, 943/940GML Express Integrated Graphics Controller
           +-02.1  Intel Corporation Mobile 945GM/GMS/GME, 943/940GML Express Integrated Graphics Controller
           +-1b.0  Intel Corporation 82801G (ICH7 Family) High Definition Audio Controller
           +-1c.0-[0000:0b]--
           +-1c.1-[0000:0c]----00.0  Intel Corporation PRO/Wireless 4965 AG or AGN Network Connection
           +-1c.2-[0000:09]----00.0  Broadcom Corporation NetXtreme BCM5752 Gigabit Ethernet PCI Express
           +-1d.0  Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #1 [...]
           +-1e.0-[0000:02-06]--+-01.0  Ricoh Co Ltd RL5c476 II
           |                    +-01.1  Ricoh Co Ltd R5C552 IEEE 1394 Controller
           |                    \-01.2  Ricoh Co Ltd R5C822 SD/SDIO/MMC/MS/MSPro Host Adapter
```

PCI domain

PCI bus 0

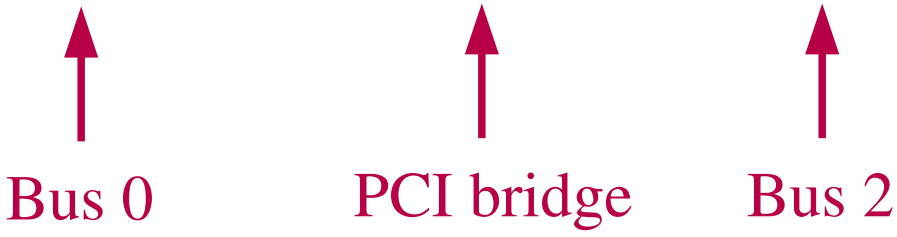
PCI bridge

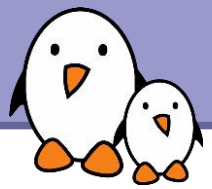
PCI bus 2



PCI device list (3)

- This tree structure reflects the structure in `/sys`:
  `/sys/devices/pci0000:00/0000:00:1e.0/0000:02:01.2`

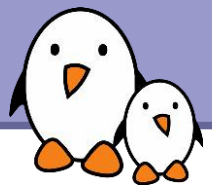




PCI device configuration (1)

- Each PCI device has a 256 byte address space containing configuration registers.
- Device configuration can be displayed with `lspci -x`:

```
0c:00.0 Network controller: Intel Corporation PRO/Wireless
4965 AG or AGN Network Connection (rev 61)
00: 86 80 29 42 06 04 10 00 61 00 80 02 10 00 00 00
10: 04 e0 df ef 00 00 00 00 00 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00 86 80 21 11
30: 00 00 00 00 c8 00 00 00 00 00 00 00 05 01 00 00
```

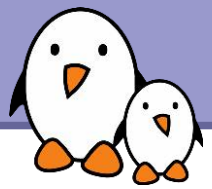


PCI device configuration (2)

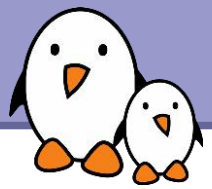
Standard information found in PCI configurations:

- Offset 0: Vendor Id
- Offset 2: Device Id
- Offset 10: Class Id (network, display, bridge...)
- Offsets 16 to 39: Base Address Registers (BAR) 0 to 5
- Offset 44: Subvendor Id
- Offset 46: Subdevice Id
- Offsets 64 and up: up to the device manufacturer

Kernel sources: these offsets are defined in `include/linux/pci_regs.h`



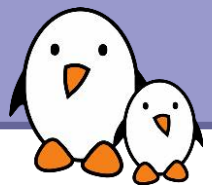
Implementing Linux drivers



Registering supported devices

From `drivers/net/ne2k-pci.c` (Linux 2.6.27):

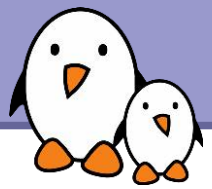
```
static struct pci_device_id ne2k_pci_tbl[] = {
        { 0x10ec, 0x8029, PCI_ANY_ID, PCI_ANY_ID, 0, 0, CH_RealTek_RTL_8029 },
        { 0x1050, 0x0940, PCI_ANY_ID, PCI_ANY_ID, 0, 0, CH_Winbond_89C940 },
        { 0x11f6, 0x1401, PCI_ANY_ID, PCI_ANY_ID, 0, 0, CH_Compex_RL2000 },
        { 0x8e2e, 0x3000, PCI_ANY_ID, PCI_ANY_ID, 0, 0, CH_KTI_ET32P2 },
        { 0x4a14, 0x5000, PCI_ANY_ID, PCI_ANY_ID, 0, 0, CH_NetVin_NV5000SC },
        { 0x1106, 0x0926, PCI_ANY_ID, PCI_ANY_ID, 0, 0, CH_Via_86C926 },
        { 0x10bd, 0x0e34, PCI_ANY_ID, PCI_ANY_ID, 0, 0, CH_SureCom_NE34 },
        { 0x1050, 0x5a5a, PCI_ANY_ID, PCI_ANY_ID, 0, 0, CH_Winbond_W89C940F },
        { 0x12c3, 0x0058, PCI_ANY_ID, PCI_ANY_ID, 0, 0, CH_Holtek_HT80232 },
        { 0x12c3, 0x5598, PCI_ANY_ID, PCI_ANY_ID, 0, 0, CH_Holtek_HT80229 },
        { 0x8c4a, 0x1980, PCI_ANY_ID, PCI_ANY_ID, 0, 0, CH_Winbond_89C940_8c4a },
        { 0, }
};
MODULE_DEVICE_TABLE(pci, ne2k_pci_tbl);
```



Registering a driver (1)

Declaring driver hooks and supported devices table:

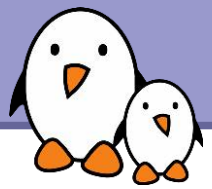
```
static struct pci_driver ne2k_driver = {
        .name           = DRV_NAME,
        .probe          = ne2k_pci_init_one,
        .remove         = __devexit_p(ne2k_pci_remove_one),
        .id_table       = ne2k_pci_tbl,
#ifdef CONFIG_PM
        .suspend        = ne2k_pci_suspend,
        .resume         = ne2k_pci_resume,
#endif /* CONFIG_PM */
};
```



Registering a driver (2)

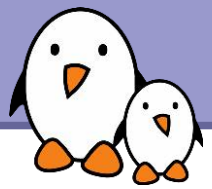
```
static int __init ne2k_pci_init(void)
{
        return pci_register_driver(&ne2k_driver);
}

static void __exit ne2k_pci_cleanup(void)
{
        pci_unregister_driver (&ne2k_driver);
}
```

- The hooks and supported devices are loaded at module loading time.
- The `probe()` hook gets called by the PCI generic code when a matching device is found.
- Very similar to USB device drivers!



Marking driver hooks (1)

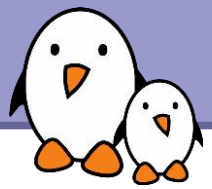
- `__init`: module init function.
  Code discarded after driver initialization.
- `__exit`: module exit function.
  Ignored for statically compiled drivers.
- `__devinit`: probe function and all initialization functions
  Normal function if `CONFIG_HOTPLUG` is set. Identical to `__init` otherwise.
- `__devinitconst`: for the device id table



Marking driver hooks (2)

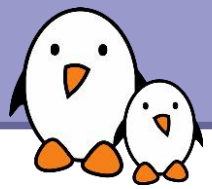
- `__devexit`: functions called at remove time. Same case as in `__devinit`
- All references to `__devinit` function addresses should be declared with `__devexit_p(`*`fun`*`)`. This replaces the function address by `NULL` if this code is discarded.
- Example: same driver:

```
static struct pci_driver ne2k_driver = {
        .name           = DRV_NAME,
        .probe          = ne2k_pci_init_one,
        .remove         =
__devexit_p(ne2k_pci_remove_one),
        .id_table       = ne2k_pci_tbl,
...
};
```



Device initialization steps

- Enable the device
- Request I/O port and I/O memory resources
- Set the DMA mask size
  (for both coherent and streaming DMA)
- Allocate and initialize shared control data
  (`pci_allocate_coherent()`)
- Initialize device registers (if needed)
- Register IRQ handler (`request_irq()`)
- Register to other subsystems (network, video, disk, etc.)
- Enable DMA/processing engines.

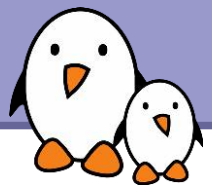


Enabling the device

Before touching any device registers, the driver should first execute `pci_enable_device()`. This will:

- Wake up the device if it was in suspended state
- Allocate I/O and memory regions of the device (if not already done by the BIOS)
- Assign an IRQ to the device (if not already done by the BIOS)

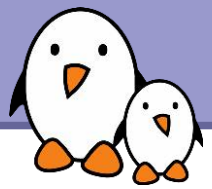
`pci_enable_device()` can fail. Check the return value!



pci_enable_device example

From `drivers/net/ne2k-pci.c` (Linux 2.6.27):

```
static int __devinit ne2k_pci_init_one
    (struct pci_dev *pdev,
     const struct pci_device_id *ent)
{
        ...
        i = pci_enable_device (pdev);
         if (i)
                 return i;

         ...
}
```



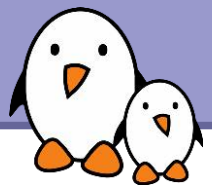
Enabling the device (2)

Enable DMA by calling `pci_set_master()`. This will:

- Enable DMA by setting the bus master bit in the `PCI_COMMAND` register. The device will then be able to act as a master on the address bus.
- Fix the latency timer value if it's set to something bogus by the BIOS.

If the device can use the PCI Memory-Write-Invalidate transaction (writing entire cache lines), you can also call `pci_set_mwi()`:

- This enables the `PCI_COMMAND` bit for Memory-Write-Invalidate
- This also ensures that the cache line size register is set correctly.



Accessing configuration registers (1)

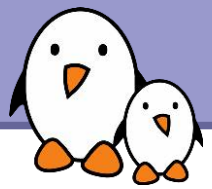
Needed to access I/O memory and port information

- `#include <linux/pci.h>`
- Reading:

```
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);
```

- Example: `drivers/net/cassini.c`

```
pci_read_config_word(cp->pdev, PCI_STATUS, &cfg);
```



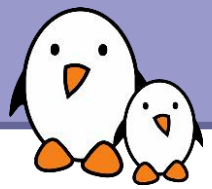
Accessing configuration registers (2)

- Writing:

```
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);
```

- Example: `drivers/net/s2io.c`

```
/* Clear "detected parity error" bit
pci_write_config_word(sp->pdev, PCI_STATUS, 0x8000);
```



Accessing I/O registers and memory (1)

- Each PCI device can have up to 6 I/O or memory regions, described in `BAR0` to `BAR5`.
- Access the base address of the I/O region:

```
#include <linux/pci.h>
long iobase = pci_resource_start (pdev, bar);
```

- Access the I/O region size:

```
long iosize = pci_resource_len (pdev, bar);
```

- Reserve the I/O region:

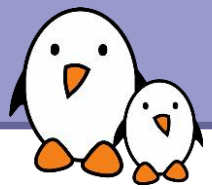
```
request_region(iobase, iosize, “my driver”);
```

or simpler:

```
pci_request_region(pdev, bar, “my driver”);
```

or even simpler (regions for all BARs):

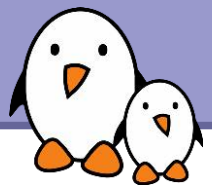
```
pci_request_regions(pdev, “my driver”);
```



Accessing I/O registers and memory (2)

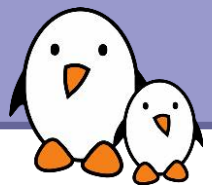
From `drivers/net/ne2k-pci.c` (Linux 2.6.27):

```
ioaddr = pci_resource_start (pdev, 0);
irq = pdev->irq;

if (!ioaddr || ((pci_resource_flags (pdev, 0) & IORESOURCE_IO) == 0))
{
        dev_err(&pdev->dev, "no I/O resource at PCI BAR #0\n");
        return -ENODEV;
}

if (request_region (ioaddr, NE_IO_EXTENT, DRV_NAME) == NULL) {
        dev_err(&pdev->dev, "I/O resource 0x%x @ 0x%lx busy\n",
                NE_IO_EXTENT, ioaddr);
        return -EBUSY;
}
```



Setting the DMA mask size (1)

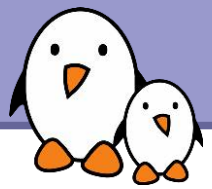
- Use `pci_dma_set_mask()` to declare any device with more (or less) than 32-bit bus master capability
- In particular, must be done by drivers for PCI-X and PCIe compliant devices, which use 64 bit DMA.
- If the device can directly address "consistent memory" in System RAM above 4G physical address, register this by calling `pci_set_consistent_dma_mask()`.



Setting the DMA mask size (2)

Example (`drivers/net/wireless/ipw2200.c` in Linux 2.6.27):

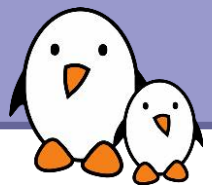
```
err = pci_set_dma_mask(pdev, DMA_32BIT_MASK);

if (!err)
    err = pci_set_consistent_dma_mask(pdev, DMA_32BIT_MASK);
if (err) {
    printk(KERN_WARNING DRV_NAME ": No suitable DMA available.\n");
    goto out_pci_disable_device;
}
```



Allocate consistent DMA buffers

Now that the DMA mask size has been allocated...

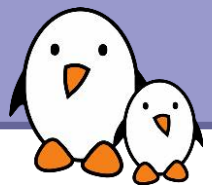
- You can allocate your cache consistent buffers if you plan to use such buffers.
- See our DMA presentation and `Documentation/DMA-API.txt` for details.



Initialize device registers

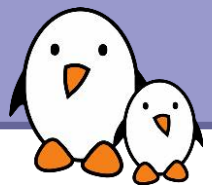
If needed by the device

- Set some “capability” fields
- Do some vendor specific initialization or reset
  Example: clear pending interrupts.



Register interrupt handlers

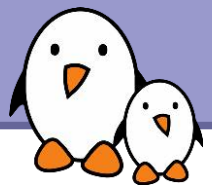
- Need to call `request_irq()` with the `IRQF_SHARED` flag, because all PCI IRQ lines can be shared.
- Registration also enables interrupts, so at this point
  - Make sure that the device is fully initialized and ready to service interrupts.
  - Make sure that the device doesn't have any pending interrupt before calling `request_irq()`.
- Where you actually call `request_irq()` can actually depend on the type of device and the subsystem it could be part of (network, video, storage...).
- Your driver will then have to register to this subsystem.



PCI device shutdown (1)

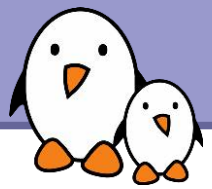
In the `remove()` function, you typically have to undo what you did at device initialization (`probe()` function):

- Disable the generation of new interrupts.
  If you don't, the system will get spurious interrupts, and will eventually disable the IRQ line. Bad for other devices on this line!
- Release the IRQ
- Stop all DMA activity.
  Needed to be done after IRQs are disabled (could start new DMAs)
- Release DMA buffers: streaming first and then consistent ones.



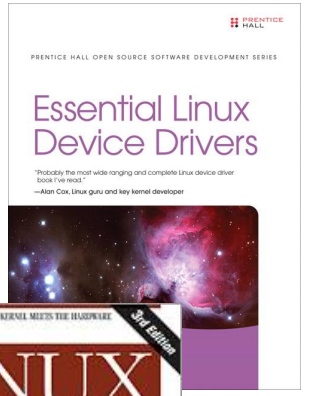
PCI device shutdown

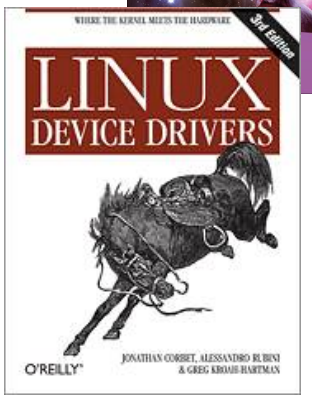
- Unregister from other subsystems
- Unmap I/O memory and ports with `io_unmap()`.
- Disable the device with `pci_disable_device()`.
- Unregister I/O memory and ports.
  If you don't, you won't be able to reload the driver.

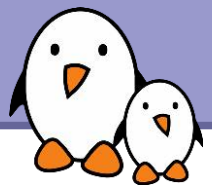


Useful resources

- `Documentation/PCI/pci.txt` in the kernel sources
  An excellent guide to writing PCI drivers, which helped us to write these slides.
- Book: Essential Linux device drivers (Prentice Hall)
  A very good and recent book!
  http://free-electrons.com/redirect/eldd-book.html
- Book: Linux Device Drivers (O'Reilly)
  Available under a free documentation license.
  http://free-electrons.com/community/kernel/ldd3/
- Wikipedia:
  http://en.wikipedia.org/wiki/Peripheral_Component_Interconnect



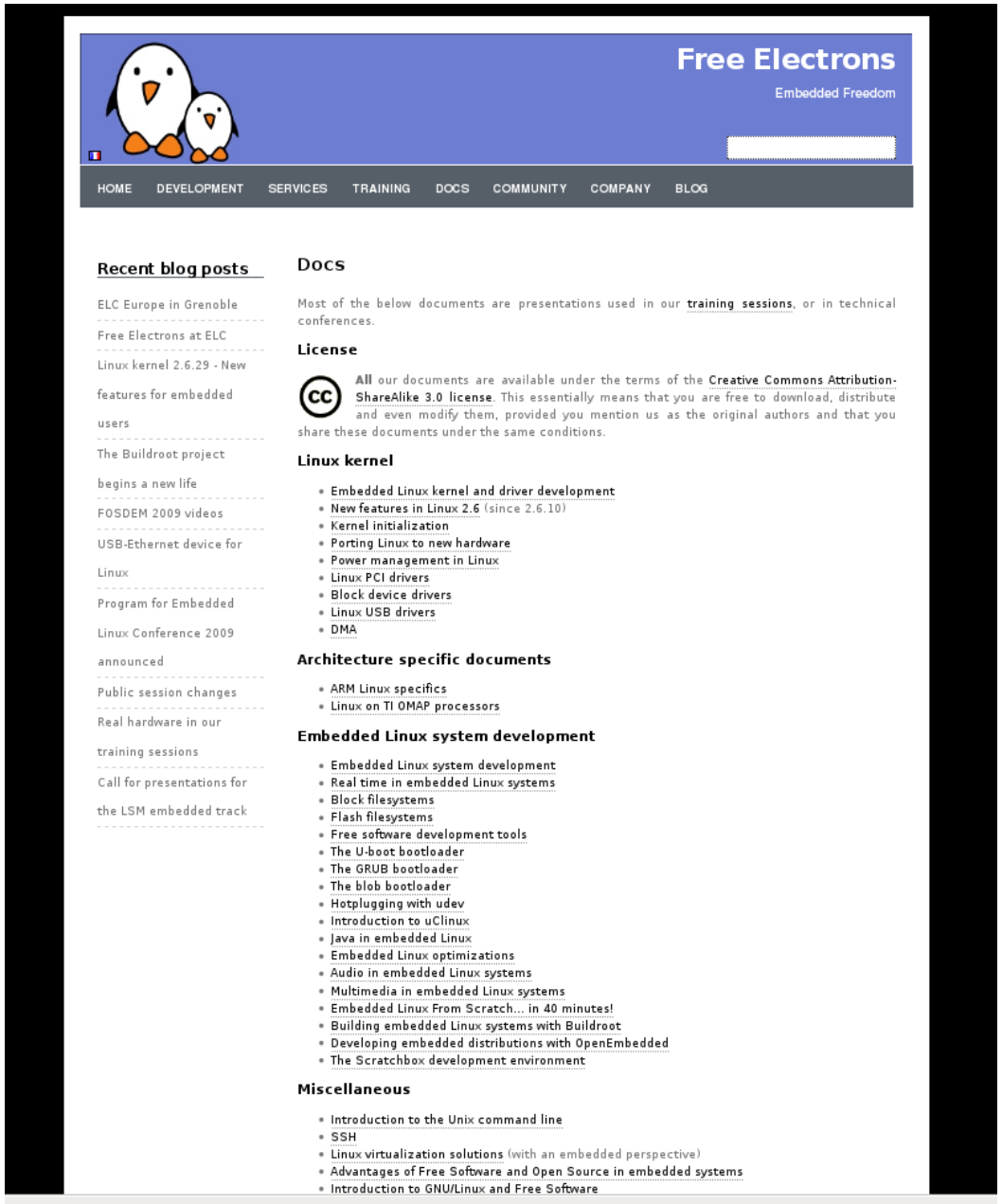




Useful resources (2)

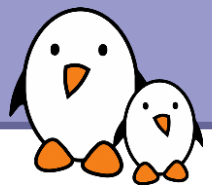
- PCI Utilities: http://mj.ucw.cz/pciutils.html
  `lspci`: shows PCI bus and device information
  `setpci`: allows to manipulate PCI device configuration registers
- PCI vendor and device id repository: http://pci-ids.ucw.cz/read/PC/

Related documents



All our technical presentations on http://free-electrons.com/docs

- Linux kernel
- Device drivers
- Architecture specifics
- Embedded Linux system development



How to help

You can help us to improve and maintain this document...

- By sending corrections, suggestions, contributions and translations
- By asking your organization to order development, consulting and training services performed by the authors of these documents (see http://free-electrons.com/).
- By sharing this document with your friends, colleagues and with the local Free Software community.
- By adding links on your website to our on-line materials, to increase their visibility in search engine results.

Linux kernel

Linux device drivers
Board support code
Mainstreaming kernel code
Kernel debugging

Embedded Linux Training

All materials released with a free license!

Unix and GNU/Linux basics
Linux kernel and drivers development
Real-time Linux, uClinux
Development and profiling tools
Lightweight tools for embedded systems
Root filesystem creation
Audio and multimedia
System optimization

Free Electrons

Our services

Custom Development

System integration
Embedded Linux demos and prototypes
System optimization
Application and interface development

Consulting and technical support

Help in decision making
System architecture
System design and performance review
Development tool and application support
Investigating issues and fixing tool bugs