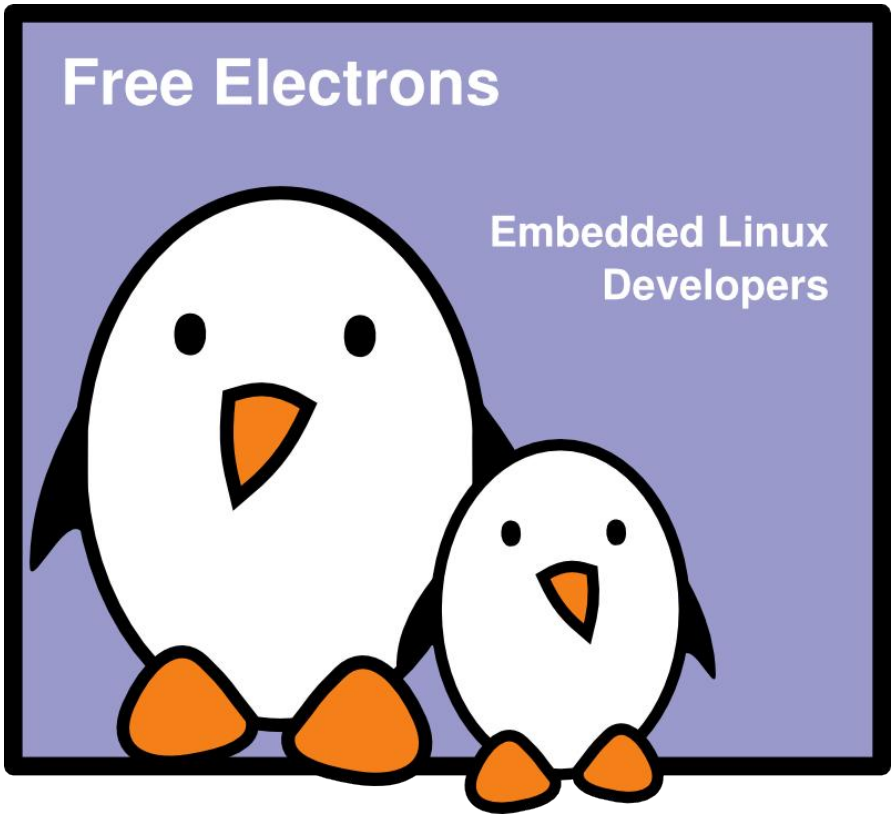
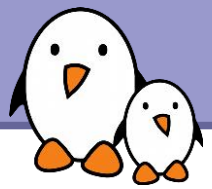
DMA

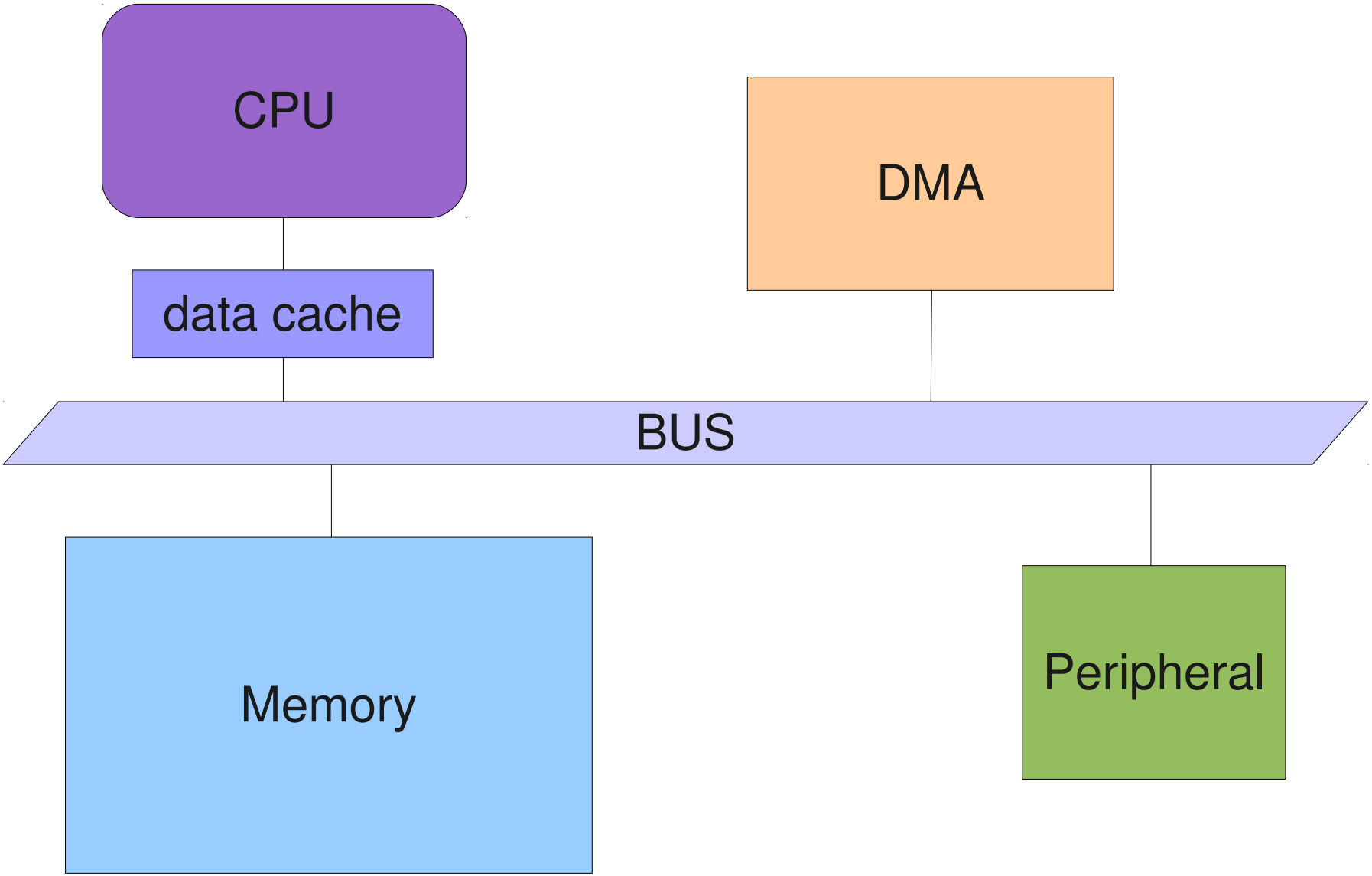
Sebastien Jan
Michael Opdenacker
Thomas Petazzoni
Free Electrons

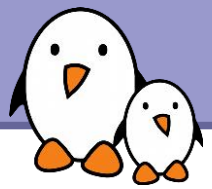


Latest update: Feb 21, 2011,
Document sources, updates and translations:
http://free-electrons.com/docs/dma
Corrections, suggestions, contributions and translations are welcome!



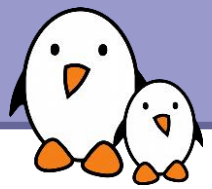
DMA integration





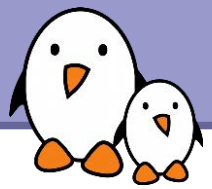
Constraints with a DMA

- A DMA deals with physical addresses, so:
  - Programming a DMA requires retrieving a physical address at some point (virtual addresses are usually used)
  - The memory accessed by the DMA shall be physically contiguous
- The CPU can access memory through a data cache
  - Using the cache can be more efficient (faster accesses to the cache than the bus)
  - But the DMA does not access to the CPU cache, so one need to take care of cache coherency (cache content vs memory content)
  - Either flush or invalidate the cache lines corresponding to the buffer accessed by DMA and processor at strategic times



DMA memory constraints

- Need to use contiguous memory in physical space.
- Can use any memory allocated by `kmalloc` (up to 128 KB) or `__get_free_pages` (up to 8MB).
- Can use block I/O and networking buffers, designed to support DMA.
- Can not use `vmalloc` memory (would have to setup DMA on each individual physical page).

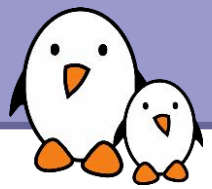


Reserving memory for DMA

To make sure you've got enough RAM for big DMA transfers...
Example assuming you have 32 MB of RAM, and need 2 MB for DMA:

- Boot your kernel with `mem=30`
  The kernel will just use the first 30 MB of RAM.
- Driver code can now reclaim the 2 MB left:

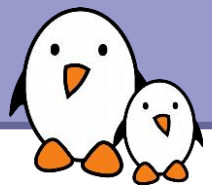
```
dmabuf = ioremap (
                0x1e00000,          /* Start: 30 MB */
                0x200000            /* Size: 2 MB */
                );
```



Memory synchronization issues

Memory caching could interfere with DMA

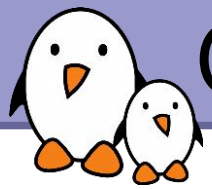
- Before DMA to device:
  Need to make sure that all writes to DMA buffer are committed.
- After DMA from device:
  Before drivers read from DMA buffer, need to make sure that memory caches are flushed.
- Bidirectional DMA
  Need to flush caches before and after the DMA transfer.



Linux DMA API

The kernel DMA utilities can take care of:

- Either allocating a buffer in a cache coherent area,
- Or making sure caches are flushed when required,
- Managing the DMA mappings and IOMMU (if any).
- See `Documentation/DMA-API.txt` for details about the Linux DMA generic API.
- Most subsystems (such as PCI or USB) supply their own DMA API, derived from the generic one. May be sufficient for most needs.



Coherent or streaming DMA mappings

Coherent mappings

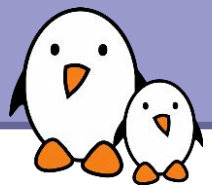
The kernel allocates a suitable buffer and sets the mapping for the driver.

- Can simultaneously be accessed by the CPU and device.
- So, has to be in a cache coherent memory area.
- Usually allocated for the whole time the module is loaded.
- Can be expensive to setup and use on some platforms.

Streaming mappings

The kernel just sets the mapping for a buffer provided by the driver.

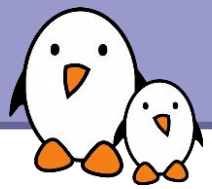
- Use a buffer already allocated by the driver.
- Mapping set up for each transfer. Keeps DMA registers free on the hardware.
- Some optimizations also available.
- The recommended solution.



Allocating coherent mappings

The kernel takes care of both buffer allocation and mapping:

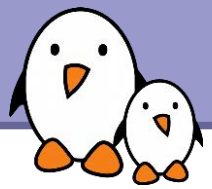
```
include <asm/dma-mapping.h>

void *                                  /* Output: buffer address */
  dma_alloc_coherent(
     struct device *dev,                /* device structure */
     size_t size,                       /* Needed buffer size in bytes */
     dma_addr_t *handle,                /* Output: DMA bus address */
     gfp_t gfp                          /* Standard GFP flags */
  );

void dma_free_coherent(struct device *dev,
  size_t size, void *cpu_addr, dma_addr_t handle);
```



Setting up streaming mappings

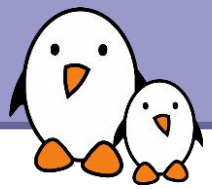
Works on buffers already allocated by the driver

```
<include linux/dmapool.h>

dma_addr_t dma_map_single(
  struct device *,                          /* device structure */
  void *,                                   /* input: buffer to use */
  size_t,                                   /* buffer size */
  enum dma_data_direction  /* Either DMA_BIDIRECTIONAL,
                 DMA_TO_DEVICE or DMA_FROM_DEVICE */
  );

void dma_unmap_single(struct device *dev, dma_addr_t
  handle, size_t size, enum dma_data_direction dir);
```



DMA streaming mapping notes

- When the mapping is active: only the device should access the buffer (potential cache issues otherwise).
- The CPU can access the buffer only after unmapping! Use locking to prevent CPU access to the buffer.
- Another reason: if required, this API can create an intermediate *bounce buffer* (used if the given buffer is not usable for DMA).
- The Linux API also supports scatter / gather DMA streaming mappings.



DMA summary

Most drivers can use the specific API provided by their subsystem: USB, PCI, SCSI... Otherwise they can use the Linux generic API:

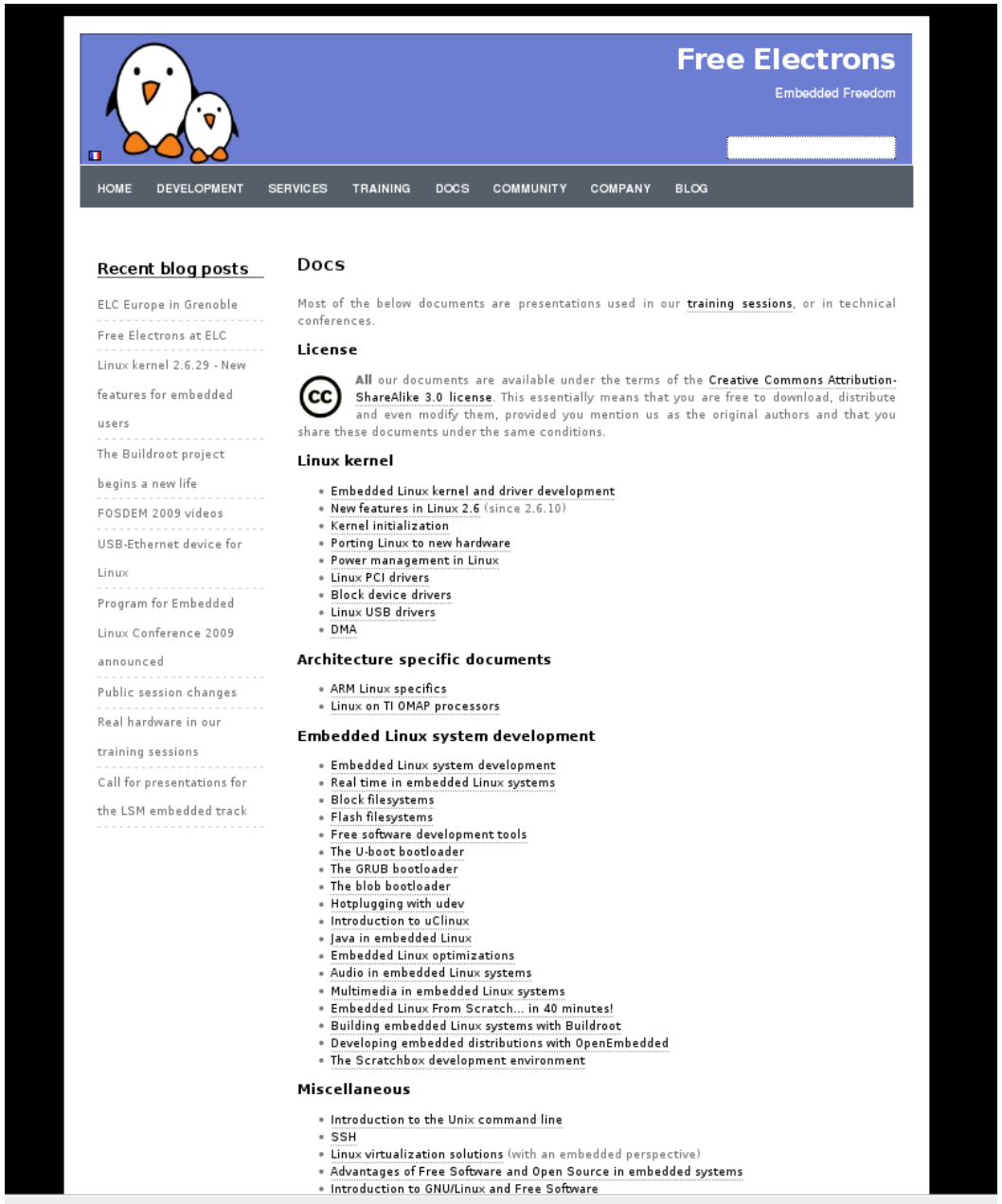
Coherent mappings

- DMA buffer allocated by the kernel
- Set up for the whole module life
- Can be expensive. Not recommended.
- Let both the CPU and device access the buffer at the same time.
- Main functions:
  dma_alloc_coherent
  dma_free_coherent

Streaming mappings

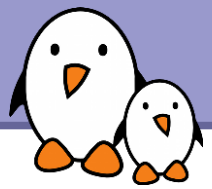
- DMA buffer allocated by the driver
- Set up for each transfer
- Cheaper. Saves DMA registers.
- Only the device can access the buffer when the mapping is active.
- Main functions:
  dma_map_single
  dma_unmap_single

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