Next-Generation DMABUF

How To Efficiently Play Back Video on Embedded Systems

Embedded Linux Conference Europe
Edinburgh, 2013-10-25
Lucas Stach  <l.stach@pengutronix.de>
Philipp Zabel <p.zabel@pengutronix.de>
Agenda

- Simple videoplayback using Gstreamer
- Adding hardware units in the mix
- DMA-BUF – why and how
- Current DMA-BUF flaws → our solution
GStreamer software pipeline

UVC → Buffer → Storage

SW-Scaler

DRM → Buffer → Storage

mmap

http://www.pengutronix.de - 29.10.2013
Now add another HW element
Video4Linux UserPTR

Diagram showing the process flow involving UVC, HW-Scaler, DRM, and Storage, with pointers and buffer management.
Introducing DMABUF
Fundamental DMABUF API

```c
struct dma_buf_attachment *
dma_buf_attach(struct dma_buf *dmabuf, struct device *dev);

struct dma_buf_attachment {
    struct dma_buf *dmabuf;
    struct device *dev;
    struct list_head node;
    void *priv;
};

void dma_buf_detach(struct dma_buf *dmabuf,
                     struct dma_buf_attachment *dmabuf_attach);
```
Fundamental DMABUF API

```c
struct sg_table *
dma_buf_map_attachment(struct dma_buf_attachment *,
                        enum dma_data_direction);

void
dma_buf_unmap_attachment(struct dma_buf_attachment *,
                          struct sg_table *,
                          enum dma_data_direction);
```
Sounds like a good idea and reasonably easy, but ...
Possible memory constraints

- different DMA windows
- contiguous vs. paged
- different MMU page sizes
Common restriction on embedded systems

- devices unable to do scather-gather DMA
- no IOMMU available

→ DMA memory needs to be physically contiguous
Mixed systems...

UVC

Buffer

scather-gather storage

fd

exp-buf

req-buf

HW-Scaler

Buffer

Pengutronix
Our solution

Transparent backing store migration
Prerequisites

- drivers need to be able to describe their device's DMA capabilities
- commonly known: dma_mask
- there's more:

```c
struct device_dma_parameters {
    unsigned int min_segment_size;
    unsigned int max_segment_size;
    unsigned long segment_boundary_mask;
    unsigned int max_segments;
};
```
Prerequisites

• drivers need a more generic way for allocating backing store

• traditional DMA-API:

```c
void *
dma_alloc_attrs(struct device * dev, size_t size, dma_addr_t
     *dma_handle, gfp_t flag, struct dma_attrs * attrs)
```

What's wrong with that?
Prerequisites

- new way to allocate DMA memory

```c
int
arm_dma Alloc_sgt(struct device *dev, size_t size,
        struct sg_table *sgt, gfp_t gfp,
        struct device_dma_parameters *dma_parms);

struct sg_table {
    struct scatterlist {
        unsigned long page_link;
        unsigned int length;
        dma_addr_t dma_address;
    } *sgl;
    unsigned int nents;
};
```
Prerequisites

- map for device with well-known DMA-API

  ```c
  int dma_map_sg(struct device *dev, struct scatterlist *sg, int nents,
                 enum dma_data_direction dir, struct dma_attrs *attrs)
  ```

- map for CPU with new function

  ```c
  void *dma_cpumap_sgtable(struct device *dev, struct sg_table *sgt,
                           pgprot_t prot);
  ```
Migration

- `dma_buf_map_attachment`
  - current storage compatible with attachment?
    - Yes
      -> return sg_table
    - No
      -> wait for other maps to go away
      -> reallocate storage
Reallocation

- try to find storage dma parameters compatible with all currently attached devices

```c
int dma_coalesce_constraints(int num parms,
   struct device_dma_parameters **inparms,
   struct device_dma_parameters *outparms)
```

- if not possible use parameters from device currently trying to map and exporter only
- last resort: parameters from mapping device only
- use parameters to alloc new storage
Migration

- dma_buf_map_attachment
  - current storage compatible with attachment?
    - Yes
      → return sg_table
    - No
      → wait for other maps to go away
      → reallocate storage
      → move current content to new storage
Move buffer content

- simple and almost always working:
  - map both buffers to CPU
  - memmove()

- exporter is free to implement optimized move
  - examples:
    - GPU behind MMU can blit content
    - usage of dedicated on-chip DMA engines
Migration

- *dma_buf_map_attachment*
  - *current storage compatible with attachment?*
    - *Yes*
      → return *sg_table*
    - *No*
      → wait for other maps to go away
      → reallocate storage
      → move current content to new storage
      → return *sg_table* to new storage
Why isn't this dead slow?

• GStreamer reuses allocated buffers – and you should too
Corner cases

- sharing a buffer between devices with no overlap in `device_dma_parameters`
  - will work, but leads to ping-pong

- devices with memory not accessible to CPU and no way to migrate a buffer on its own
  - Do you know of any real world example?
  - If you can't access a common memory region, why are you sharing a buffer?
Possible optimization

- Delay allocation to last possible point in time
  → alloc when first user wants to read/write

- Userspace hands buffer handle to all devices before starting the pipeline
  → all users attach before usage

  → exporter is able to allocate matching storage right from the start