DMA Buffer Sharing Framework: An Introduction

Embedded Linux Conference, SFO, 2012

Rob Clark (rob@ti.com / rob.clark@linaro.org)

Sumit Semwal (sumit.semwal@ti.com / sumit.semwal@linaro.org)

Linaro

Agenda

- Why DMA Buffer Sharing?
- What is dma_buf buffer sharing API?
- dma_buf operations
- Buffer Exporter
- Buffer Importer
- API
- Example of sharing
- Importance of buffer operations
- Next Steps
- References

Linaro

Shared DMA Buffers

- Use cases (example):
  - Decoding video stream into buffers suitable for graphics rendering and display.
  - Camera capture into buffers suitable for encoding and rendering.
- Requirements
  - Support added to *existing* kernel subsystems
  - Hardware that allows common pixel formats and pages to be mapped to multiple devices

Why DMA buffer sharing?

- A uniform mechanism to share DMA buffers across different devices and sub-systems does not exist.
- Different Approaches
  - Video for Linux (V4L2) has a 'USERPTR' mechanism, which requires userspace mmap if the underlying memory is coming from some other device.
  - Similarly, Wayland and X11 has their own mechanisms, to share buffers between client and compositor processes only (but does not define sharing between devices).
  - Other SoC vendors, frameworks and sub-systems also have their own ways of sharing buffers.
- Problems are:
  - A userspace mmap might be necessary (such as in V4L2), which might be an overkill as it oscillates between kernel-space and user-space. (And doesn't allow for migration.)
  - Uniform Sharing APIs are not available, so there is no uniform way available to share.

What is dma_buf API?

- A generic kernel level framework to share buffers.
- Defines a new buffer object, which provides mechanism for exporting and using shared buffers.
- Provides uniform APIs that allow various operations related to buffer sharing.

Operations defined on dma_buf

- **attach()**
  - This is an optional op, which could allow the exporter to gather information about the attached devices, their needs about the buffer (like backing storage constraints etc) and then make decisions accordingly.
  - Can also be used to return error in case the needs of the requesting device can not be met by the exporter
- **detach()**
  - This is an optional op, which could allow the exporter to decide if migration of backing storage is required, and similar decisions about buffers based on the updated needs of still-attached devices.

Operations defined on dma_buf

- **map_dma_buf()**
  - Mandatory dma_buf_op.
  - Used by importing device to indicate to exporter the request to start DMA access.
  - Returns the list of scatter pages allocated, mapped in device address space.
  - At least one attach must be called before call to map_dma_buf().
  - Should increase the pin count of this buffer.
- **unmap_dma_buf()**
  - Mandatory dma_buf_op.
  - Used by importing device to indicate to exporter the end of DMA access.
  - Should decrease the use count of this buffer.
  - Might unpin buffer (allow it to be moved in memory, swapped out, etc).. although it is expected for performance reasons that the exporter would not migrate buffers unnecessarily.
  - Once all current users have relinquished access, the exporter might decide to migrate the buffer storage if required and possible on a platform

Buffer Exporter

- The exporter:
  - Implements and manages all operations on the buffer
  - Allows other users to share buffer using the dma_buf sharing APIs
  - Manages the actual allocation of memory backing storage
  - Optionally, and if possible on a platform, takes care of migration of scatterlist if required.

Buffer Importer (user)

- The importer:
  - Is one of (potentially many) sharing users of the buffer.
  - Is not concerned with wheres and hows of buffer allocation.
  - Needs a mechanism to get access to the scatterlist that makes up this buffer in memory, so it can access the same.

API

- **dma_buf_export():**
  - Used to announce the wish to export a buffer
  - Connects the exporter's private metadata for the buffer, an implementation of buffer operations for this buffer, and flags for the associated file.
  - Returns a handle to the dma_buf object with all the above associated information.
- **dma_buf_fd():**
  - Returns a FD associated with the dma_buf object.
  - Userspace then passes the FD to other devices / sub-systems participating in sharing this dma_buf object.
- **dma_buf_get():**
  - Used by importing device to get the dma_buf object associated with the FD

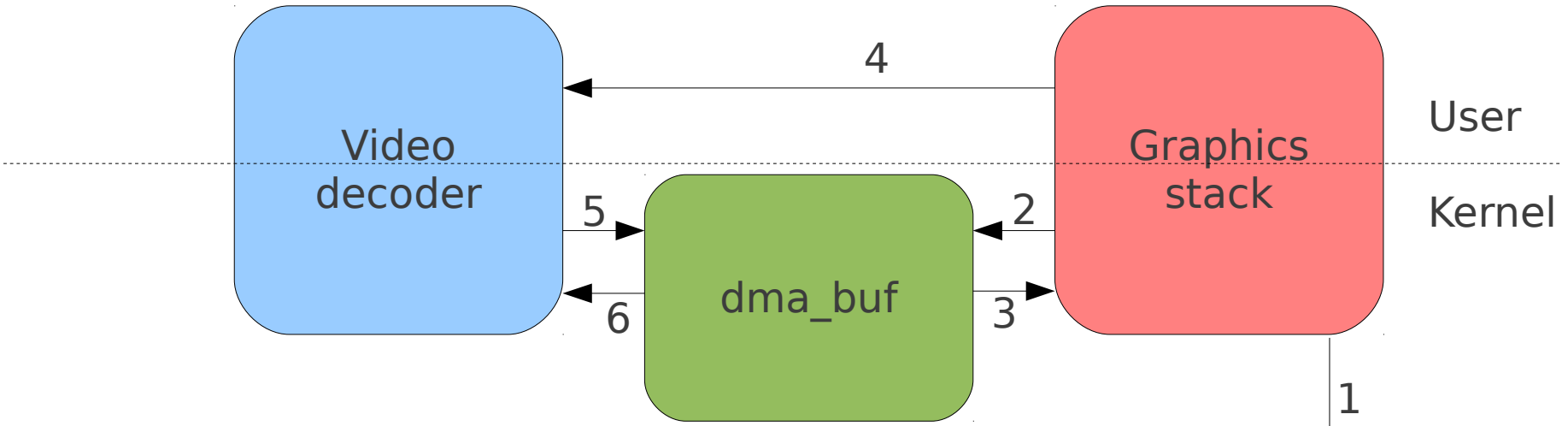
API

- **dma_buf_attach():**
  - The importing device can attach itself with the dma_buf object.
  - Called once at beginning of dma_buf object sharing.
  - May call attach() dma_buf_op if provided by exporter.
  - Returns a struct dma_buf_attachment ptr
- **dma_buf_map_attachment():**
  - Used by importing device to request access to the buffer so it can do DMA.
  - Internally calls map_dma_buf() dma_buf_op provided by exporter.
  - Returns an sg_table, containing the scatterlist mapped in the importing device's address space.
- **dma_buf_unmap_attachment():**
  - Once the DMA access is done, the device tells the exporter that the currently requested access is completed by calling this API.
  - Internally calls unmap_dma_buf() dma_buf_op provided by exporter.

API

- **dma_buf_detach():**
  - At the end of need to access this dma_buf object, the importer device tells the exporter of its intent to 'detach' from the current sharing.
  - May call detach() dma_buf_op if provided by the exporter.
- **dma_buf_put():**
  - After dma_buf_detach() is called, the reference count of this buffer is decremented by calling dma_buf_put().

dma_buf usage flow



1) allocation (CMA usage is optional).

2) dma_buf_export(): request the creation of a dma_buf for previously allocated memory.

3) dma_buf_fd(): provides a fd to return to userspace.

4) fd passed to video decoder.

5) dma_buf_get(fd): takes ref and returns 'struct dma_buf'.

6) dma_buf_attach() + dma_buf_map_attachment(): to get info for dma

   a) dev->dma_parms should be expanded to tell if receiving device needs contiguous memory or any other special requirements

   b) allocation of backing pages could be deferred by exporting driver until it is known if importing driver requires contiguous memory.. to make things a bit easier on systems without IOMMU

Importance of buffer operations

- dma_buf_{attach, detach}:
  - This operation pair allows the exporter to figure out backing-storage constraints for the currently-interested users. Allows preferential allocation, and/or migration of pages across different types of storage available, if possible.
- dma_buf_{map, unmap}_attachment:
  - Bracketing the DMA access with this operation pair is essential to allow just-in-time backing of storage, and migration mid-way through a use-case if possible.

Next Steps

- Define CPU access
- eglImage extension for dma_buf providing for userspace CPU access with adequate fencing/barriers to deal with asynchronous command submission to GPU
- Sync-objects
- Dealing with “expensive importers”

References

- [1]: Linaro OCTO documentation effort: UMM presentation at LPC 2011
- [2]: LWN article about dma_buf sharing: http://lwn.net/Articles/474819/

Linaro