Media Controller (MC) and OMAP2+ Display Subsystem (DSS)

Embedded Linux Conference, SFO, 2011

Sumit Semwal

Agenda

OMAP2+ Display Subsystem (DSS)

- Overview of OMAP2+ DSS
- Current software design – DSS2 framework
- Interesting Display Use cases

Media Controller (MC)

- Overview
- MC – how it helps DSS2
- MC mapping to DSS
- Media entity, pads, links and registration
- MC – DSS: Userspace API
- MC – DSS: Writeback (mem-2-mem)
- MC – DSS: Writeback (capture)
- Limitations in moving to MC + V4L2

Direct Rendering Manager (DRM)

- Overview
- DRM – Nice features
- Gaps

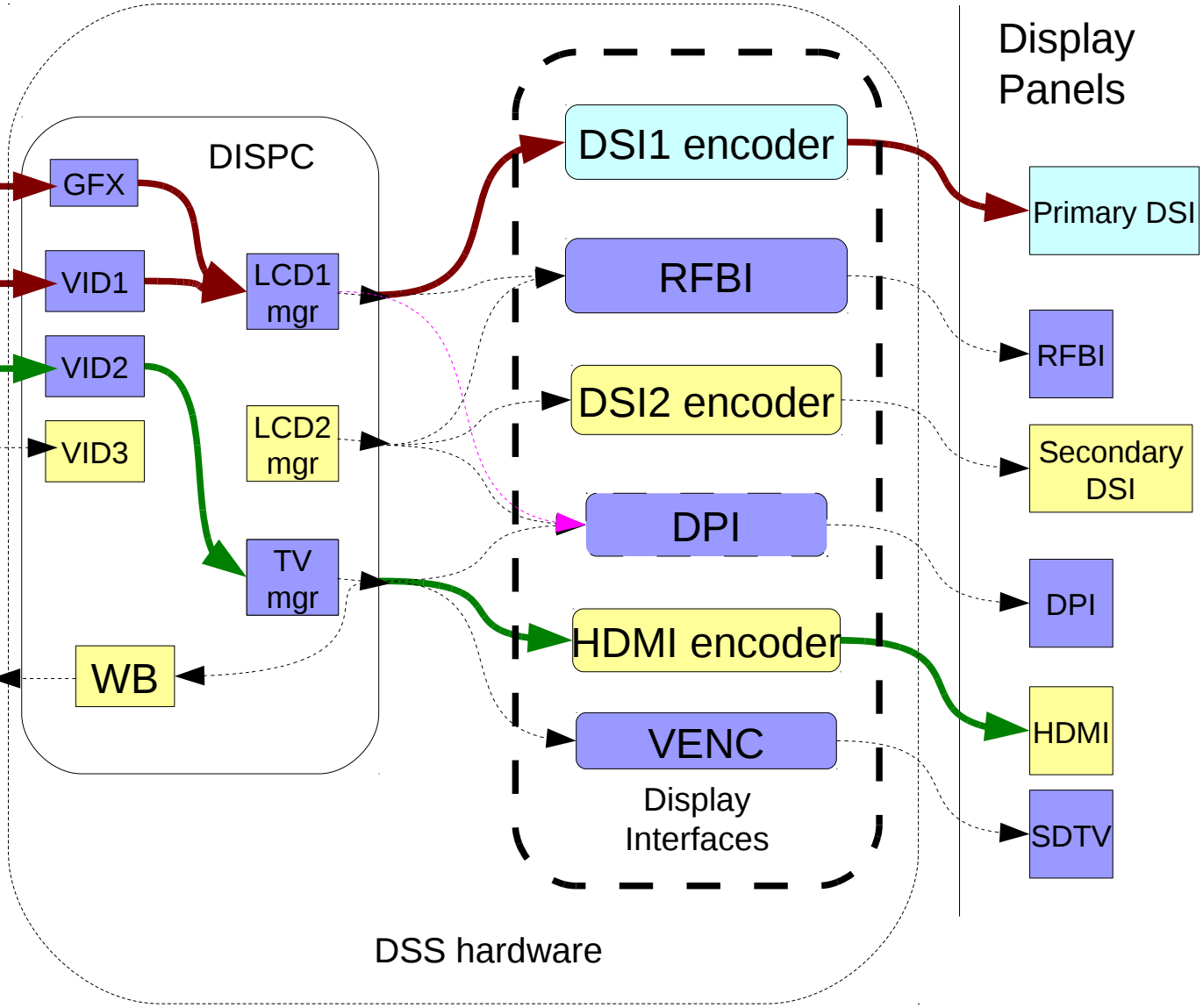
Inter-op need between MC and DRM

Summary

Q & A

Overview of OMAP2+ DSS

OMAP2+ DSS hardware: Overview



OMAP2,3,4
OMAP3,4
OMAP4

- Gfx, vid1-vid3 are overlays, which color-convert, scale each frame

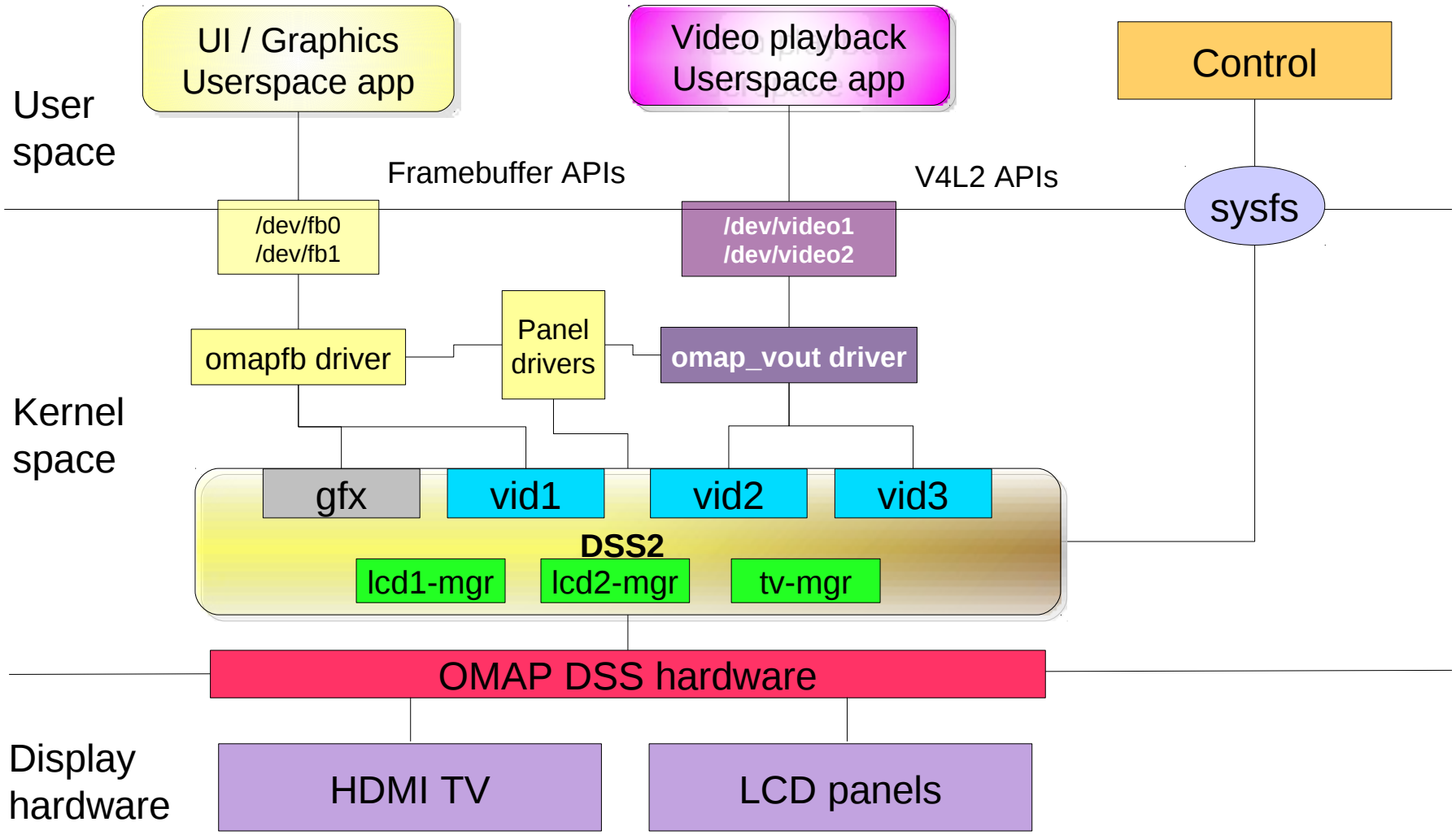
- Managers can compose, overlay multiple streams into one.

- Writeback is a device to writeback to memory.

- Not all paths are shown: All overlays connect to all overlay managers, and WB. In addition, all overlay managers can also give output in parallel to WB.

- WB, or writeback can take inputs from either overlays' or managers' output.

Current software design – DSS2 framework



Current software design – DSS2 framework

- S/w maps one-to-one w/ DSS hardware:
  - One overlay entity for each overlay
  - One manager entity for each overlay manager
  - One display interface for each type of panel connectable [DSI / DPI / SDI / HDMI interfaces]
  - Panel drivers are written which adhere to OMAP panel driver APIs.
  - Control via DSS2 sysfs entries, common to both UI and Video – entries for overlays, managers and displays.
  - Writeback is a kind of 'special' case right now, as it maps both as an overlay and an overlay manager.
- 2 Users of DSS2 - FB and V4L2. No common arbitrator between them.
  - 'pre-defined' policy about allocation of pipelines to FB and V4L2.

Interesting Display Use cases

- Writeback
  - Memory-to-memory
    - Store processed images (after color-space conversion, scaling, rotation, or even composition)
  - Capture mode
    - Display and also record composed content
- Dynamic configurability, parallel usage
  - Switch output paths, using multiple paths simultaneously
- Handle audio interfacing additionally
  - HDMI support
  - DSI support in future
- Support for 'virtual' pipelines – when we run out of video pipelines
  - Co-use of 2D/3D graphics accelerator.
- Multi-display configurations
  - Clone, Virtual.
- Display security

Media Controller (MC) - Overview

MC Overview

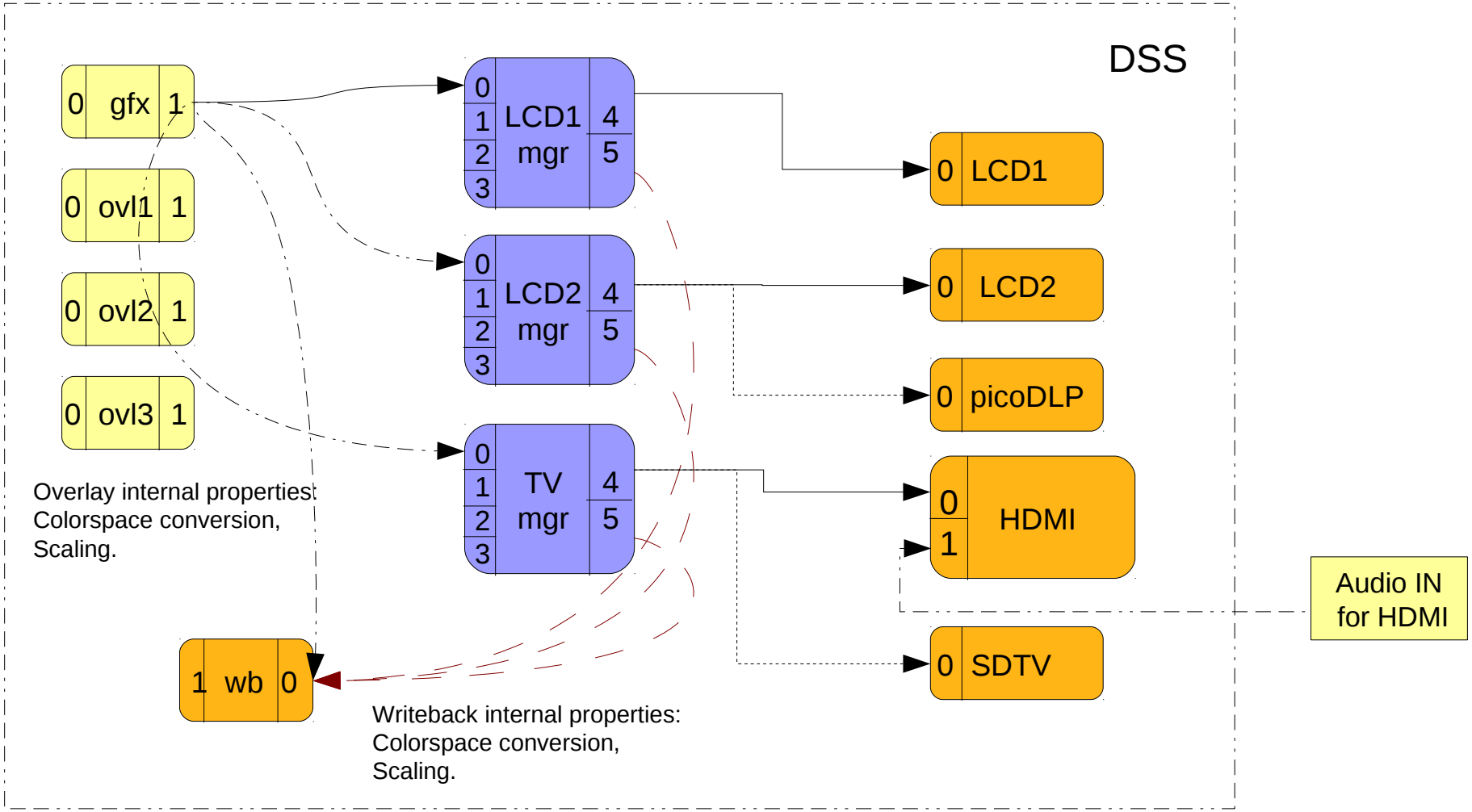
- What?
  - Framework built to model device topology.
- Why?
  - Userspace applications can have finer control over media device parameters.
  - Allow configurability and flexibility of setting multiple, parallel execution paths.
- How?
  - Expose media device topology to userspace, as a graph of building blocks called **entities** connected through **pads**.
  - Enable/disable **links** from userspace.
  - Give access to entities **internal parameters** through read/write/ioctl calls.
  - Configure image **streaming parameters** at each pad.

Ref: Laurent Pinchart's MC presentation at V4L2 summit

MC – how it helps DSS2

- Use cases that benefit
  - Mapping of DSS elements, connections
    - Each overlay, each manager map to an MC entity.
    - Properties of each DSS element map to internal parameters.
    - Connections between overlays and managers map as links.
  - Dynamic reconfiguration
    - The **dynamic** nature of links helps **re-routing** of content from one display to another, or from overlay to writeback (mem-to-mem mode), or display to writeback (capture mode)
  - Handling of Audio and Video as separate MC entities
    - Helps clean design of HDMI-like AV panels [with both Audio and Video]
  - Writeback
  - HDMI

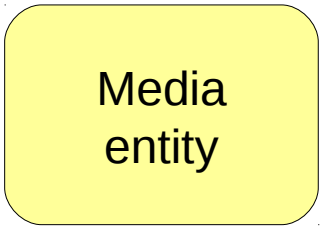
MC mapping to DSS



* doesn't show all paths possible, but I'm sure you get the drift :) - All overlays can connect to any one of (managers or WB)
LCD1 Mgr can connect to LCD1, LCD2 Mgr can connect to LCD2 or picoDLP, TV Mgr to either HDMI or SDTV.
WB can take input from either one of overlays, or from one of managers.

Media entity

```
struct media_entity
{
        u32 id;
        const char *name;
        u32 type;
        ...
};
```



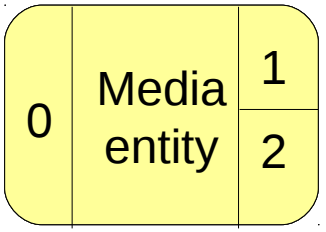
media_entity::type

- contains info about both type and subtype
- Type can be either a NODE or a SUBDEV
- Multiple subtype of DEVNODEs and SUBDEVs can be added

MEDIA_ENT_T_DEVNODE
MEDIA_ENT_T_DEVNODE_V4L
:
:
(proposed)
MEDIA_ENT_T_V4L2_SUBDEV_DSS_OVL
MEDIA_ENT_T_V4L2_SUBDEV_DSS_MGR
MEDIA_ENT_T_V4L2_SUBDEV_DSS_WB

Media entity - pads

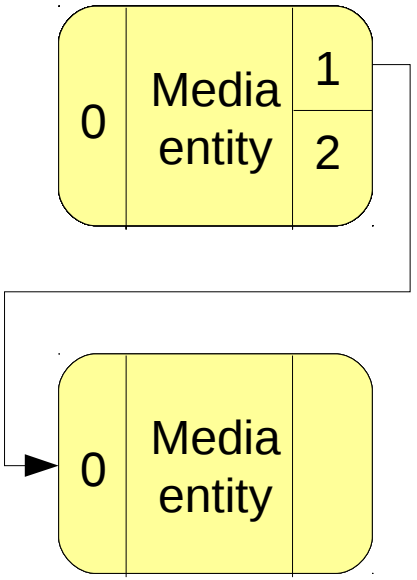
```
struct media_entity
{
        ...
        u16 num_pads;
        struct media_pad *pads;
        ...
};

struct media_pad {
        struct media_entity *entity;
        u16 index;
        unsigned long flags;
};
```



media_pad::flags
- MEDIA_PAD_FL_SINK
- MEDIA_PAD_FL_SOURCE

Media entity - links

```
struct media_entity
{
        ...
        u16 num_links;
        struct media_link *links;
        ...
};

struct media_link {
        struct media_pad *source;
        struct media_pad *sink;
        struct media_link *reverse;
        unsigned long flags;
};
```

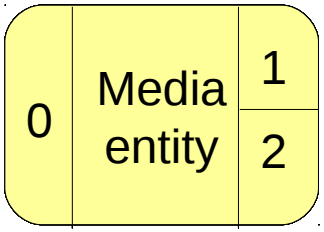


media_link::flags

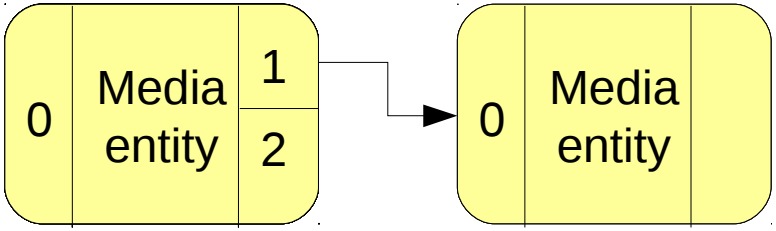
- MEDIA_LNK_FL_ENABLED
- MEDIA_LNK_FL_IMMUTABLE
- MEDIA_LNK_FL_DYNAMIC

Media entity registration

Initialize entity
media_entity_init()



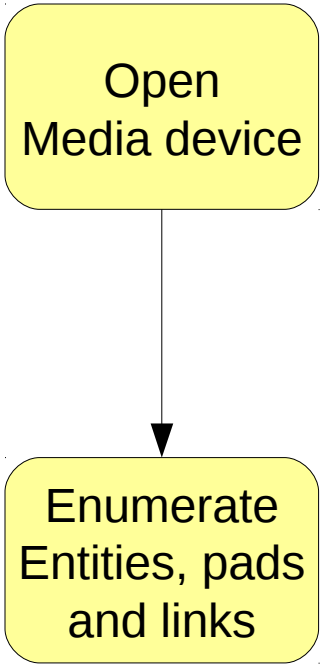
Create links
media_entity_create_link()



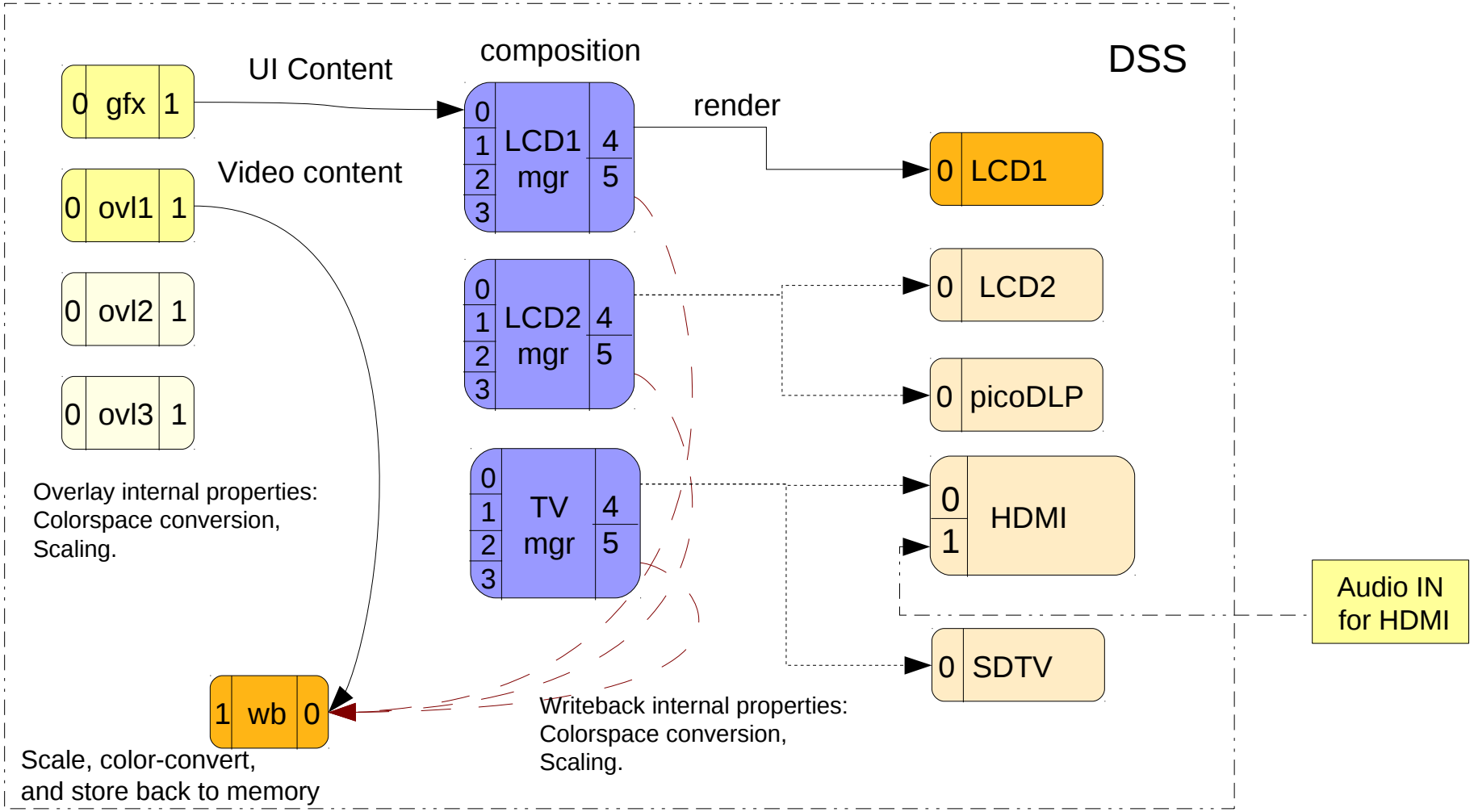
Register entity
media_device_register_entity()

MC – DSS: Userspace API

```
int fd;
fd = open("/dev/media0", O_RDWR);
while (1) {
     struct media_entity_desc entity;
     struct media_links_enum links_enumerator;
     struct media_pad_desc *pads;
     struct media_link_desc *links;

     entity.id = MEDIA_ENT_ID_FLAG_NEXT;
     ret = ioctl(fd, MEDIA_IOC_ENUM_ENTITIES,&entity);
     if (ret < 0)
          Break;

     links_enumerator.entity = entity.id;

     pads = (struct media_pad_desc*) malloc (sizeof(struct
media_pad_desc) * entity.pads);
     links = (struct media_link_desc*) malloc (sizeof(struct
media_link_desc) * entity.links);

     links_enumerator.pads = pads;
     links_enumerator.links = links;

     ret = ioctl(fd, MEDIA_IOC_ENUM_LINKS, &links_enumerator);
     if (ret < 0)
          break;
}
```



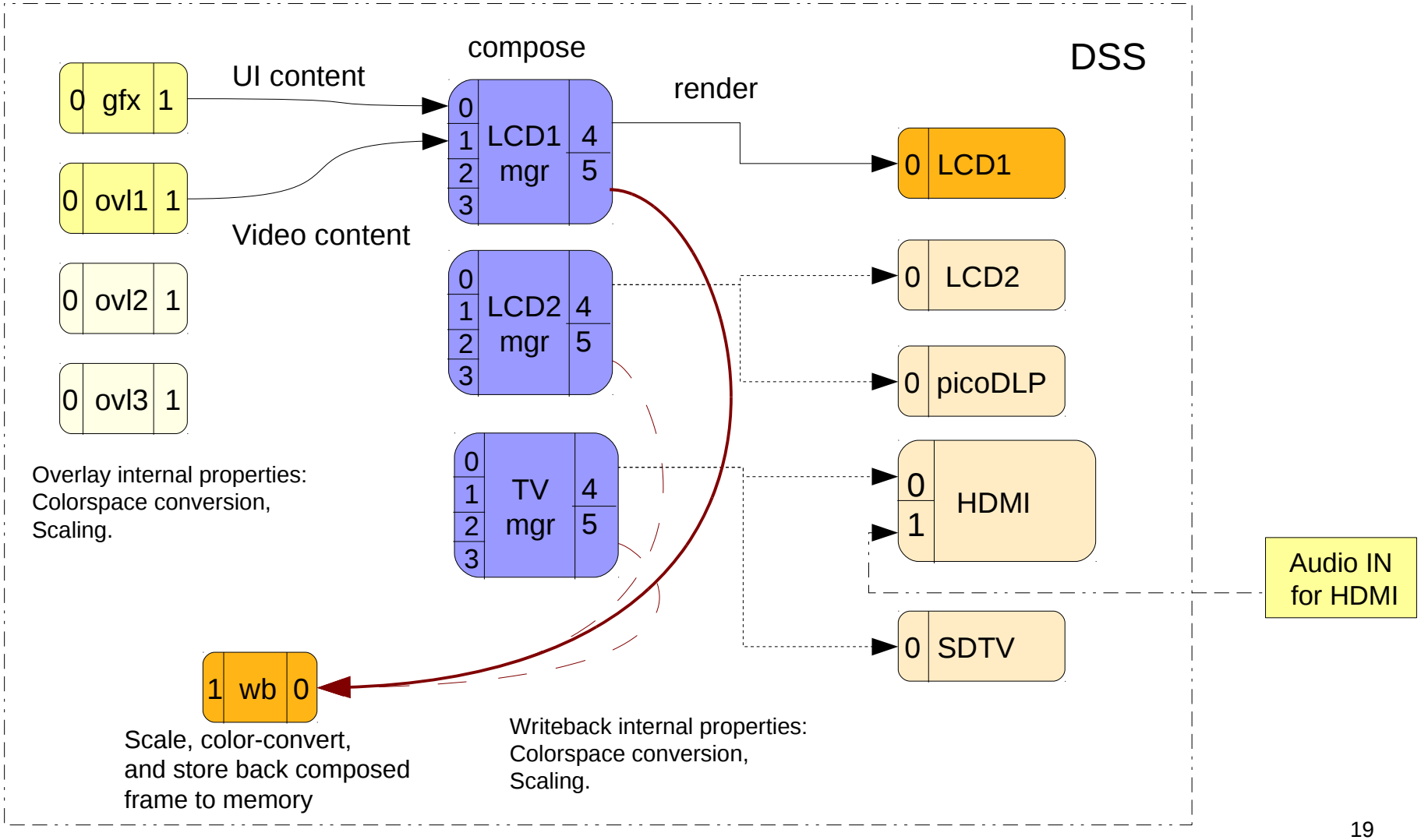
MC – DSS: Writeback (mem-2-mem)



MC – DSS: Writeback (mem-2-mem)

```
struct media_link_desc link;

link.source.entity = OMAP_DSS_ENTITY_OVERLAY0;
link.source.index = 1;
link.sink.entity = OMAP_DSS_ENTITY_MGR_LCD1;
link.sink.index = 0;
link.flags = MEDIA_LNK_FL_ENABLED;
ioctl(fd, MEDIA_IOC_SETUP_LINK, &link);

link.source.entity = OMAP_DSS_ENTITY_MGR_LCD1;
link.source.index = 4;
link.sink.entity = OMAP_DSS_ENTITY_PANEL_LCD1;
link.sink.index = 0;
link.flags = MEDIA_LNK_FL_ENABLED;
ioctl(fd, MEDIA_IOC_SETUP_LINK, &link);

/* Writeback path */
link.source.entity = OMAP_DSS_ENTITY_OVERLAY1;
link.source.index = 1;
link.sink.entity = OMAP_DSS_ENTITY_WRITEBACK;
link.sink.index = 0;
link.flags = MEDIA_LNK_FL_ENABLED;
ioctl(fd, MEDIA_IOC_SETUP_LINK, &link);
```

MC – DSS: Writeback (capture)



MC – DSS: Writeback (capture)

```
struct media_link_desc link;

link.source.entity = OMAP_DSS_ENTITY_OVERLAY0;
link.source.index = 1;
link.sink.entity = OMAP_DSS_ENTITY_MGR_LCD1;
link.sink.index = 0;
link.flags = MEDIA_LNK_FL_ENABLED;
ioctl(fd, MEDIA_IOC_SETUP_LINK, &link);

link.source.entity = OMAP_DSS_ENTITY_OVERLAY1;
link.source.index = 1;
link.sink.entity = OMAP_DSS_ENTITY_MGR_LCD1;
link.sink.index = 1;
link.flags = MEDIA_LNK_FL_ENABLED;
ioctl(fd, MEDIA_IOC_SETUP_LINK, &link);

link.source.entity = OMAP_DSS_ENTITY_MGR_LCD1;
link.source.index = 4;
link.sink.entity = OMAP_DSS_ENTITY_PANEL_LCD1;
link.sink.index = 0;
link.flags = MEDIA_LNK_FL_ENABLED;
ioctl(fd, MEDIA_IOC_SETUP_LINK, &link);

/* Writeback path */
link.source.entity = OMAP_DSS_ENTITY_MGR_LCD1;
link.source.index = 5;
link.sink.entity = OMAP_DSS_ENTITY_WRITEBACK;
link.sink.index = 0;
link.flags = MEDIA_LNK_FL_ENABLED;
ioctl(fd, MEDIA_IOC_SETUP_LINK, &link);
```

Limitations in moving to MC + V4L2

- DSS User management
  - FB and V4L2 are still two 'disjoint' users
    - Unless, we find a way of working with FB and V4L2 together?
      - One proposal to explore is the option of building an FB on top of a V4L device.
        - » This approach may not work in case of inter-op required with DRM (use case discussed in a later slide)
- Integration / inter-op with GPU
  - No support available in V4L2 as of now
- Dependent subsystem needs to be MC-compliant
  - Audio driver would need to 'adapt' to MC to be able to use Audio MC entity.
  - SDRAM based DSI direct display (bypassing overlays)

Direct Rendering Manager (DRM) - Overview

Slides on DRM attributed to Rob Clark – rob@ti.com

DRM Overview

- The modern Linux (and *BSD) display driver framework
- The kernel component of DRI (Direct Rendering Infrastructure)
- But not just DRI:
  - Hotplug
  - XRandR/KMS
  - EDID parsing
    - Discussions have already been started to move out EDID parsing to make it non-DRM dependent
- Kernel Mode Setting (KMS)
  - Setting screen resolution/timings
  - Hotplug, and retrieving EDID
  - Multi-display configurations

DRM – Nice features

- Maps well to DSS
  - drm_framebuffer → a piece of memory
  - drm_crtc → overlay
  - drm_encoder → manager
  - drm_connector → omap_dss_device (represents dss panel)
- Separation of buffers from overlays
  - Lets userspace dynamically reconnect buffers to overlays
  - This lets us implement single buffer spanning multiple displays
  - In userspace we can implement virtual overlays when we run out of video pipes
  - And if we use this for video too we can dynamically switch between GPU (GL) and DSS composition
- Security is inherent in DRM

DRM – gaps

- No clear way of handling Writeback
  - Or any new components upcoming, since the component 'types' are predefined [crtc, connector, framebuffer, encoder]
- Audio interface doesn't seem to be too clear
  - HDMI – Aud+Vid, future DSI with Audio?

Inter-op need between MC and DRM

Need for inter-op between MC and DRM

- Usecase where there is a fallback capability available:
  - We have 1 gfx pipe, and 3 video pipes available.
  - To render 4 surfaces, this is sufficient to use via V4L2 [MCF] and FB.
  - What if we need to render 7 surfaces?
    - One solution is to use 3 video pipes for three video surfaces, and use a 2D/3D Graphics Accelerator [like SGX540 in OMAP4] for composing remaining 4 surfaces which can then be rendered via 1 gfx pipe.
  - DRM framework is more suited for Graphics accelerator kind of devices.
- V4L2 camera, DRM-supported GPU.
  - Buffer sharing
    - V4L2 supports USERPTR, so if some way is available for DRM to share buffers, it would help.

Summary

- Media Controller seems quite adaptable to display sub systems in general.
  - Flexibility – in linking, defining new components, using different frameworks.
  - Re configurability.
- Still needs to improve on certain key areas – buffer-sharing, inter-op with the FB world (DRM), inter-working with GPUs

Acknowledgements

- Hans Verkuil
- Laurent Pinchart
- Rob Clark
- TI LDC display team :)

References

- Laurent Pinchart's MC presentation at V4L2 summit
- 3430 public Technical Reference Manual
- 4430 public Technical Reference Manual

Q and A?

Thank You!

Backup

Acronyms

- **DSI**: **D**isplay **S**erial **I**nterface: a MIPI standard interface for serial display panels.
- **DPI**: **D**isplay **P**ixel **I**nterface: a MIPI standard interface for parallel display panels.
- **RFBI: R**emote **F**rame **B**uffer **I**nterface: a MIPI standard interface for parallel display panels supporting remote frame buffer mechanism.
- **HDMI**: **H**igh **D**efinition **M**ultimedia **I**nterface: a standard compact AV interface to transmit uncompressed digital data.
- **VENC**: **V**ideo **ENC**oder: standard interface to display content on standard definition TVs, using TV formats like NTSC / PAL.

Features of OMAP2+ DSS

Feature	OMAP2430	OMAP3430	OMAP4430
GFX overlay	1	1	1
VID overlay	2	2	3
WB overlay	-	-	1
LCD overlay manager	1	1	2
TV overlay manager	1	1	1
Rotation (per overlay)	Via VRFB*	Via VRFB*	Via TILER*

OMAP3/4 support ARGB color formats.
OMAP4 additionally has native support for NV12, configurable Z-order.

Overlay perspective

GFX overlay supports CLUT/gamma table, anti-flicker, replication logic – primarily meant for UI.

VID overlays support Multiple color formats(RGB, YUV, NV12 based on OMAP version), colorspace conversion, independent horizontal and vertical polyphase filter scaling, and VC-1 range mapping.

WB overlay, in

mem-to-mem mode: colorspace conversion, rescaling, compositing processes.
Capture mode: allows composited encoded pixel data to be stored back into memory.

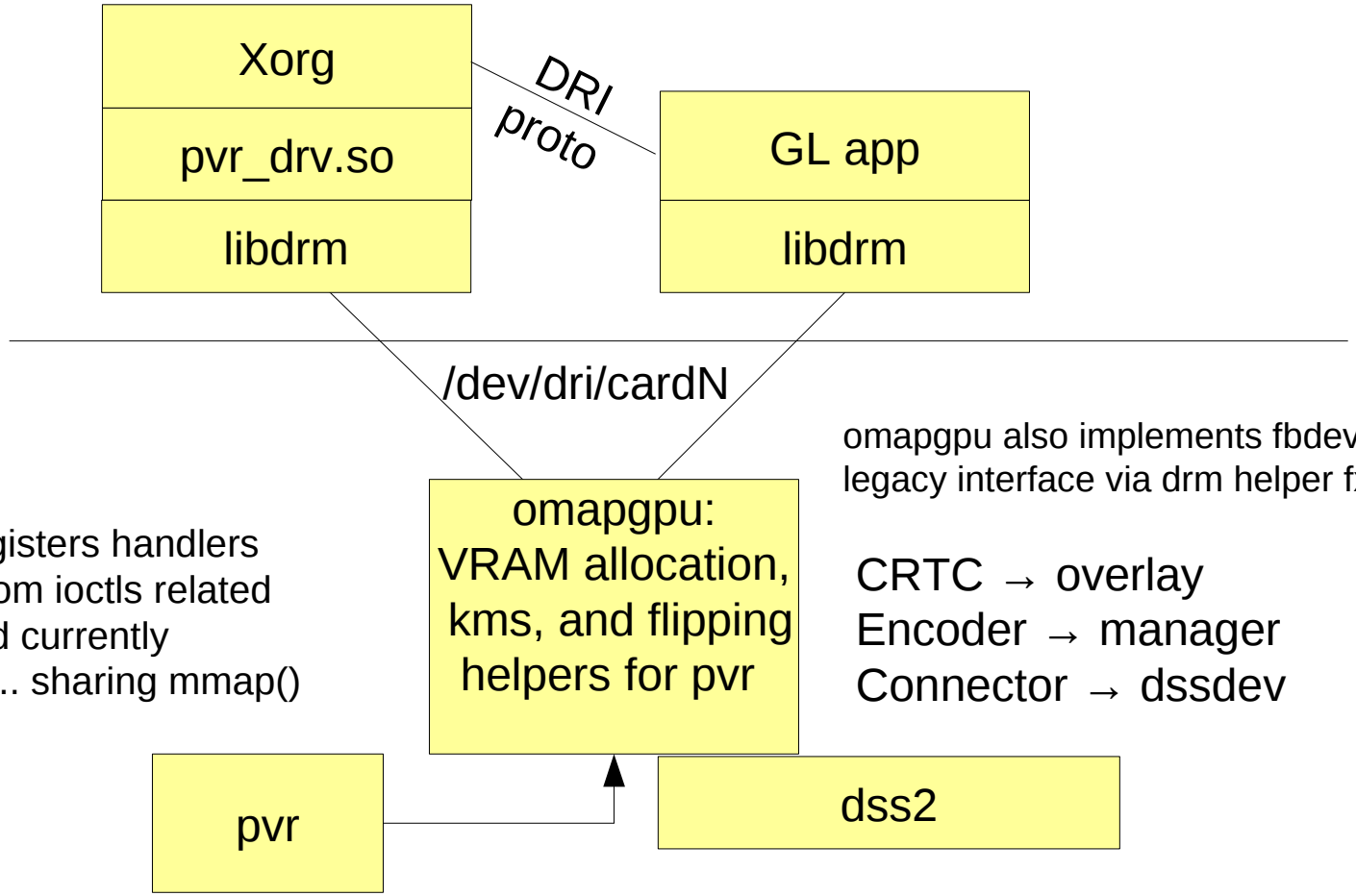
Overlay perspective

Each overlay supports: overlaying [fixed-order for OMAP2/3, z-order per pipeline for OMAP4], transparency color-keys [source and destination] and global and pixel alpha blending [OMAP3/4], Gamma correction, Sync buffers.

LCD overlays additionally support Active / Passive matrix dithering, color phase rotation.

*** Read more about Virtual Rotated Frame Buffer (VRFB) and TILER in the 3430 public TRM and 4430 public TRM respectively**

Current architecture of Rob's omapgpu driver



omapgpu also implements fbdev legacy interface via drm helper fxns

CRTC → overlay
Encoder → manager
Connector → dssdev

pvr 3d driver registers handlers for a set of custom ioctls related to gfx accel (and currently mmap() handler.. sharing mmap() is TBD)

Security – Why does it matter?

- Say you had a multi-user system with user-a and user-b both are in “video” group
  - So both have permissions to open /dev/fbN, /dev/videoN..
- Currently user-a is logged in and owns the display
  - Normally x-server would enforce security so user-b could not pop up windows
- But current v4l2 driver interface (plus sysfs) lets user-b subvert this
  - user-b opens /dev/video1 and displays a fake login screen and tricks user-a to type password, for example
- DRM/DRI, while it allows for rendering directly from other processes than the x-server, it still enforces a security model so the client process must authenticate with the x-server before being granted direct rendering access.

Note: while DRM/DRI originated in the x-server world, Android also has a display server (SurfaceFlinger); while the security model maybe isn't much of a concern in Android, the DRM based gfx stack is starting to be used in Android for alignment with (generic) linux.