Linux DRM: New Picture Processing API

Marek Szyprowski
m.szyprowski@samsung.com
Samsung R&D Institute Poland

SAMSUNG

Agenda

- Quick Introduction to Linux DRM
- A few words on atomic KMS API
- Exynos DRM IPP subsystem
- New API proposal
- Some code examples
- Summary

SAMSUNG

Linux DRM subsystem

- DRM = Direct Rendering Manager

- Main framework for various display related drivers
  - Full-blown GPUs (Intel, AMD, Nvidia)
  - Simple graphics modules found in embedded SoCs
  - Access to hardware (IOCTLs) from user space
  - GEM (buffers)
  - KMS
  - libdrm

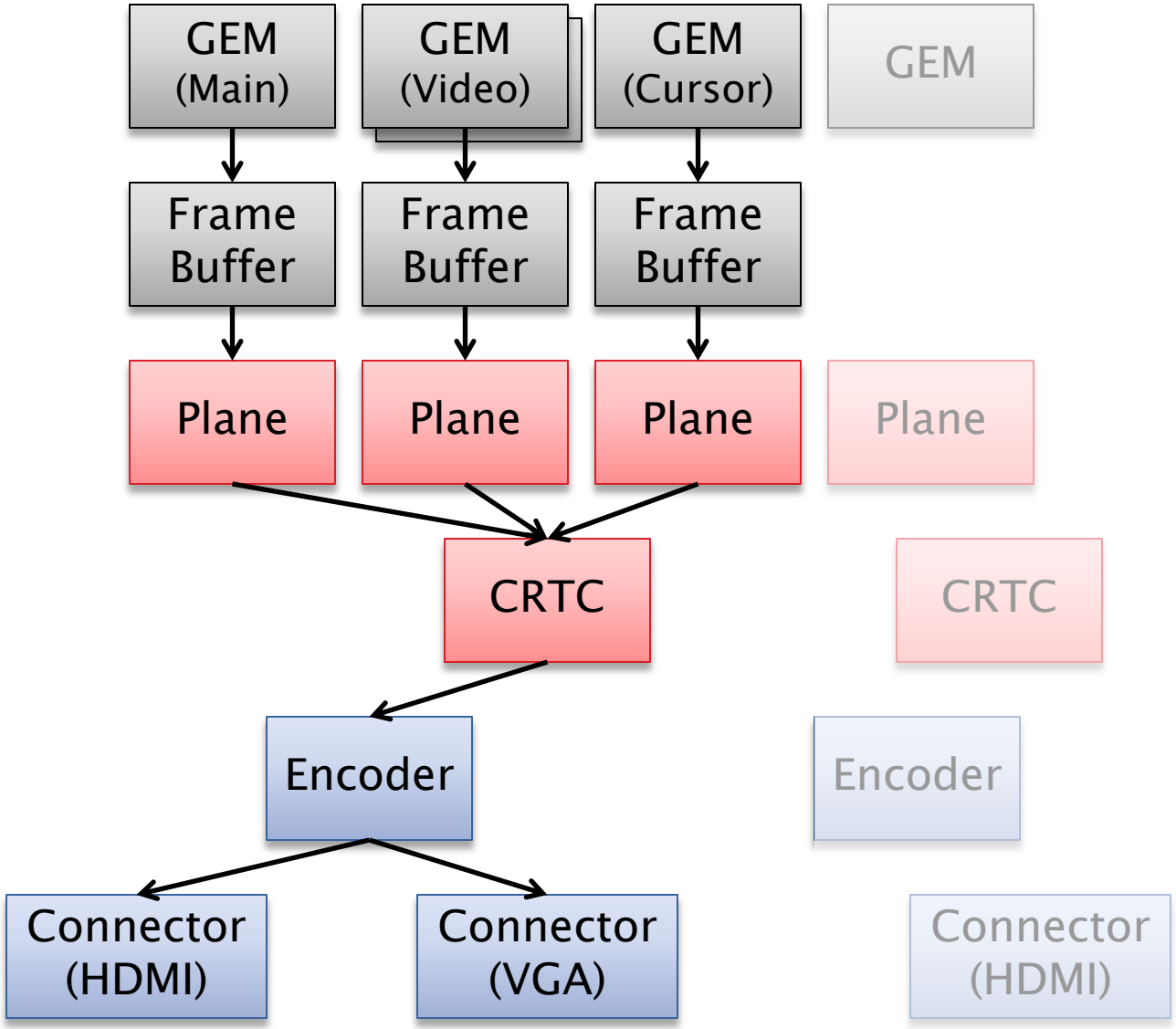
Kernel Mode Setting

- KMS = Kernel Mode Setting
- Generic abstraction of the hardware
  - CRTCs, Connectors, Encoders, Planes, ...
  - Generic, hardware independent IOCTLs
- Configure given display mode on a display pipe-line
  - Mode: resolution, pixel format, display buffer
  - Display pipe-line: CRTC, encoder, connector, ...
- KMS provide emulation of legacy FBDev API
- Together with dumb framebuffers allows to create hardware independed userspace application

Quick introduction to KMS Objects (1/2)

- GEM = memory buffer
- Frame Buffer = **GEM** + (format, width/height, ...)
- Plane = Hardware for scanning out **Frame Buffer**
- CRTC = Catode Ray Tube Controller (historical), nowadays hardware for mixing/blending **Planes**
- Encoder = Generates signal from the **CRTC** output
- Connector = Routes signal from **Encoder** to external world (i.e. Display Panel)

Quick introduction to KMS Objects (2/2)



DRM objects and properties

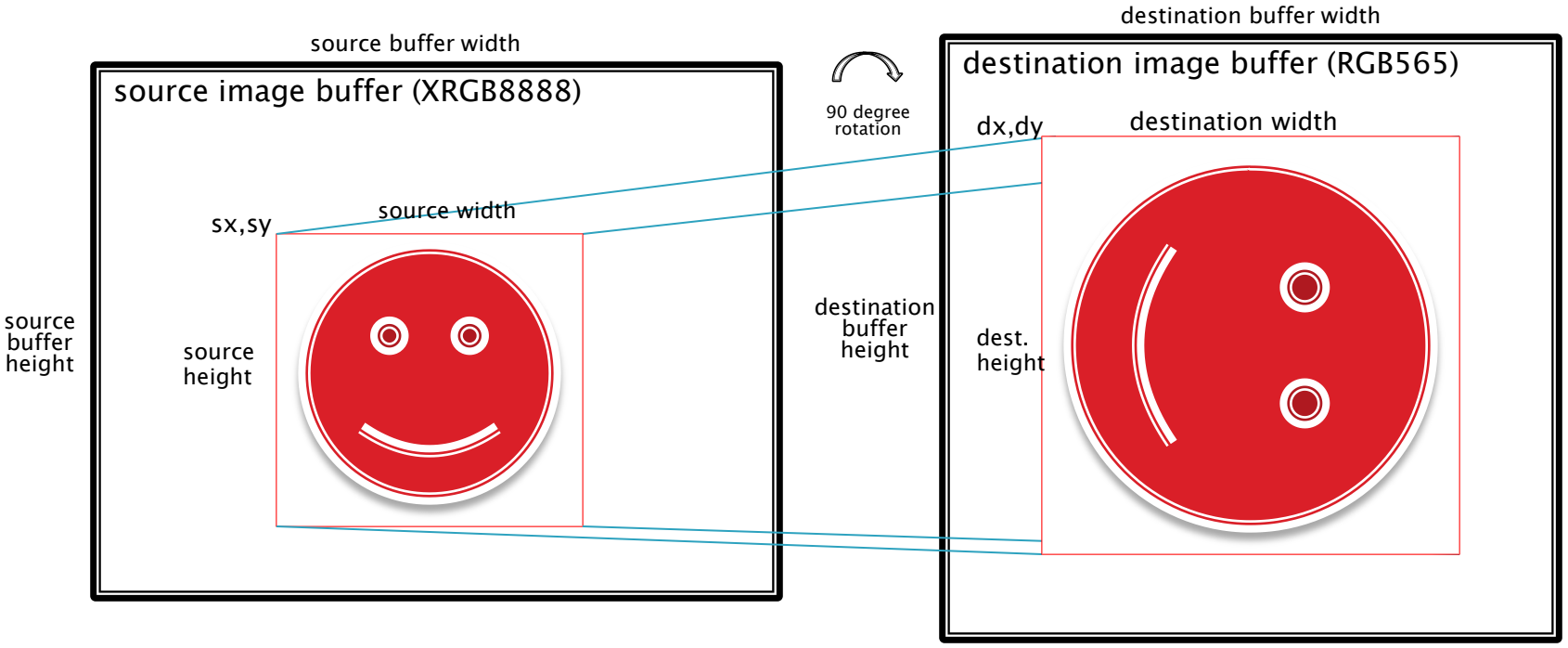
- DRM Object
  - Unique ID
  - Type (CRTC, Connector, Encoder, Plane, FB, Blob, ...)
  - Set of Properties
- DRM Property
  - Unique ID
  - Type:
    - Range (Integer)
    - Enum (Enumerated type with text strings)
    - Blob (Binary data)
    - Bitmask (Bitmask of enumerated types)
    - Object (other DRM Object ID)
  - Value stored on 64 bits

Atomic KMS API

- Atomic KMS API – do all in a single ioctl:
  - Reconfigure display pipeline
  - Update multiple scanout buffers (planes) on page flip
- Enabled in Linux v4.2
- Drivers most drivers already converted to atomic API
- Basic idea
  - Use objects and properties
  - State is a set of properties assigned to given objects

Exynos Image Post Processing API (1/2)

- IPP = Image Post-Processing
- Exynos DRM custom extension
- Memory-to-memory operation:
  - image scaling, croping, colorspace conversion, rotation and flip



Exynos Image Post Processing API (2/2)

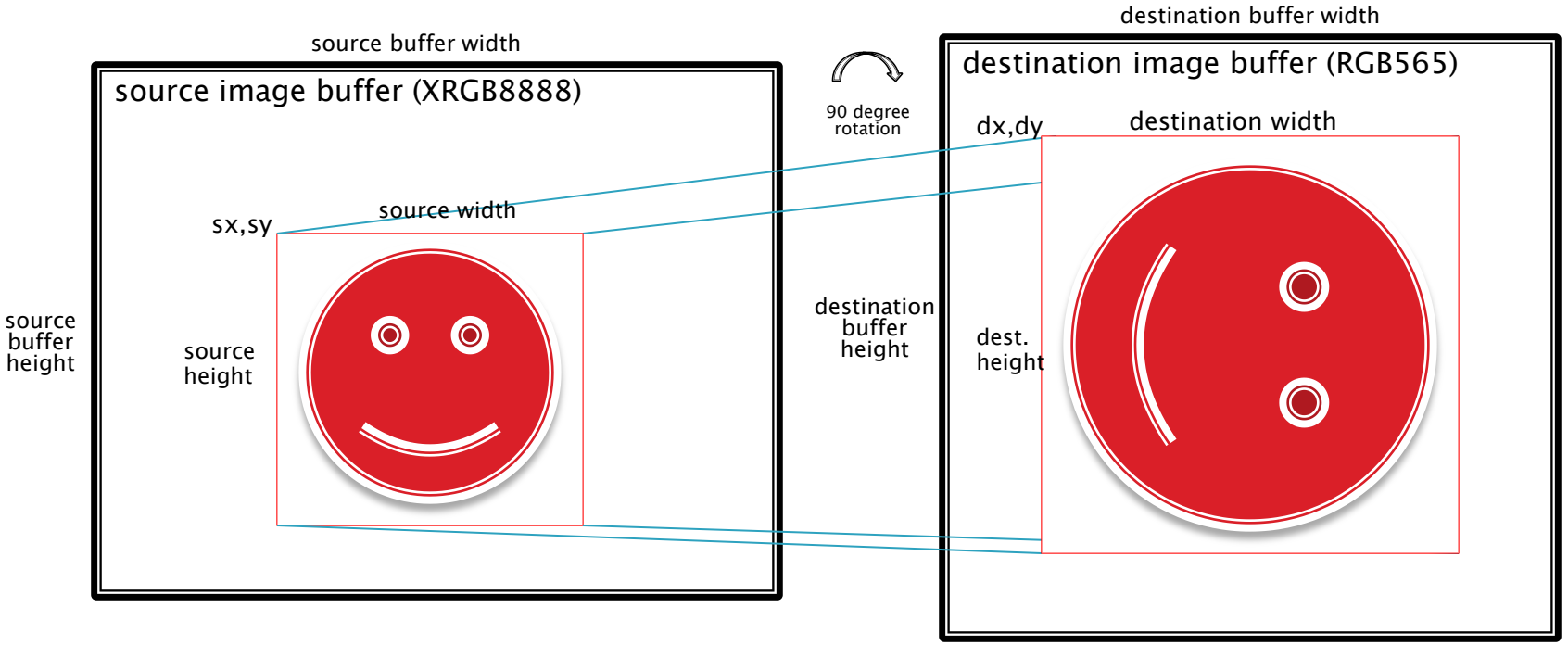
- Introduced in Linux v3.8 (early 2013)
- Userspace API heavily based on internal way of operation of Exynos image processing modules
- Userspace API hard to understand, prone to errors
- Additional modes of operation: writeback and output
  - Not fully implemented...

- Basic idea: rewrite and do it right!

SAMSUNG

Image processing operation

- Single picture processing operation:
  - Source image buffer + operation area (x, y, width, height)
  - Destination image buffer + operation area (x, y, width, height)
  - Optional transformation (rotation and flip)



IPP API Rewrite – assumptions

- Picture processing API
  - Support for memory-to-memory operations:
    - Image scaling, croping, colorspace conversion, rotation and flip
  - Support for query capabilities
  - Hide details of underlying hardware module
  - Follow convention of DRM/KMS
    - Use DRM objects and properties
  - Allow to write hardware independent application code
  - Be ready for future extensions

Image buffer

- GEM object
  - Size in bytes, unspecified format
- DRM Frame Buffer object
  - GEM + offset + size
  - pixel format
  - width, height, stride
  - optional support for multi-buffer formats (i.e. NV12)

Image processing operation

- [RFC 0/2] New feature: Framebuffer processors
  - http://www.spinics.net/lists/linux-samsung-soc/msg54810.html
- Basic idea: introduce new objects
  - **Frame Buffer Processors**
- Image processing operation is defined by properties
- Heavily inspired by DRM Atomic KMS API

Technical Detail

- Simple user interface – 3 new ioctls:
  - DRM_IOCTL_MODE_GETFBPROCRESOURCES
    - Get number of FrameBuffer Processor Objects and their IDs
  - DRM_IOCTL_MODE_GETFBPROC
    - Get capabilities of given FrameBuffer Processor
  - DRM_IOCTL_MODE_FBPROC
    - Perform operation on given FrameBuffer Processor
- Additional 2 standard DRM ioctls needed:
  - DRM_IOCTL_MODE_OBJ_GETPROPERTIES
    - Get array of property IDs for given DRM object
  - DRM_IOCTL_MODE_GETPROPERTY
    - Get parameters of given property

SAMSUNG

DRM_IOCTL_MODE_GETFBPROCRESOURCES

- Get number of FrameBuffer Processor Objects and their Ids
- First call – to get total number of Objects
- Second call – to fill array of Object IDs
- Arguments:

```
struct drm_mode_get_fbproc_res {
        __u64 fbproc_id_ptr;
        __u32 count_fbprocs;
};
```

DRM_IOCTL_MODE_GETFBPROC (1/2)

- Get capabilities of given FrameBuffer Processor
- First call – to get total number of supported formats
- Second call – to fill arrays of supported formats
- Arguments:

```
struct drm_mode_get_fbproc {
        __u32 fbproc_id;
        __u32 capabilities;

        __u32 src_format_count;
        __u32 dst_format_count;
        __u64 src_format_type_ptr;
        __u64 dst_format_type_ptr;
};
```

SAMSUNG

DRM_IOCTL_MODE_GETFBPROC (2/2)

- Capabilities (almost self-explanatory):
  - DRM_FBPROC_CAP_CROP
  - DRM_FBPROC_CAP_ROTATE
  - DRM_FBPROC_CAP_SCALE
  - DRM_FBPROC_CAP_CONVERT
  - DRM_FBPROC_CAP_FB_MODIFIERS
- Supported source and destination formats
  - Standard DRM fourcc values (i.e. DRM_FORMAT_XRGB8888)

DRM_IOCTL_MODE_FBPROC (1/2)

- Perform operation on given FrameBuffer Processor
- Flags:
  - DRM_MODE_FBPROC_EVENT – generate DRM event with user_data on finish
  - DRM_MODE_FBPROC_TEST_ONLY – check parameters
  - DRM_MODE_FBPROC_NONBLOCK – asynchronous call
- Arguments:

```
struct drm_mode_fbproc {
        __u32 fbproc_id;
        __u32 flags;
        __u32 count_props;
        __u64 props_ptr;
        __u64 prop_values_ptr;
        __u64 reserved;
        __u64 user_data;
};
```

DRM_IOCTL_MODE_FBPROC (2/2)

- Property set:
  - Number of properties (count_props)
  - Array of property IDs (props_ptr)
  - Array of property values (prop_values_ptr)
- Arguments:

```
struct drm_mode_fbproc {
        __u32 fbproc_id;
        __u32 flags;
        __u32 count_props;
        __u64 props_ptr;
        __u64 prop_values_ptr;
        __u64 reserved;
        __u64 user_data;
};
```

SAMSUNG

FB Processor: properties of operation

- Single picture processing operation as DRM properties:
  - SRC_FB_ID (FrameBuffer object ID),
  - SRC_X (16.16 integer), SRC_Y (16.16 integer),
  - SRC_W (16.16 integer), SRC_H (16.16 integer),

  - DST_FB_ID (FrameBuffer object ID),
  - DST_X (16.16 integer), DST_Y (16.16 integer),
  - DST_W (16.16 integer), DST_H (16.16 integer),

  - Optional ROTATION (rotation enum)

User space API – libdrm (1/3)

- DRM_IOCTL_MODE_GETFBPROCRESOURCES
  - drmModePlaneResPtr **drmModeGetFBProcResources**(int fd)
  - void **drmModeFreeFBProcResources**(drmModePlaneResPtr ptr)
  - Result:

```
struct _drmModeFBProcRes {
        uint32_t count_fbprocs;
        uint32_t *fbprocs;
}
```

User space API – libdrm (2/3)

- DRM_IOCTL_MODE_GETFBPROC
  - drmModeFBProcPtr **drmModeGetFBProcResources**(int fd, uint32_t id)
  - void **drmModeFreeFBProcResources**(drmModeFBProcPtr ptr)
  - Result:

```
struct _drmModeFBProc {
        uint32_t fbproc_id;
        uint32_t capabilities;
        uint32_t src_format_count;
        uint32_t dst_format_count;
        uint32_t *src_formats;
        uint32_t *dst_formats;
}
```

SAMSUNG

User space API – libdrm (3/3)

- DRM_IOCTL_MODE_FBPROC
  - int **drmModeFBProcReqCommit**(int fd, uint32_t fbproc_id,
    drmModeFBProcReqPtr req, uint32_t flags,
    void *user_data)
  - drmModeFBProcReqPtr **drmModeFBProcReqAlloc**(void)
  - void **drmModeFBProcReqFree**(drmModeFBProcReqPtr req)
  - int **drmModeFBProcReqAddProperty**(drmModeFBProcReqPtr req,
    uint32_t property_id, uint64_t value);
  - int **drmModeFBProcReqGetCursor**(drmModeFBProcReqPtr req)
  - void **drmModeFBProcReqSetCursor**(drmModeFBProcReqPtr req,
    int cursor)

Example application code (1/4)

```
int process_fb(int fd, int rotation, int src_fb_id, int sx, int sy,
        int sw, int sh, int dst_fb_id, int dx, int dy, int dw, int dh)
{
        drmModeObjectPropertiesPtr props;
        drmModeFBProcResPtr res;
        drmModeFBProcPtr fbproc;
        drmModeFBProcReqPtr req;
        uint32_t id, pid;

        res = drmModeGetFBProcResources(fd);

        if (res->count_fbprocs == 0) {
                printf("no fbproc object found\n");
                return 0;
        }

        id = res->fbprocs[0];
        drmModeFreeFBProcResources(res);

        fbproc = drmModeGetFBProc(fd, id);

        if (!(fbproc->capabilities & DRM_FBPROC_CAP_ROTATE)) {
                printf("fbproc has no rotation capability\n");
                return 0;
        }
```

Example application code (2/4)

```
req = drmModeFBProcReqAlloc();

props = drmModeObjectGetProperties(fd, id, DRM_MODE_OBJECT_FBPROC);

pid = get_prop_id(fd, props, "SRC_FB_ID");
drmModeFBProcReqAddProperty(req, pid, src_fb_id);

pid = get_prop_id(fd, props, "SRC_X");
drmModeFBProcReqAddProperty(req, pid, sx << 16);

pid = get_prop_id(fd, props, "SRC_Y");
drmModeFBProcReqAddProperty(req, pid, sy << 16);

pid = get_prop_id(fd, props, "SRC_W");
drmModeFBProcReqAddProperty(req, pid, sw << 16);

pid = get_prop_id(fd, props, "SRC_H");
drmModeFBProcReqAddProperty(req, pid, sh << 16);

pid = get_prop_id(fd, props, "DST_FB_ID");
drmModeFBProcReqAddProperty(req, pid, dst_fb_id);

pid = get_prop_id(fd, props, "DST_X");
drmModeFBProcReqAddProperty(req, pid, dx << 16);
```

Example application code (3/4)

```
        pid = get_prop_id(fd, props, "DST_Y");
        drmModeFBProcReqAddProperty(req, pid, dy << 16);

        pid = get_prop_id(fd, props, "DST_W");
        drmModeFBProcReqAddProperty(req, pid, dw << 16);

        pid = get_prop_id(fd, props, "DST_H");
        drmModeFBProcReqAddProperty(req, pid, dh << 16);

        pid = get_prop_id(fd, props, "rotation");
        drmModeFBProcReqAddProperty(req, pid, rotation);

        drmModeFreeObjectProperties(props);

        ret = drmModeFBProcReqCommit(fd, id, req, 0, NULL);
        if (ret) {
                printf("failed to commit fbproc request: %d\n", ret);
                return 0;
        }

        drmModeFBProcReqFree(req);

        return 1;
}
```

SAMSUNG

Example application code (4/4)

```
uint32_t get_prop_id(int fd, drmModeObjectPropertiesPtr props, char *name)
{
        drmModePropertyPtr p;
        uint32_t i, prop_id = 0;

        for (i = 0; !prop_id && i < props->count_props; i++) {
                p = drmModeGetProperty(fd, props->props[i]);
                if (!strcmp(p->name, name))
                        prop_id = p->prop_id;
                drmModeFreeProperty(p);
        }
        return prop_id;
}
```

Summary

- Patches has not been merged to mainline yet
- No positive feedback from DRM maintainers
- This interface will still be limited to Exynos DRM

SAMSUNG

Q&A

Thank you!

Any questions?

SAMSUNG

References

- Marek Szyprowski: „[RFC 0/2] New feature: Framebuffer processors” patchset http://www.spinics.net/lists/linux-samsung-soc/msg54810.html

- Daniel Vetter: „Atomic mode setting design overview, part 1” https://lwn.net/Articles/653071/
- Daniel Vetter: „Atomic mode setting design overview, part 2” https://lwn.net/Articles/653466/

- Marek Szyprowski: Simple atomic KMS userspace example: https://git.linaro.org/people/marek.szyprowski/atomictest.git

SAMSUNG