There are no dragons: Porting fbdev drivers to DRM/KMS

Javier Martinez Canillas

$ whoami

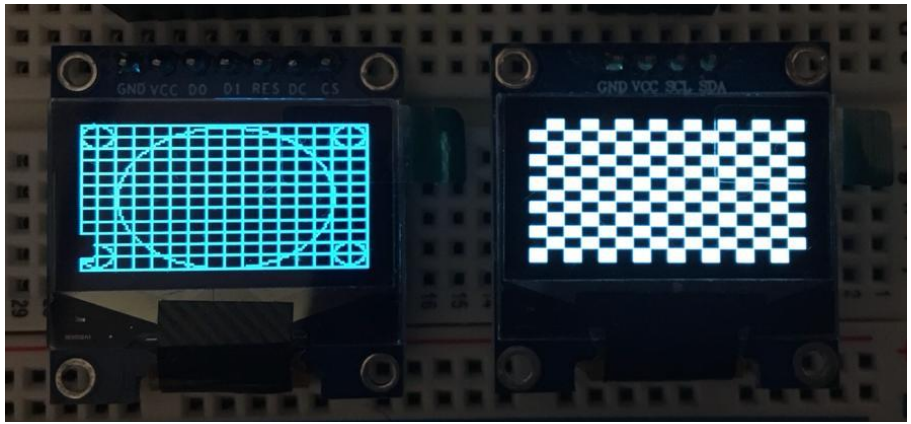
- Linux and Fedora developer
- Principal Software Engineer @ Red Hat
- Working on the In-Vehicle Operating System
- Not a graphics expert but learning about it :)

Backstory

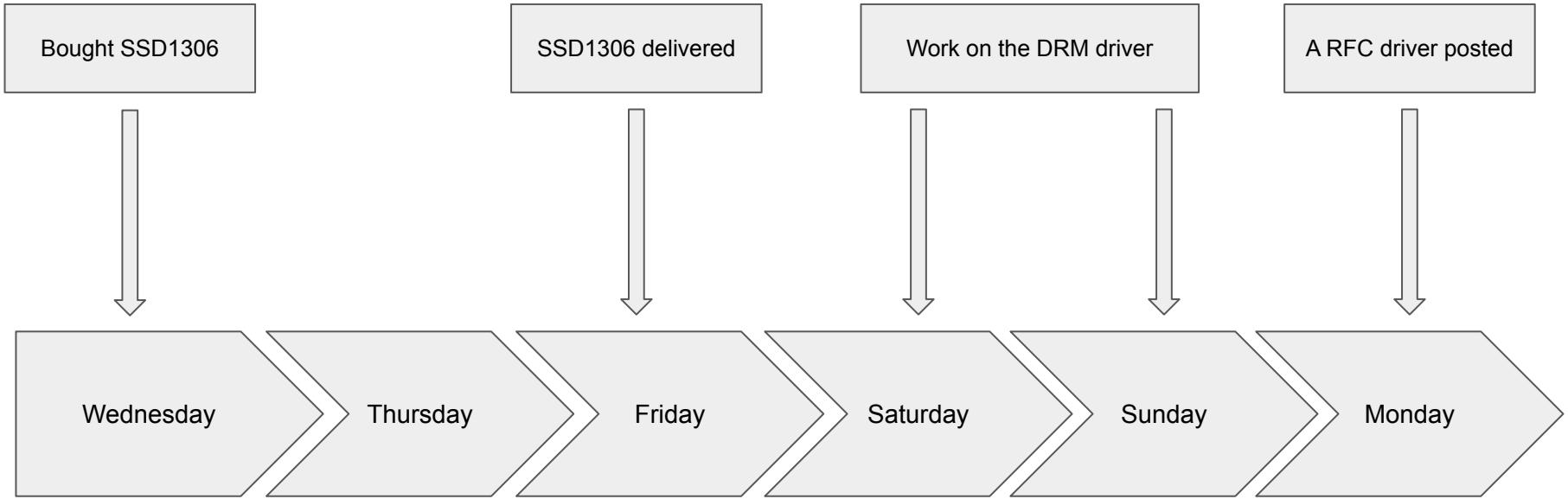
- Last year moved to a team whose responsibilities involved graphics work.
- Not being an expert decided to delve into DRM and learn its internals.
- Subscribed to dri-devel and started reading patches and conversations.
- There were some discussions about gaps between fbdev and DRM.
- SSD130X was mentioned as devices supported by fbdev but not by DRM.
- Thought that porting these drivers would be a good way to learn about DRM.

Solomon SSD130x chips

- Monochrome OLED graphic display controllers.
- Supports parallel, I2C and SPI interfaces.
- Existing fbdev drivers: ssd1307fb (I2C) and fb_ssd130{5,6} (SPI).
- Ported to a DRM ssd130x core driver plus ssd130x-{i2c,spi} transport drivers.



Was it easy to port? Spoiler alert: yes, it was



Timeline

Why porting was so easy?

- Many example drivers for simple display devices in drivers/gpu/drm/tiny/ dir.
- DRM has a lot of helpers for drivers authors to avoid boilerplate code.
- Probably one of the subsystems with the best documentation.
- The DRM community is just superb and very helpful.

Process followed to port the drivers

- Use the existing fbdev drivers to have a baseline for the DRM driver testing.
  - e.g: drivers/video/fbdev/ssd1307fb.c
- Check if there is a DRM driver for a similar device to be used as a reference.
  - e.g: drivers/gpu/drm/tiny/repaper.c
- Figure out if there is common code to factor out and add as a DRM helper.
  - e.g: drm_fb_xrgb8888_to_mono() in drivers/gpu/drm/drm_format_helper.c
- Make sure the fbdev programs can run correctly using DRM fbdev emulation.
  - e.g: https://git.kernel.org/pub/scm/linux/kernel/git/geert/fbtest.git
- But also test using DRM/KMS programs and fbcon!

Attempt #1: drivers/gpu/drm/tiny/ssd130x.c

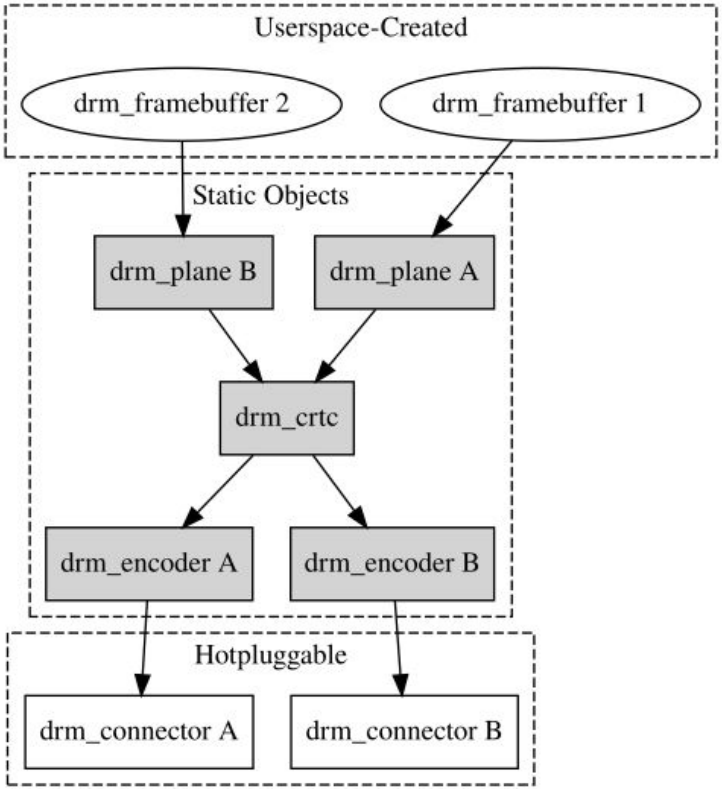
- Added ssd130x as a “tiny” driver.
- Tiny drivers are for simple devices and implemented in a single source file.
- It wouldn’t be suitable for ssd130x in the long term once SPI support added.
    - Since the driver would had be implemented in more than one source file.

Attempt #2: drivers/gpu/drm/solomon/ssd130x*.c

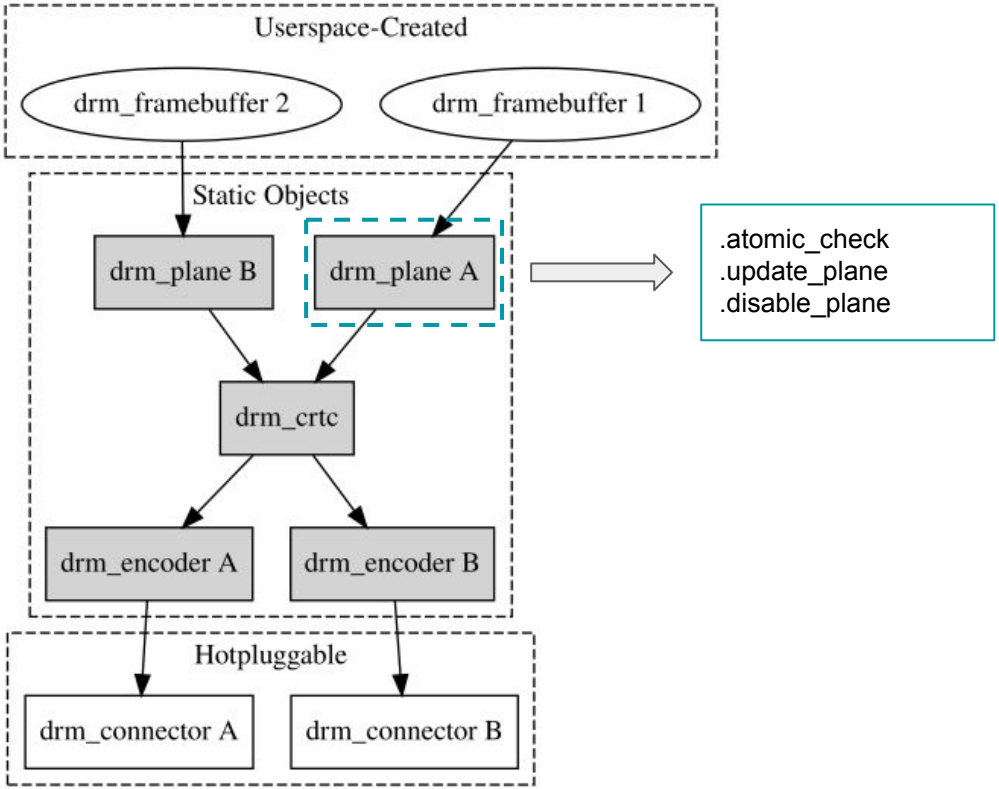
- Implemented as a core driver and separate I2C and SPI transport drivers.
  - drivers/gpu/drm/solomon/ssd130x.c
  - drivers/gpu/drm/solomon/ssd130x-i2c.c
  - drivers/gpu/drm/solomon/ssd130x-spi.c

KMS display pipeline overview

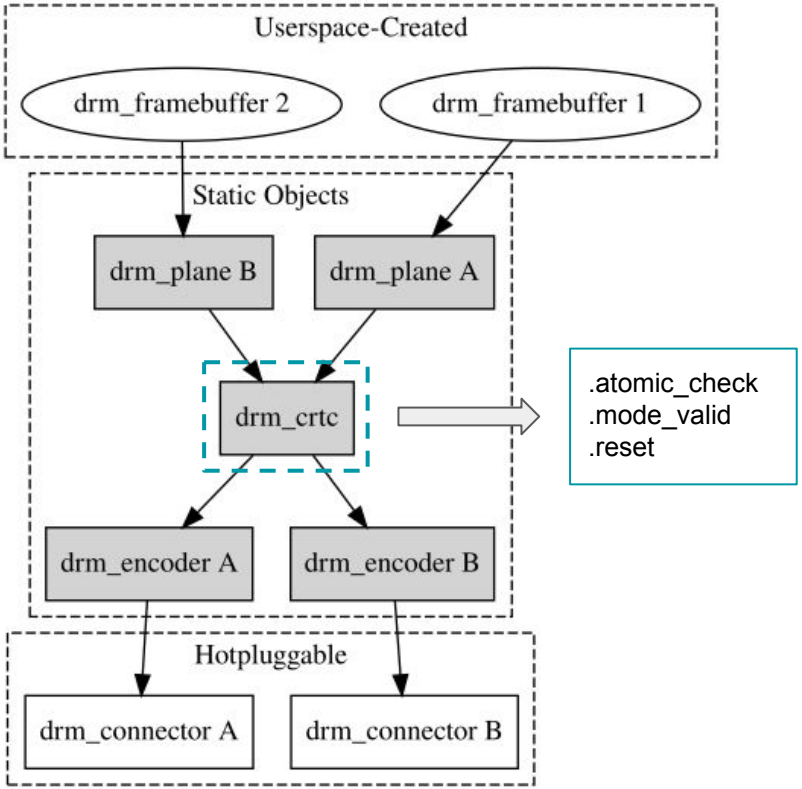
KMS display pipeline overview



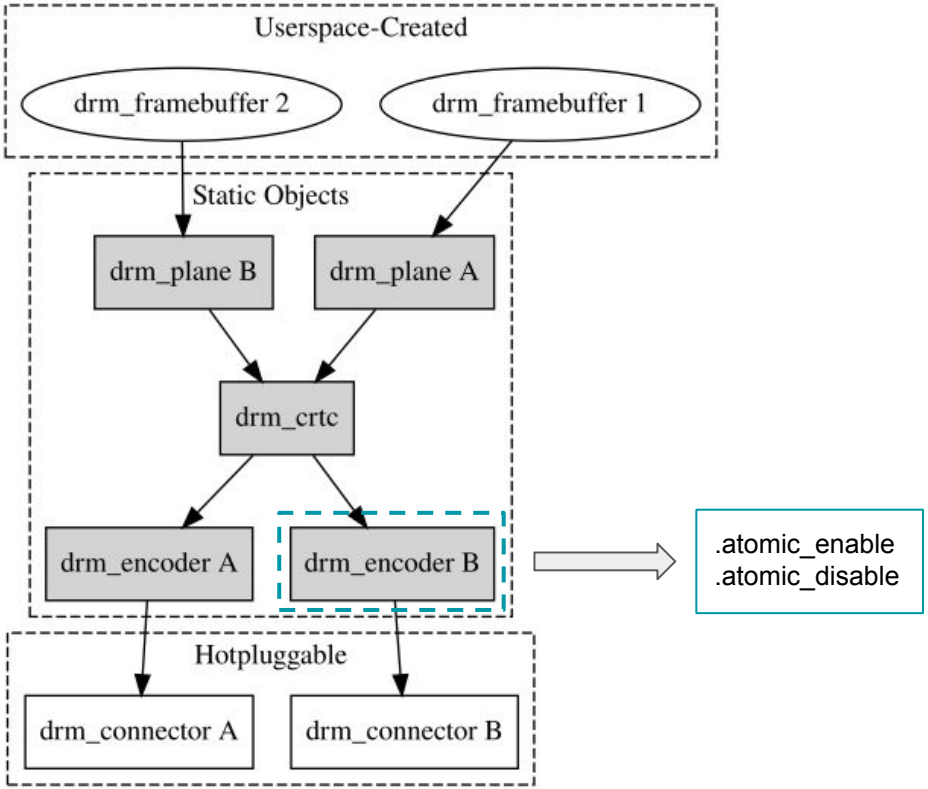
DRM plane


Userspace-Created
drm_framebuffer 2
drm_framebuffer 1
Static Objects
drm_plane B
drm_plane A
drm_crtc
drm_encoder A
drm_encoder B
Hotpluggable
drm_connector A
drm_connector B
.atomic_check
.update_plane
.disable_plane

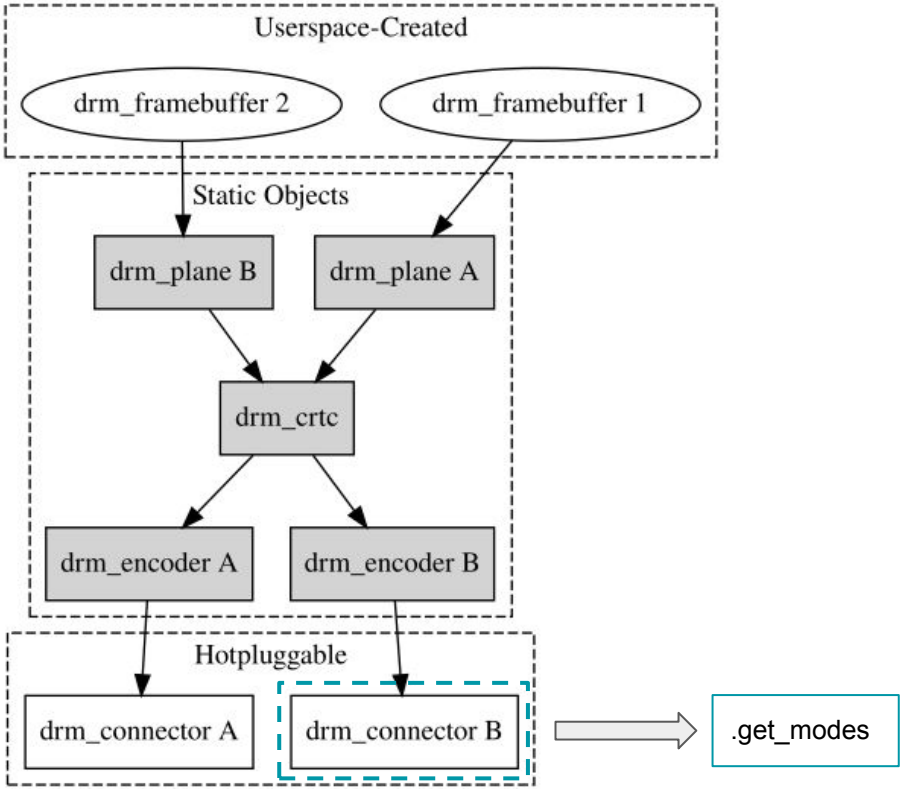
DRM crtc


Userspace-Created
drm_framebuffer 2
drm_framebuffer 1
Static Objects
drm_plane B
drm_plane A
drm_crtc
.atomic_check
.mode_valid
.reset
drm_encoder A
drm_encoder B
Hotpluggable
drm_connector A
drm_connector B

DRM encoder


Userspace-Created
drm_framebuffer 2
drm_framebuffer 1
Static Objects
drm_plane B
drm_plane A
drm_crtc
drm_encoder A
drm_encoder B
.atomic_enable
.atomic_disable
Hotpluggable
drm_connector A
drm_connector B

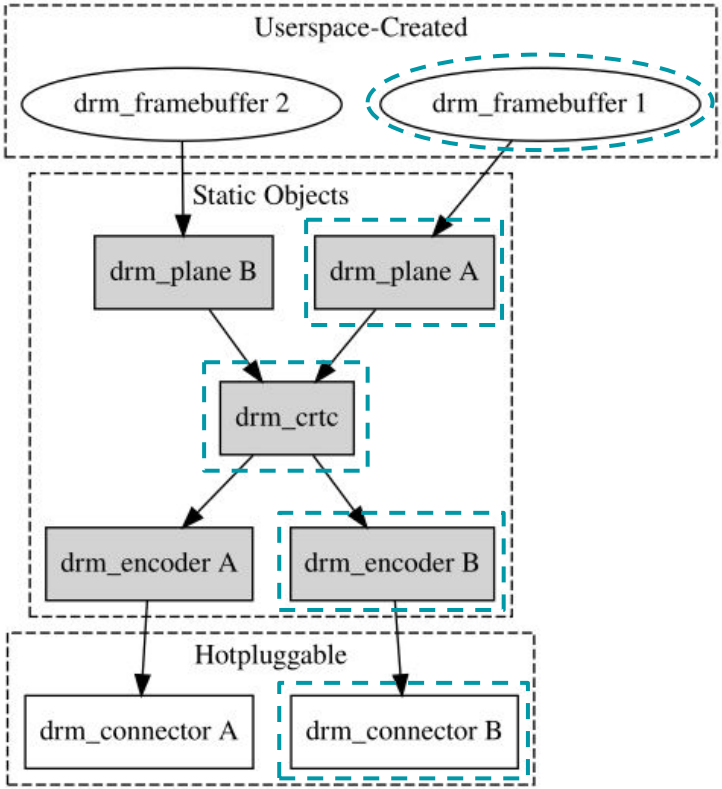
DRM connector



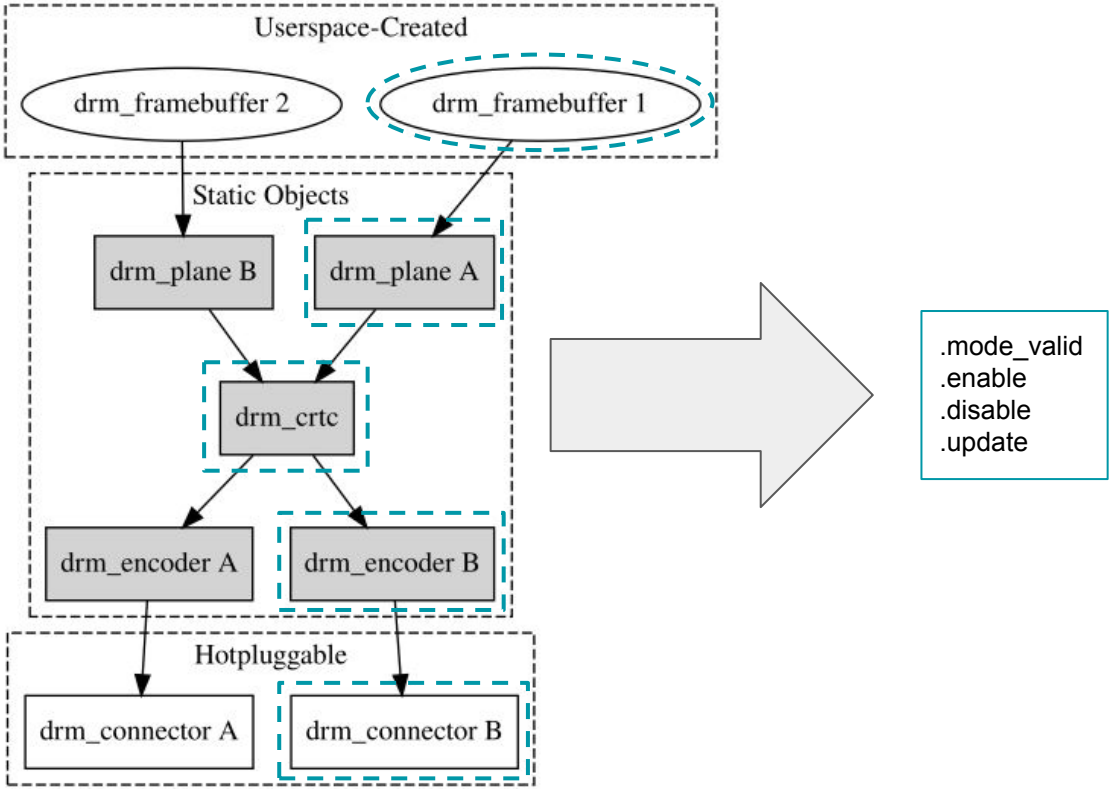
Simple display pipeline (simple-KMS)

- Abstract drivers away from some of the KMS internals and components.
- A set of helpers for simple display devices.
- If only is needed a fullscreen scanout buffer feeding a single output.
- Operate on a “simple display pipeline” with a fixed CRTC and plane.
- Ignore other DRM components such as encoders, connectors, plane types.
- It makes writing simple DRM drivers easier.

Simple display pipeline (simple-KMS)



Simple display pipeline (simple-KMS)



A critique of simple-KMS

- Arguably provides the correct level of abstraction for simple drivers.
  - Simplifies drivers but at the expense of conflating CRTC and plane concepts.
  - Makes the helpers of simple-KMS drivers less flexible and harder to reuse.
- Using the atomic helpers adds flexibility and makes helpers easier be shared.
  - It doesn’t require that much additional code, so simple-KMS isn’t that simple.
  - Better to focus on improving the atomic helpers instead of using another layer.
- For these reasons some drivers were changed to use the full atomic helpers.
  - The ssd130x driver is one of them now.

Porting fbdev drivers to DRM: framebuffer management

fbdev driver: framebuffer memory

The fbdev driver framebuffer is stored in struct fb_info .screen_{base,buffer} union

```
struct fb_info {
...
        union {
                char __iomem *screen_base;    /* Virtual address */
                char *screen_buffer;
        };
...
};
```

fbdev driver: framebuffer memory

The fbdev driver framebuffer is stored in struct fb_info .screen_{base,buffer} union

```
static int ssd1307fb_probe(struct i2c_client *client)
{
…
        struct fb_info *info;
        struct ssd1307fb_par *par;
        void *vmem;
…
        info = framebuffer_alloc(sizeof(struct ssd1307fb_par), dev);
…
        par = info->par;
        par->info = info;
…
        vmem = (void *)__get_free_pages(GFP_KERNEL | __GFP_ZERO, get_order(vmem_size));
…
        info->screen_buffer = vmem;
…
}
```

fbdev driver: framebuffer update

The fbdev driver framebuffer is updated through struct fb_ops operations

```
static const struct fb_ops ssd1307fb_ops = {
        .owner                  = THIS_MODULE,
        .fb_read        = fb_sys_read,
        .fb_write       = ssd1307fb_write,
        .fb_blank       = ssd1307fb_blank,
        .fb_fillrect    = ssd1307fb_fillrect,
        .fb_copyarea            = ssd1307fb_copyarea,
        .fb_imageblit           = ssd1307fb_imageblit,
        .fb_mmap    = fb_deferred_io_mmap,
};
```

fbdev driver: framebuffer update

```
static void ssd1307fb_fillrect(struct fb_info *info, const struct fb_fillrect *rect)
{
        struct ssd1307fb_par *par = info->par;
        sys_fillrect(info, rect);
        ssd1307fb_update_rect(par, rect->dx, rect->dy, rect->width,
                                        rect->height);
}

static void ssd1307fb_copyarea(struct fb_info *info, const struct fb_copyarea *area)
{
        struct ssd1307fb_par *par = info->par;
        sys_copyarea(info, area);
        ssd1307fb_update_rect(par, area->dx, area->dy, area->width,
                                        area->height);
}

static void ssd1307fb_imageblit(struct fb_info *info, const struct fb_image *image)
{
        struct ssd1307fb_par *par = info->par;
        sys_imageblit(info, image);
        ssd1307fb_update_rect(par, image->dx, image->dy, image->width,
                                        image->height);
}
```

fbdev driver: framebuffer update

```
static int ssd1307fb_update_rect(struct ssd1307fb_par *par, unsigned int x,
                                 unsigned int y, unsigned int width,
                                 unsigned int height)
{
        struct ssd1307fb_array *array;
        u8 *vmem = par->info->screen_buffer;
…

        u8 byte = vmem[(8 * i + k) * line_length + j / 8];
        u8 bit = (byte >> (j % 8)) & 1;
        data |= bit << k;
…
        array->data[array_idx++] = data;
…
        ret = ssd1307fb_write_array(par->client, array, width * pages);
}
```

ssd130x DRM driver framebuffer management

DRM driver: framebuffer memory

The ssd130x DRM driver uses SHMEM GEM buffers and shadow planes

```
DEFINE_DRM_GEM_FOPS(ssd130x_fops);

static const struct drm_driver ssd130x_drm_driver = {
        DRM_GEM_SHMEM_DRIVER_OPS,
        .name                   = DRIVER_NAME,
        .desc                   = DRIVER_DESC,
        .date                   = DRIVER_DATE,
        .major                  = DRIVER_MAJOR,
        .minor                  = DRIVER_MINOR,
        .driver_features        = DRIVER_ATOMIC | DRIVER_GEM | DRIVER_MODESET,
        .fops                   = &ssd130x_fops,
};
```

DRM driver: framebuffer update

The ssd130x DRM driver uses SHMEM GEM buffers and shadow planes

```
static const struct drm_plane_helper_funcs ssd130x_primary_plane_helper_funcs = {
        DRM_GEM_SHADOW_PLANE_HELPER_FUNCS,
        .atomic_check = drm_plane_helper_atomic_check,
        .atomic_update = ssd130x_primary_plane_helper_atomic_update,
        .atomic_disable = ssd130x_primary_plane_helper_atomic_disable,
};

static const struct drm_plane_funcs ssd130x_primary_plane_funcs = {
        .update_plane = drm_atomic_helper_update_plane,
        .disable_plane = drm_atomic_helper_disable_plane,
        .destroy = drm_plane_cleanup,
        DRM_GEM_SHADOW_PLANE_FUNCS,
};
```

DRM driver: framebuffer update

The ssd130x DRM driver uses SHMEM GEM buffers and shadow planes

```
static const struct drm_plane_helper_funcs ssd130x_primary_plane_helper_funcs = {
        DRM_GEM_SHADOW_PLANE_HELPER_FUNCS,
        .atomic_check = drm_plane_helper_atomic_check,
        .atomic_update = ssd130x_primary_plane_helper_atomic_update,
        .atomic_disable = ssd130x_primary_plane_helper_atomic_disable,
};

static const struct drm_plane_funcs ssd130x_primary_plane_funcs = {
        .update_plane = drm_atomic_helper_update_plane,
        .disable_plane = drm_atomic_helper_disable_plane,
        .destroy = drm_plane_cleanup,
        DRM_GEM_SHADOW_PLANE_FUNCS,
};
```

DRM driver: framebuffer update

```
static void ssd130x_primary_plane_helper_atomic_update(struct drm_plane *plane,
                                                       struct drm_atomic_state *state)
{
        struct drm_plane_state *plane_state = drm_atomic_get_new_plane_state(state, plane);
        struct drm_plane_state *old_plane_state = drm_atomic_get_old_plane_state(state, plane);
        struct drm_shadow_plane_state *shadow_plane_state = to_drm_shadow_plane_state(plane_state);
        struct drm_device *drm = plane->dev;
        struct drm_rect src_clip, dst_clip;
…
        if (!drm_atomic_helper_damage_merged(old_plane_state, plane_state, &src_clip))
                return;

        dst_clip = plane_state->dst;
        if (!drm_rect_intersect(&dst_clip, &src_clip))
                return;
…
        ssd130x_fb_blit_rect(plane_state->fb, &shadow_plane_state->data[0], &dst_clip);
…
}
```

DRM driver: framebuffer update

```
static int ssd130x_fb_blit_rect(struct drm_framebuffer *fb, const struct iosys_map *vmap,
                                struct drm_rect *rect)
{
        struct ssd130x_device *ssd130x = drm_to_ssd130x(fb->dev);
        struct iosys_map dst;
…
        buf = kcalloc(dst_pitch, drm_rect_height(rect), GFP_KERNEL);
…
        iosys_map_set_vaddr(&dst, buf);
        drm_fb_xrgb8888_to_mono(&dst, &dst_pitch, vmap, fb, rect);
…
        ssd130x_update_rect(ssd130x, buf, rect);
}
```

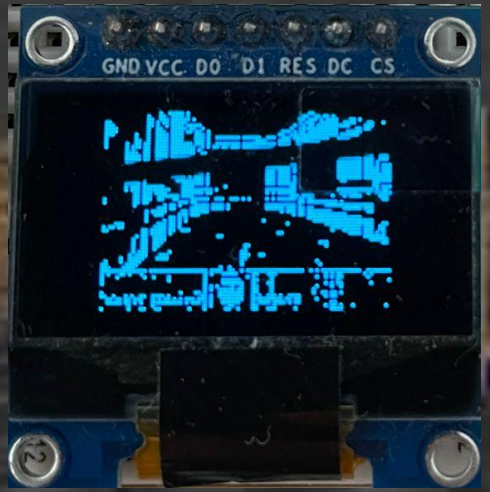
DRM driver: framebuffer update

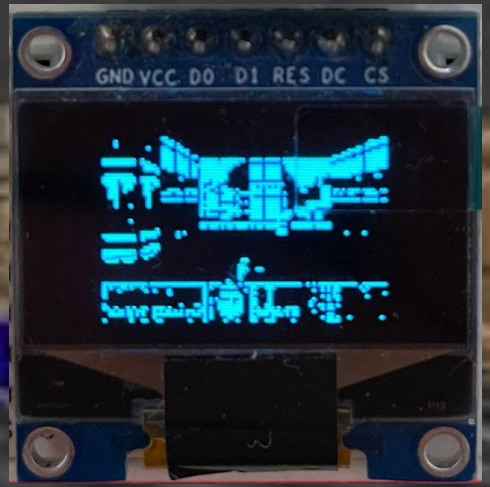
```
static int ssd130x_fb_blit_rect(struct drm_framebuffer *fb, const struct iosys_map *vmap,
                                struct drm_rect *rect)
{
        struct ssd130x_device *ssd130x = drm_to_ssd130x(fb->dev);
        struct iosys_map dst;
…
        buf = kcalloc(dst_pitch, drm_rect_height(rect), GFP_KERNEL);
…
        iosys_map_set_vaddr(&dst, buf);
        drm_fb_xrgb8888_to_mono(&dst, &dst_pitch, vmap, fb, rect);
…
        ssd130x_update_rect(ssd130x, buf, rect); // similar to ssd1307fb_update_rect() in the fbdev driver
}
```

DRM driver: framebuffer update

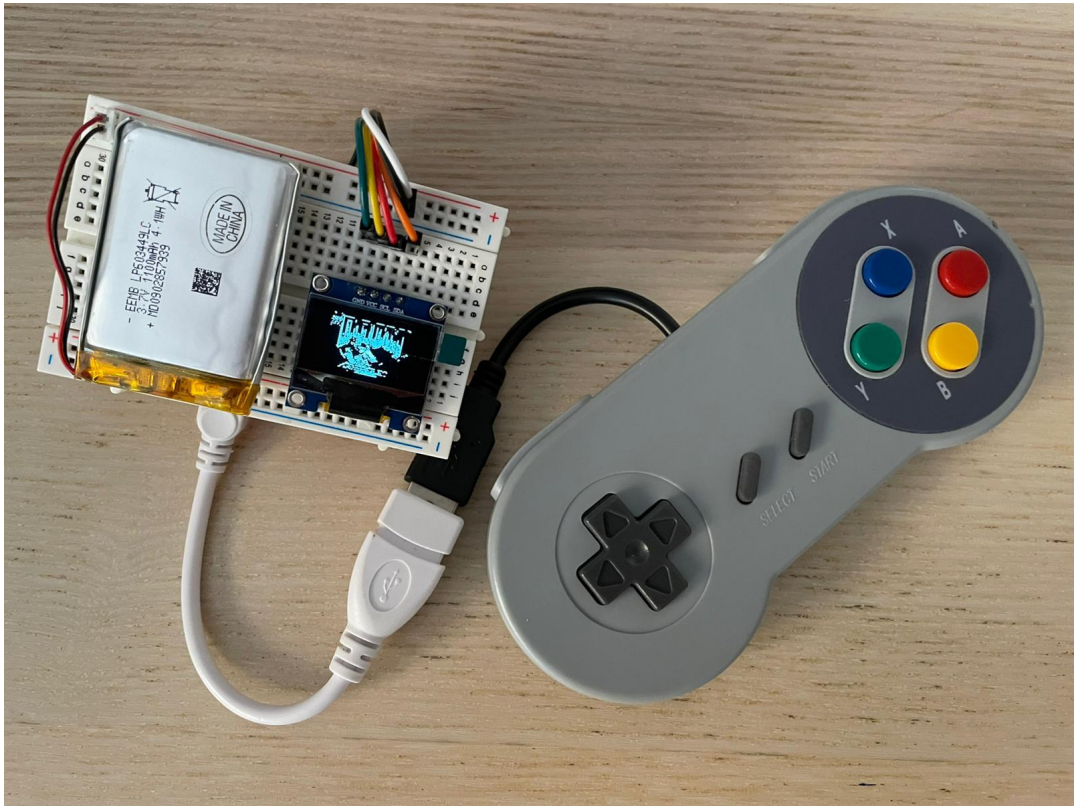
```
static int ssd130x_update_rect(struct ssd130x_device *ssd130x, u8 *buf,
                               struct drm_rect *rect)
{
…
        u8 *data_array = NULL;
…
        data_array = kcalloc(width, pages, GFP_KERNEL);
…
                        for (k = 0; k < m; k++) {
                                u8 byte = buf[(8 * i + k) * line_length + j / 8];
                                u8 bit = (byte >> (j % 8)) & 1;

                                data |= bit << k;
                        }
                        data_array[array_idx++] = data;
...
                ret = ssd130x_write_data(ssd130x, data_array, width * pages);
…
}
```

RUN DOOM OR IT DID NOT HAPPEN ?





Demo!



Acknowledgments

- Thomas Zimmermann
- Maxime Ripard
- Noralf Trønnes
- Sam Ravnborg
- Andy Shevchenko
- Geert Uytterhoeven

Questions?