Overview

The Problem
- What problem are we trying to solve
- How bad is it really?

The Solution
- What have we done so far
- What is left to do?

Future work
- Where do we go next?
- How far are we away?

Demo
- Time permitting
Terminology

**DRM/KMS** - Direct Rendering Manager / Kernel Mode Setting

**SBC** - Single Board Computer

**SOC** - System on a Chip

**V4L2** - Video 4 Linux 2

**BSP** - Board Support Package

**RKMPP** - Rockchip Media Process Platform
The Problem

- Many SOC manufacturers each with their own proprietary blob
- Stuck with vendor BSP kernel which is often old and outdated
- Proprietary blob implements various functions such as windowing (using EGL) and video decoding
- Maintenance burden due to each method being different and requiring a unique code path.

Currently Supported SOC’s in Kodi (v17 Krypton)

- Raspberry Pi
- Amlogic
- i.MX6

Rejected PR’s for SOC’s

- Allwinner (PR6268)
- Rockchip (PR11772)
<table>
<thead>
<tr>
<th>Year</th>
<th>Filepath</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013</td>
<td>xbmc/cores/VideoPlayer/DVDCodecs/Video/DVDVideoCodecIMX.cpp</td>
</tr>
<tr>
<td></td>
<td>xbmc/cores/VideoPlayer/DVDCodecs/Video/DVDVideoCodecIMX.h</td>
</tr>
<tr>
<td>237</td>
<td>xbmc/cores/VideoPlayer/VideoRenderers/HwDecRender/RendererIMX.cpp</td>
</tr>
<tr>
<td>69</td>
<td>xbmc/cores/VideoPlayer/VideoRenderers/HwDecRender/RendererIMX.h</td>
</tr>
<tr>
<td>24</td>
<td>xbmc/linux/imx/GlobalsIMX.cpp</td>
</tr>
<tr>
<td>243</td>
<td>xbmc/linux/imx/IMX.cpp</td>
</tr>
<tr>
<td>207</td>
<td>xbmc/linux/imx/IMX.h</td>
</tr>
<tr>
<td>453</td>
<td>xbmc/windowing/egl/EGLNativeTypeIMX.cpp</td>
</tr>
<tr>
<td>74</td>
<td>xbmc/windowing/egl/EGLNativeTypeIMX.h</td>
</tr>
</tbody>
</table>

**3774** Total lines of platform specific code
The Solution

- DRM/KMS Windowing Method
- FFmpeg decoders outputting AVDRMFrameDescriptor
- DRM Prime Video Rendering Method
DRM/KMS

- Initial implementation - Merged July 8, 2017
- Atomic and Legacy DRM support

Platforms running on DRM/KMS in Kodi

- Rockchip
- Allwinner
- NXP i.MX6
- Raspberry Pi (VC4)
- Qualcomm
- Other?

https://events.static.linuxfound.org/sites/events/files/slides/brezillon drm-kms.pdf
## DRM/KMS Lines of Code (as of January 31st, 2018)

<table>
<thead>
<tr>
<th>File</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>xbmc/windowing/gbm/OptionalsReg.h</td>
<td>64</td>
</tr>
<tr>
<td>xbmc/windowing/gbm/CMakeLists.txt</td>
<td>22</td>
</tr>
<tr>
<td>xbmc/windowing/gbm/DRMLegacy.h</td>
<td>39</td>
</tr>
<tr>
<td>xbmc/windowing/gbm/WinSystemGbm.h</td>
<td>71</td>
</tr>
<tr>
<td>xbmc/windowing/gbm/WinSystemGbmGLESContext.h</td>
<td>59</td>
</tr>
<tr>
<td>xbmc/windowing/gbm/GBMUtils.h</td>
<td>43</td>
</tr>
<tr>
<td>xbmc/windowing/gbm/DRMAbstract.h</td>
<td>43</td>
</tr>
<tr>
<td>xbmc/windowing/gbm/DRMAtomic.h</td>
<td>260</td>
</tr>
<tr>
<td>xbmc/windowing/gbm/GLContextEGL.h</td>
<td>48</td>
</tr>
<tr>
<td>xbmc/windowing/gbm/OptionalsReg.cpp</td>
<td>181</td>
</tr>
<tr>
<td>xbmc/windowing/gbm/GBMUtils.cpp</td>
<td>105</td>
</tr>
<tr>
<td>xbmc/windowing/gbm/DRMUtils.cpp</td>
<td>106</td>
</tr>
<tr>
<td>xbmc/windowing/gbm/DRMLegacy.cpp</td>
<td>172</td>
</tr>
<tr>
<td>xbmc/windowing/gbm/WinSystemGbmGLESContext.cpp</td>
<td>182</td>
</tr>
<tr>
<td>xbmc/windowing/gbm/GLContextEGL.cpp</td>
<td>252</td>
</tr>
<tr>
<td>xbmc/windowing/gbm/WinSystemGbm.cpp</td>
<td>246</td>
</tr>
<tr>
<td>xbmc/windowing/gbm/DRMUtils.cpp</td>
<td>663</td>
</tr>
</tbody>
</table>

**Total:** 2556
Video Decoding and Rendering

- Implementing a zero-copy path from decoding to rendering for each frame
- Allow the decoded frame to be passed directly to a DRM plane
DRMPRIME Decoder & Renderer Lines of Code

248  xbmc/cores/VideoPlayer/VideoRenderers/HwDecRender/RendererDRMPRIME.cpp
74   xbmc/cores/VideoPlayer/VideoRenderers/HwDecRender/RendererDRMPRIME.h

347  xbmc/cores/VideoPlayer/DVDCodecs/Video/DVDVideoCodecDRMPRIME.cpp
83   xbmc/cores/VideoPlayer/DVDCodecs/Video/DVDVideoCodecDRMPRIME.h

752  total
typedef struct AVDRMFrameDescriptor {
    int nb_objects;
    AVDRMOObjectDescriptor objects[AV_DRM_MAX_PLANES];
    int nb_layers;
    AVDRMLayerDescriptor layers[AV_DRM_MAX_PLANES];
};

typedef struct AVDRMOObjectDescriptor {
    int fd;
    size_t size;
    uint64_t format_modifier;
}
FFmpeg Decoding via V4L2 and RKMPP

- Take the video frame, decode it, and place it in a memory buffer.
- Pass the fd and relevant information about this buffer to the renderer. (information contained in AVDRMFrameDescriptor)
Rendering

- Receive the decoded frame from FFmpeg
- Unpack the information in AVDRMFrameDescriptor
- Get the handle from the fd:

  `drmPrimeFDToHandle(m_DRM->m_fd, descriptor->objects[object].fd, &buffer->m_handles[object]);`

- Use the handle and relevant information to add a framebuffer and get a framebuffer ID:

  `drmModeAddFB2(m_DRM->m_fd, buffer->GetWidth(), buffer->GetHeight(), layer->format, handles, pitches, offsets, &buffer->m_fb_id, 0);`
● Set the relevant atomic properties:

```c
atomic->AddPlaneProperty(atomic->m_req, atomic->m_primary_plane, "FB_ID", buffer->m_fb_id);
atomic->AddPlaneProperty(atomic->m_req, atomic->m_primary_plane, "CRTC_ID", atomic->m_crtc->crtc->crtc_id);
atomic->AddPlaneProperty(atomic->m_req, atomic->m_primary_plane, "SRC_X", src_x);
atomic->AddPlaneProperty(atomic->m_req, atomic->m_primary_plane, "SRC_Y", src_y);
atomic->AddPlaneProperty(atomic->m_req, atomic->m_primary_plane, "SRC_W", src_w);
atomic->AddPlaneProperty(atomic->m_req, atomic->m_primary_plane, "SRC_H", src_h);
```

● Commit the atomic page flip
Rendering GUI and Video

- Using Primary and Overlay DRM Planes

<table>
<thead>
<tr>
<th>id</th>
<th>crtc</th>
<th>fb</th>
<th>CRTC x,y</th>
<th>x,y</th>
<th>gamma size</th>
<th>possible.crtcs</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>0</td>
<td>0</td>
<td>0,0</td>
<td>0,0</td>
<td>0</td>
<td>0x000000ff</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>props:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6 type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>flags: immutable enum</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>enums: Overlay=0 Primary=1 Cursor=2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>value: 1</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>0</td>
<td>0,0</td>
<td>0,0</td>
<td>0</td>
<td>0x000000ff</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>props:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6 type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>flags: immutable enum</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>enums: Overlay=0 Primary=1 Cursor=2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>value: 0</td>
</tr>
</tbody>
</table>
Rendering GUI and Video (Continued...)

- No video = Kodi GUI on the primary plane
- Video = Kodi GUI on the overlay plane and video on the primary plane
- We can do this because everything is done in a single atomic commit. So we can switch planes (or disable them) instantly.
- z-pos atomic attribute not widely supported.

Video Playback

No Video Playback
Caveats

- Hardware or driver may not support plane scaling
  - If the screen is 1920x1080 but the video is 1280x720 it will fail (i.MX6)
- No video manipulation via shaders
  - Kodi supports scaling, color correction and deinterlacing via shaders
Alternative Solutions

- Instead of directly passing the fd to the plane we can import it into GLES directly.
- Using the EGL extension `EGL_EXT_image_dma_buf_import`.
- This allows us to import the image so we can do the processing in GL and output to a single plane.
- Current code requires GLES 3.0.
- Can be done in GLES 2.0 and `GL_TEXTURE_EXTERNAL_OES` but requires the HW and GLES driver to allow importing NV12 or other formats directly.
GLint attribsY[] =
{
    EGL_LINUX_DRM_FOURCC_EXT, fourcc_code('R', '8', ' ', ' '),
    EGL_WIDTH, buffer->GetWidth(),
    EGL_HEIGHT, buffer->GetHeight(),
    EGL_DMA_BUF_PLANE0_FD_EXT, descriptor->objects[layer->planes[0].object_index].fd,
    EGL_DMA_BUF_PLANE0_OFFSET_EXT, layer->planes[0].offset,
    EGL_DMA_BUF_PLANE0_PITCH_EXT, layer->planes[0].pitch,
    EGL_NONE
};

eglImageY = m_interop.eglCreateImageKHR(m_interop.eglDisplay,
    EGL_NO_CONTEXT,
    EGL_LINUX_DMA_BUF_EXT,
    (EGLClientBuffer)NULL,
    attribsY);
GLint attribsVU[] =
{
    EGL_LINUX_DRM_FOURCC_EXT, fourcc_code('G', 'R', '8', '8'),
    EGL_WIDTH, (buffer->GetWidth() + 1) >> 1,
    EGL_HEIGHT, (buffer->GetHeight() + 1) >> 1,
    EGL_DMA_BUF_PLANE0_FD_EXT, descriptor->objects[layer->planes[1].object_index].fd,
    EGL_DMA_BUF_PLANE0_OFFSET_EXT, layer->planes[1].offset,
    EGL_DMA_BUF_PLANE0_PITCH_EXT, layer->planes[1].pitch,
    EGL_NONE
};

eglImageVU = m_interop.eglCreateImageKHR(m_interop.eglDisplay,
    EGL_NO_CONTEXT,
    EGL_LINUX_DMA_BUF_EXT,
    (EGLClientBuffer)NULL,
    attribsVU);
glGenTextures(1, &m_textureY);
glBindTexture(GL_TEXTURE_2D, m_textureY);
m_interop.glEGLImageTargetTexture2DOES(GL_TEXTURE_2D, eglImageY);

glGenTextures(1, &m_textureVU);
glBindTexture(GL_TEXTURE_2D, m_textureVU);
m_interop.glEGLImageTargetTexture2DOES(GL_TEXTURE_2D, eglImageVU);

glBindTexture(m_interop.textureTarget, 0);
GLint attribs[] =
{
    EGL_LINUX_DRM_FOURCC_EXT, fourcc_code('N', 'V', '1', '2'),
    EGL_DMA_BUF_PLANE0_FD_EXT, descriptor->objects[layer->planes[0].object_index].fd,
    EGL_DMA_BUF_PLANE0_OFFSET_EXT, layer->planes[0].offset,
    EGL_DMA_BUF_PLANE0_PITCH_EXT, layer->planes[0].pitch,
    EGL_DMA_BUF_PLANE1_FD_EXT, descriptor->objects[layer->planes[1].object_index].fd,
    EGL_DMA_BUF_PLANE1_OFFSET_EXT, layer->planes[1].offset,
    EGL_DMA_BUF_PLANE1_PITCH_EXT, layer->planes[1].pitch,
    EGL_NONE
};

eglImage = m_interop.eglCreateImageKHR(m_interop.eglDisplay,
    EGL_NO_CONTEXT,
    EGL_LINUX_DMA_BUF_EXT,
    (EGLClientBuffer)NULL,
    attribs);
Bonjour
Broadcom

- Raspberry Pi
- No V4L2 Support
- Mainline mesa (not used)
- Proprietary bootloader
- Proprietary decoding
Amlogic

- Mainline DRM driver (not used)
- No V4L2 support
- Android partitioning
- Mali proprietary blob
- Magic decoding through libamcodec
Allwinner

- DRM driver almost mainline
- Early V4L2 support
- Mainline u-boot
- Mali proprietary blob
Rockchip

- 4.4 based kernel
- No V4L2 support
- Mainline u-boot
- Mali proprietary blob
- Vendor specific decoding
Qualcomm

- Mainline kernel (almost)
- Great V4L2 support
- Android partitioning
- Mainline mesa
- DB820 soon
Freescale

- Mainline kernel
- Mainline u-boot
- Good V4L2 support
- Mainline mesa
- i.MX8 coming soon
Future Work

**FFmpeg**
- Mainline support for v4l2 outputting AVDRMFrameDescriptor

**Kodi**
- HDR output
Demo
Questions?