Zero-Copy Video Streaming on Embedded Systems the Easy Way

Michael Tretter - m.tretter@pengutronix.de
Philipp Zabel - p.zabel@pengutronix.de
Examples

- Presentation capture and streaming
- Augmented Reality
- UAV video downlink
- Intercom
Agenda

- Video and graphics on embedded devices
- Hardware acceleration units and Zero-Copy buffer sharing
- Case study: i.MX6
- The easy way
- Open Issues & Future Work
Building Blocks

- Recording / Streaming
- Receiving / Projection / Compositing
- Lens correction / Warping
- Transcoding
Embedded System Requirements

- Portable
- Energy efficient
- Lightweight
- Soft real-time
- “High” data rates

Limited processing power vs. Audio/Video use case
**Specialized Co-processors**

- Graphics Processing Unit
- Video encoder and decoder
- FPGA
- Camera
- Display Controller
- Network Controller

- Supported or preferred format in memory differ
- Copy and conversion between hardware units required
Zero-Copy

"Zero-copy" describes computer operations in which the CPU does not perform the task of copying data from one memory area to another. (Wikipedia)

- Copying in CPU is expensive
- CPU memory bandwidth smaller than hardware acceleration units
- CPU cache management
Putting it all Together, Case study: i.MX6

- Memory bandwidth
  - Up to 533 MHz DDR3 SDRAM (1066 MT/s @ 64-bit)
  - Realistically, up to 2.5 GiB/s on i.MX6Q, more on i.MX6QP

- Up to quad-core Cortex A9, 1 GHz
- CPU memcpy ~500 MiB/s
- 1080p30 YUYV: 120 MiB/s
- Cache management overhead
Putting it all Together, Case study: i.MX6

- GPU: Vivante GC2000
- Display: IPUv3 display interface
- Camera: IPUv3 capture interface
- VPU: Chips&Media CODA960
Device Drivers and Interfaces

- GPU: Vivante GC2000
- Display: IPUv3 display interface
- Camera: IPUv3 capture interface
- VPU: Chips&Media CODA960

- etnaviv (DRM)
- imx-drm (DRM, KMS)
- imx-media (Video4Linux2), staging
- coda (Video4Linux)

- Userspace: Mesa/etnaviv (OpenGL)

- DMABuf
Before DMABuf

Capture (IPUv3 CSI, V4L2)

Encode (CODA960, V4L2)

Userspace

Kernel

mmap

R

CPU copy

W

mmap

W

R

W
DMABuf

Capture (IPUv3 CSI, V4L2)  Encode (CODA960, V4L2)

Userspace

fd  dup

Kernel

W

R  W
DMABuf

Decode (CODA960, V4L2)  Display (IPUv3, DRM)

Userspace

Kernel

R  W

fd  dup

R
Device Drivers and Interfaces (API)

- **V4L2 ioctl**:  
  V4L2 ioctl:
  - VIDIOC_EXPBUF
  - VIDIOC_QBUF

- **DRM ioctl**:  
  DRM ioctl:
  - DRM_IOCTL_PRIME_HANDLE_TO_FD
  - DRM_IOCTL_PRIME_FD_TO_HANDLE

- **EGL extensions**:  
  EGL extension:
  - EGL_EXT_image_dma_buf_import
  - EGL_EXT_image_dma_buf_import_modifiers
  - EGL_MESA_image_dma_buf_export

Import/export DMABuf handles from/into video devices  
Import/export DMABuf handles from/into GPU or display controller devices, used by libdrm/Mesa  
These sit on top of DRM_IOCTL_PRIME_*
Device Drivers and Interfaces: V4L2

/* V4L2 DMABuf export */

int video_fd = open("/dev/v4l/by-name/csi", O_RDWR);

struct v4l2_requestbuffers reqbuf = {
    .count = 1,
    .type = V4L2_BUF_TYPE_VIDEO_CAPTURE,
    .memory = V4L2_MEMORY_MMAP,
};
ioctl(video_fd, VIDIOC_REQBUFS, &reqbuf);

struct v4l2_exportbuffer expbuf = {
    .type = V4L2_BUF_TYPE_VIDEO_CAPTURE,
    .index = 0,
};
ioctl(video_fd, VIDIOC_EXPBUF, &expbuf);

int dmabuf_fd = expbuf.fd;

/* V4L2 DMABuf import */

int dmabuf_fd;
int video_fd = open("/dev/v4l/by-name/coda", O_RDWR);

struct v4l2_requestbuffers reqbuf = {
    .count = 1,
    .type = V4L2_BUF_TYPE_VIDEO_OUTPUT,
    .memory = V4L2_MEMORY_DMABUF,
};
ioctl(video_fd, VIDIOC_REQBUFS, &reqbuf);

struct v4l2_buffer buf = {
    .type = V4L2_BUF_TYPE_VIDEO_OUTPUT,
    .memory = V4L2_MEMORY_DMABUF,
    .index = 0,
    .m.fd = dmabuf_fd,
};
ioctl(video_fd, VIDIOC_QBUF, &buf);

Device Drivers and Interfaces: EGL/OpenGL ES

EGLint attrib_list[] = {
    EGL_WIDTH, 1920,
    EGL_HEIGHT, 1280,
    EGL_LINUX_DRM_FOURCC_EXT,
    DRM_FORMAT_YUYV,
    EGL_DMA_BUF_PLANE0_FD_EXT, dmabuf_fd,
    EGL_DMA_BUF_PLANE0_FD_OFFSET_EXT, 0,
    EGL_DMA_BUF_PLANE0_FD_PITCH_EXT, 3840,
    EGL_NONE,
};
EGLImageKHR egl_image = eglCreateImageKHR(
    egl_display,
    EGL_NO_CONTEXT,
    EGL_LINUX_DMA_BUF_EXT,
    NULL,
    attrib_list);

glEGLImageTargetTexture2DOES(
    GL_TEXTURE_EXTERNAL_OES,
    egl_image);

int dmabuf_fd;
int stride;
eglExportDMABUFImageMESA(
    egl_display,
    egl_image,
    &dmabuf_fd,
    &stride);

https://www.khronos.org/registry/EGL/extensions/EXT/EGL_EXT_image_dma_buf_import.txt
https://www.khronos.org/registry/EGL/extensions/MESA/EGL_MESA_image_dma_buf_export.txt
The Easy Way
GStreamer

“GStreamer is a library for constructing graphs of media-handling components. The applications it supports range from simple Ogg/Vorbis playback, audio/video streaming to complex audio (mixing) and video (non-linear editing) processing.

Applications can take advantage of advances in codec and filter technology transparently. Developers can add new codecs and filters by writing a simple plugin with a clean, generic interface.”
GStreamer

gst-launch-1.0 playbin uri=file:///home/mtr/Videos/tears_of_steel_720p.mkv
GStreamer

- Sink support
  - Wayland
  - WebRTC
  - QML
  - ...

- Plugins
  - V4L2
  - OpenGL
  - Third party
  - ...

- Language bindings
  - C++
  - Python
  - Rust
  - ...

- Autoplugging
  - decodebin
  - encodebin
  - playsink
  - ...

Slide 20 - © Pengutronix - http://www.pengutronix.de - 10/25/2017
Video4Linux2

- Elements: `v4l2sink`, `v4l2videodec`, `v4l2videoenc`, ...
- Support DMABuf import and export
- Recent feature: stable element names

- Nicolas Dufresne - Implementing Zero-Copy pipelines in GStreamer
- GStreamer Conference 2017
Direct Rendering Manager / Kernel Mode Setting

- Kernel subsystem for video cards
- API and user space library
- Element: kmssink
- Import DMABuf automatically
- Output via video card
- Depends on features of kms driver (e.g., no scaling on i.MX6)

Simple tool for testing
Wayland

- Display server protocol
- DMABuf: linux_dmabuf_unstable_v1
- Compositor decides
  - OpenGL upload for compositing
  - Display as overlay
- Element: waylandsink
- Your mileage may vary
- Depends on compositor
- Imported format might not be supported
GStreamer on i.MX6: Sender

- Camera: v4l2src
- CODA encode: v4l2h264enc

```
gst-launch-1.0 v4l2src io-mode=dmabuf\ 1 device=/dev/v4l/by-name/csi ! v4l2h264enc output-io-mode=dmabuf-import\ 2 ! rtph264pay ! udpsink
```

1 Still necessary in GStreamer 1.12, automatic in master
2 Still necessary, will be auto-negotiated in the future
GStreamer on i.MX6: Receiver

- CODA decode: v4l2h264dec
- GPU, display: waylandsink

```sh
gst-launch-1.0 udpsrc ! application/x-rtp,payload=96 ! rtpromise ! rtph264depay ! h264parse ! v4l2h264dec\io-mode=dmabuf\ ! waylandsink
```

1 Stable element names in master, for 1.12: `v4l2videodec` device=/dev/videoX
2 Still necessary in GStreamer 1.12, automatic in master
Camera Input Pipeline

- Media input pipeline
- Currently needs manual configuration: media-ctl
- Pavel Machek: Cheap Complex Cameras (http://sched.co/ByYH)

```
media-ctl --links "'tc358743 1-000f':0->'imx6-mipi-csi2':0[1]"
media-ctl --links "'imx6-mipi-csi2':1->'ipu1_csi0_mux':0[1]"
media-ctl --links "'ipu1_csi0_mux':2->'ipu1_csi0':0[1]"
media-ctl --links "'ipu1_csi0':2->'ipu1_csi0 capture':0[1]"
media-ctl --set-dv "'tc358743 1-000f':0"
media-ctl --set-v4l2 "'tc358743 1-000f':0[fmt:UYVY8 1X16/1920x1080]"
media-ctl --set-v4l2 "'imx6-mipi-csi2':1[fmt:UYVY8 1X16/1920x1080]"
media-ctl --set-v4l2 "'ipu1_csi0_mux':2[fmt:UYVY8 1X16/1920x1080]"
media-ctl --set-v4l2 "'ipu1_csi0':0[fmt:UYVY8 1X16/1920x1080@1/60]"
media-ctl --set-v4l2 "'ipu1_csi0':2[fmt:AYUV32/1920x1080@1/30]"
```
Future Work

- Useful default media-controller configuration
- Mesa/etnaviv
  - NV12 and YUYV texture import (GL_TEXTURE_2D)
  - Direct sampling from linear buffers
  - OpenCL support
- Weston: Atomic modesetting patchset for overlay plane support
Open Questions

- Camera pipeline configuration → Autoconfiguration? Device-tree default?
- Remaining proprietary blob: CODA VPU firmware
- V4l2 access as root → Pipewire?
Conclusion

- Modern embedded system use various coprocessors
- DMABuf is usable abstraction for zero-copy on Linux
- Let GStreamer manage all the ugly details

- Know your hardware
- Be aware of corner cases
- Check resulting GStreamer pipeline
- Zero-copy between driver blobs problematic or impossible → Avoid blobs!
Thank You!

- Questions?