[sys-base]

# V4L2 Stateless Video Encoding: Hardware Support and uAPI

**Paul Kocialkowski**

paulk@sys-base.io

FOSDEM 2024

Saturday February 3rd 2024

## Why encode videos?

Because pictures are too big.

## So let's compress videos!

And now they look crappy.

# Rationale

## Main topic of encoding:

Trade-off betwen data size and perceived quality.

# Video Codecs

What makes a good codec?

- Better trade-off between size and perceived quality
- Codecs have improved very significantly
- Less size with more perceived quality

Codec specifications:

- Standards and specifications (hard to read)
- Some require royalties, some don't
- Good fit for both software and hardware implementations
- Hype codecs: AV1, VP9, H.265 (HEVC), H.266 (VVC)
- Adoption of new standards is slow

Can help drastically reduce global network/storage power consumption.

# Video Compression Techniques

Common video compression techniques:

- Spatial compression:
  - YUV chroma sub-sampling (typical *4:2:0 8-bit*)
  - Frequency domain transform and quantization (*QP* value)
  - Intra prediction (redundancy)
  - Entropy coding
- Temporal compression:
  - Intra-coded (**I**) frames: without reference
  - Predictive (**P**) frames: with past references
  - Bi-predictive (**B**) frames: with future and past references
- Pictures are split in **macroblocks**, with a deblocking filter
- Various more advanced codec-specific techniques

# Video Compression Techniques

Visualization of inter-frame motion vectors

*Caminandes 2: Gran Dillama, Blender Foundation (2013)*

# Video Encoding Techniques

Strategies for target behavior/use case:

- Constant/average bitrate (**CBR/ABR**)
- Constant quality (**CQP/CRF**)
- Variable bitrate (**VBR**)
- Fine-tuned, custom

Rate-control feedback loop implementation:

- Implement the selected strategy
- Decide on frame type and quantization parameter ($QP$)
- Handle variable scenes and react to changes

Rate-control implementation is key for best results!

# Hardware Video Encoding

Video encoding acceleration:

- CPU-based encoding is generally very demanding/slow
- Use-cases with high sizes and frame rates
- Use-cases with on-the-spot (real-time) needs (cameras)
- Dedicated hardware encoder circuits relieve the pain!

Hardware encoder features:

- Produce conformant bitstream for codec(s)
- Common pre-processing: format adaptation, anti-shake, crop
- Usual limitations: profile/level support, number of reference slots
- Time-sharing between contexts (parallelization too!)

# Hardware Video Encoding Implementation

Two major types of hardware implementations:

- **Stateful** encoders (abstracted, less flexible):
  - Include a dedicated micro-controller and compression units
  - Firmware (proprietary) manages: context (state), memory, rate-control
  - Mailbox and message interface with main CPU
  - Generates bitstream with coded meta-data and picture data
- **Stateless** encoders (bare-metal, more flexible):
  - No micro-controller, only compression units
  - CPU-side driver driver manages: context(state), memory and rate-control
  - Register-driven configuration from main CPU
  - Generates bitstream with coded picture data only

Memory considerations:

- Reconstruction buffers for references
- Dedicatetd DMA memory (without IOMMU), cache coherency
- Zero-copy buffer sharing from other units (camera)

# Hardware Video Encoding Known Designs

Known stateful designs:

- Imagination: **PowerVR VPU**
- Chips&Media: **CODA, WAVE**
- Allegro/Amphion: **Windsor**
- Qualcomm: **Venus**, **Iris**
- Samsung: **MFC**
- Amlogic: **VPU**
- Mediatek: **Video Codec**
- NVIDIA: **NVENC**
- AMD: **VCE**

Known stateless designs:

- Verisilicon: **Hantro**
- Allwinner: **Video Engine**
- Intel: **Quick Sync Video**
- Maybe more?

# V4L2 Stateful Encoding Support

Stateful encoding API:

- **V4L2 memory-to-memory (M2M)** API with 2 queues:
  - Single video device
  - Output queue: picture *(source)*
  - Capture queue: coded *(destination)*
- Dedicated pixel formats: e.g. `V4L2_PIX_FMT_H264`
- Dedicated controls for encoding features and rate-control: e.g. `V4L2_CID_MPEG_VIDEO_H264_ENTROPY_MODE`
- Frame interval enumeration and selection
- Frame size enumeration, alignment and target crop
- Supported by **GStreamer** and **FFmpeg**

# V4L2 Stateless Encoding Support

Stateless encoding is significantly more complex:

- Bitstream meta-data needs to be generated
- Rate-control needs to be implemented
- References need to be selected explicitly
- More memory management needed: side and reconstruction buffers
- uAPI still needs to be hardware-agnostic

Stateless encoding should be flexible:

- Low-level control over the hardware opens possibilities
- Userspace might know relevant information
- Userspace might want/need custom rate-control
- Simple/usual cases should be covered without too much userspace logic

# V4L2 Stateless Encoding: Hantro H1

Existing work (not mainline-based):

- **MPP** (Rockchip):
  - User-space rate-control and meta-data bitstream generation
  - Custom interface with full userspace register configuration
  - `https://github.com/rockchip-linux/mpp`, path: `mpp/hal/vpu/h264e/`
- **ChromiumOS custom V4L2 driver** (Google):
  - User-space rate-control and meta-data bitstream generation
  - Custom register configuration and feedback data via V4L2 controls
  - Kernel: `https://chromium.googlesource.com/chromiumos/third_party/kernel/`, branch: `chromeos-4.4`, path: `drivers/media/platform/rockchip-vpu/`

# V4L2 Stateless Encoding: Hantro H1

Mainline-based attempts:

- **H.264 encoding** (Bootlin):
  - User-space rate-control (basic) and meta-data bitstream generation
  - Custom register configuration and feedback data via V4L2 controls
  - Kernel: `https://github.com/bootlin/linux`, branch: `hantro/h264-encoding-v5.11`
  - Userspace: `https://github.com/bootlin/v4l2-hantro-h264-encoder`
- **VP8 encoding** (Collabora):
  - User-space rate-control (basic), kernel-side meta-data bitstream generation
  - Kernel: [RFC 0/2] VP8 stateless V4L2 encoding uAPI + driver
  - Userspace: GStreamer merge request #3736

Hardware notes:

- Specific constraints on some meta-data fields
- In-loop rate-control helpers (checkpoints, MAD)

# V4L2 Stateless Encoding: Allwinner Video Engine

Existing work:

- **A10/A13/A20 cedrus h264enc** (Jens Kuske):
  - Research effort from the *linux-sunxi* community: `https://linux-sunxi.org/VE_Register_guide`
  - User-space rate-control (basic) and meta-data bitstream generation: `https://github.com/jemk/cedrus.git`
  - Using Allwinner's downstream kernel driver
  - Fully userspace implementation (MMIO register map)

Mainline-based attempt:

- **V3/V3s/S3 H.264 encoding** (Bootlin):
  - Kernel-side rate-control (basic) and bitstream generation
  - Using the stateful encoding uAPI (more or less)
  - Complete re-architecture of the cedrus driver
  - Kernel: `https://github.com/bootlin/linux`, branch: `cedrus/h264-encoding`
  - Userspace: `https://github.com/bootlin/v4l2-cedrus-enc-test`

# V4L2 Stateless Encoding uAPI: Lessons Learned

Bottomline:

- Re-using the stateful API brings significant limitations
- Bitstream meta-data needs to be produced kernel-side
- Rate-control on kernel-side is simple but limiting
- Rate-control in userspace is flexible but more involved

State of the art:

- Finding an acceptable middle-ground is hard
- Ongoing discussions on the *linux-media* mailing-list
- uAPI is needed before adding drivers

**Stateless Encoding uAPI Discussion and Proposal**

`https://lore.kernel.org/linux-media/ZK2NiQd1KnraAr20@aptenodytes/`

# V4l2 Stateless Encoding uAPI: Proposal and Thoughts

Possible ways forward:

- Have a switch between kernel-side and user-side rate-control?
  - Stateful uAPI clone for simple cases
  - Explicit frame type, QP and reference list decision for advanced needs
- Provide suggestions, let userspace decide:
  - Feedback data provided from kernel-side rate-control implementation
  - Let userspace decide and tweak suggestion
  - Have a switch to auto-apply feedback for next frame
- Common code for stateless encoders:
  - Codec-specific bitstream meta-data generation
  - Rate-control implementations

Follow-up work:

- Merge encoder work in *hantro/verisilicon* and *cedrus* drivers
- Gstreamer and FFmpeg integration

Thanks for listening!