[sys-base]

V4L2 Stateless Video Encoding uAPI

Paul Kocialkowski

paulk@sys-base.io

Linux Media Summit
Tuesday May 13th 2025

Stateless Video Codecs

Types of hardware video codecs:

- **Stateful**: driven by a dedicated microcontroller
  - Abstracted commands via mailbox interface for (proprietary) firmware
  - Limited control and decisions
  - Easier to use but low flexibility
  - Often wrapping stateless codecs
- **Stateless**: driven by the main processor
  - Register-level configuration
  - Low-level control and advanced decisions
  - More involved but high flexibility

V4L2 support:

- **Stateful encoders and decoders**: using V4L2 controls (simple) and M2M API
- **Stateless decoders**: using V4L2 controls (compound), M2M and Media Request APIs
- **Stateless encoders**: missing upstream support

Stateless Video Encoding

Main challenges:

- Decide of all encode parameters
  - Codec features
  - Quality vs size trade-off
  - Reference frame management for inter-prediction
  - Account for (variable) hardware limitations: features, reference slots, etc
- Implement rate control
  - Closed-loop regulation for each strategy
  - Assisted by generic and hardware-specific statistics
  - Possibly assisted by hardware-specific mechanisms
  - Possibly using external feedback/analysis (e.g. ML)
- Generate (valid and consistent) bitstream headers
- Manage memory not visible to userspace
  - Reconstruction buffers for references
  - Various scratch/metadata buffers (e.g. motion vectors)

Concerned Hardware and Formats

Applicable hardware:

- VeriSilicon Hantro H1/H2/VC8000E
- Allwinner Video Engine
- Rockchip RKVENC VEPU510/VEPU540/VEPU580
- Stateful designs with pass-through firmware?

Relevant video codec formats:

- **H.264/AVC**: main target of interest
- **H.265/HEVC**: common
- **VP8, VP9**: common
- **AV1**: recent

Linux uAPI Attempts

- **2021**: Hantro H1 H.264 encoding on PX30/RK3399,
  - Paul Kocialkowski, *Bootlin*
  - Userspace-driven, hardware-specific controls
- **2023**: Hantro H1 VP8 encoding on RK3399,
  - Andrzej Pietrasiewicz, *Collabora*
  - Userspace-driven, hardware-specific controls
- **2023**: Allwinner Video Engine H.264 encoding on V3/V3s,
  - Paul Kocialkowski, *Bootlin*
  - Kernel-driven, stateful controls
- **2025**: Hantro VC8000E H.264 encoding on i.MX8MP,
  - Marco Felsch, *Pengutronix*
  - Userspace-driven, hardware-specific controls

Linux uAPI Attempts Approaches

Userspace-driven:

- Bitstream headers generated by userspace (has to know and apply hardware constraints)
- Hardware-specific parameters and statistics controls
- Userspace rate control decision, lock-step encoding (or latency)
- Little to no decision from the kernel, very flexible
- Involved/complex userspace, implementation not hardware-agnostic

Kernel-driven:

- Bitstream headers generated by the kernel
- Stateful encoder controls
- Kernel rate control decision, no lock-step encoding
- All decisions from the kernel, not flexible
- Wastes all advantages of stateless
- Hardware-agnostic, simple userspace

Existing Userspace APIs

Vendor-specific libraries:

- Very low-level, control over most encode parameters and hardware features
- Bluntly refuse unsupported cases
- Stateful proprietary rate control strategy with hints

VA-API

- Low-level encode parameters, explicit reference list
- Some codec capabilities reporting (e.g. reference limitations)
- Stateful rate control strategy with hints

Vulkan Video

- Low-level encode parameters, explicit reference list
- Some codec capabilities reporting (e.g. reference limitations)
- Allows encode parameters and reference override
- Stateful rate control strategy with hints

Design Goals

Desirable uAPI design:

- Fully hardware-agnostic
- Leverage and expose low-level aspect of stateless encoders
- Compatible with VA-API and Vulkan Video
- Work without lock-step encoding
- Covering all expected use cases:
  - Camera source recording
  - Camera streaming (RTP, RTCP RPSI, WebRTC)
  - Offline transcoding and storage (bi-directional prediction)

Acceptable downsides:

- Involved/complex userspace implementations (few in practice)

Proposed Path: Overview

Userspace side:

- Decides the low-level encode parameters
  including frame type (may have extra relevant info)
- Based on capability reporting?
- Decides the reference list explicitly
- Selects rate control strategy and provides hints
- Follows common stateless encoder specification

Kernel side:

- Adapts parameters and references based on hardware capabilities
- Returns actually selected parameters and references
  provided as read-only controls attached to media request

Proposed Path: Encode Parameters

Possible approaches:

- Using existing controls (stateful encoders):
  - Implicit (presence) and explicit (value) capability reporting
  - Missing some parameters (e.g. frame type)
- Using flat compound controls (stateless decoders):
  - Cover most codec headers
  - No capability reporting

Proposed Path: References

Userspace side:

- Flat global list of possible references (e.g. H.264 DPB)
- Per-frame list of active references (e.g. H.264 L0/L1)
- Frames marked as possible reference when submitted

Kernel side:

- May need to apply reference list management operations
- May need to discard some references to match hardware constraints
- May benefit from decorelation with V4L2 queue

Proposed Path: Rate Control

Userspace side:

- Select rate control strategy (using existing stateful controls)
- Provide hints about GOP structure (reusing existing stateful controls)

Kernel side:

- Generic and common implementations for various strategies
  - Tracking accumulated bitstream size
  - Detecting actual frame type patterns
  - Complying to HRD, VBV and level constraints (with frame drops?)
  - Estimating quantization parameter
- Final decision and tweaking by drivers,
  allowing hardware-specific statistics and optimizations

Proposed Path: Bitstream Generation

Kernel side:

- Generic codec-specific helpers (e.g. exponential-Golomb)
- Ingesting bitstream headers flat structures (from controls)
- Calling I/O hooks for buffer append
  - Generic using virtual memory access
  - Driver-specific using hardware mechanisms
- Also relevant for some stateful encoder drivers (e.g. Allegro-DVT)

Extensions/Future Planning

Specific extra features of interest:

- Interlaced encoding (probably H.264 only)
- Slice-granularity encoding
- Scalable coding (H.264 SVC, SHVC)
- Custom kernel-side rate control with eBPF?

Implementation notes:

- Typically added with extra V4L2 controls
- Not within scope for immediate focus
- Beware of incompatible choices

Involved Parties and Timeline

Informal working group:

- Benjamin Gaignard, *Collabora*
- Marco Felsch, *Pengrutronix*
- Michael Tretter, *Pengutronix*
- Nicolas Dufresne, *Collabora*
- Paul Kocialkowski, *sys-base*
- Let us know if interested!

Timeline Targets:

- First working prototype with GStreamer plugin during summer 2025
- Technical community consensus before 2026

Takeaway

Summary:

- Most encode parameters are best decided by userspace
- References are best decided by userspace
- Rate control is best decided by the kernel
- Bitstream generation is best done by the kernel
- Kernel drivers need to be able to enforce hardware constraints
- Compatibility with existing userspace APIs is desirable
- Various aspects and details are still open to discussion