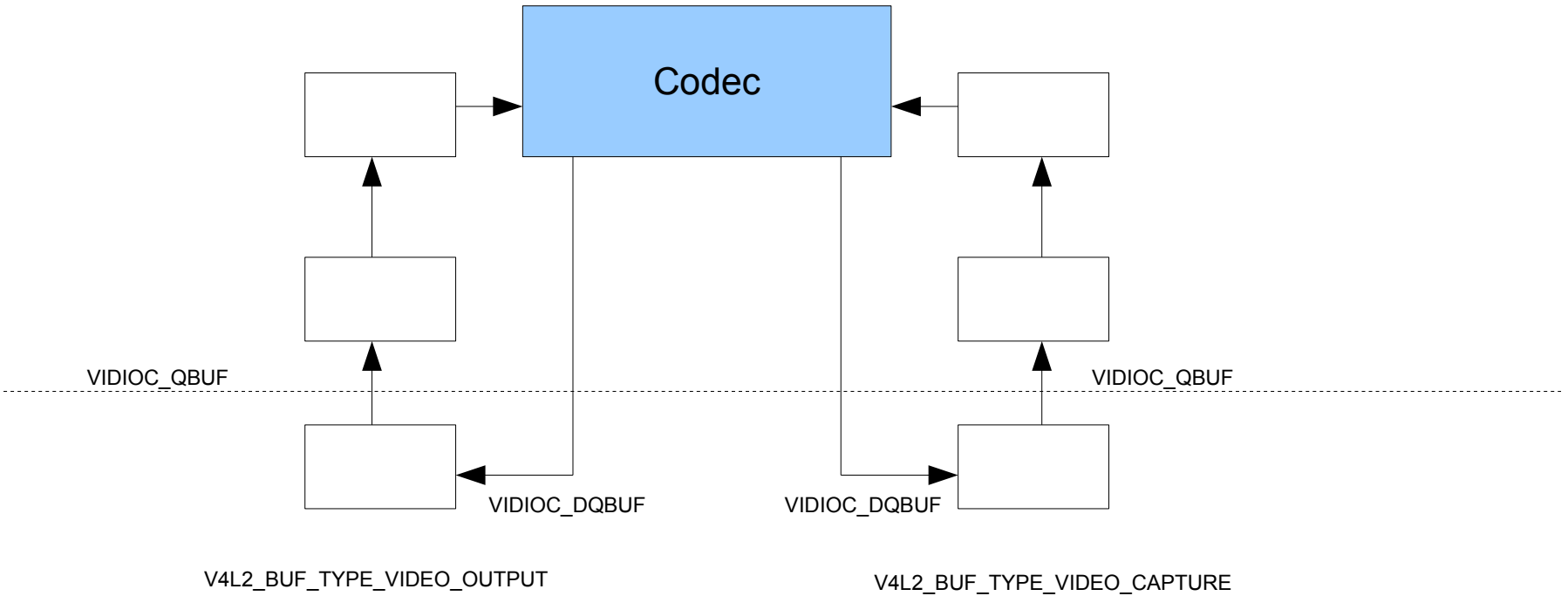
V4L2: A Status Update

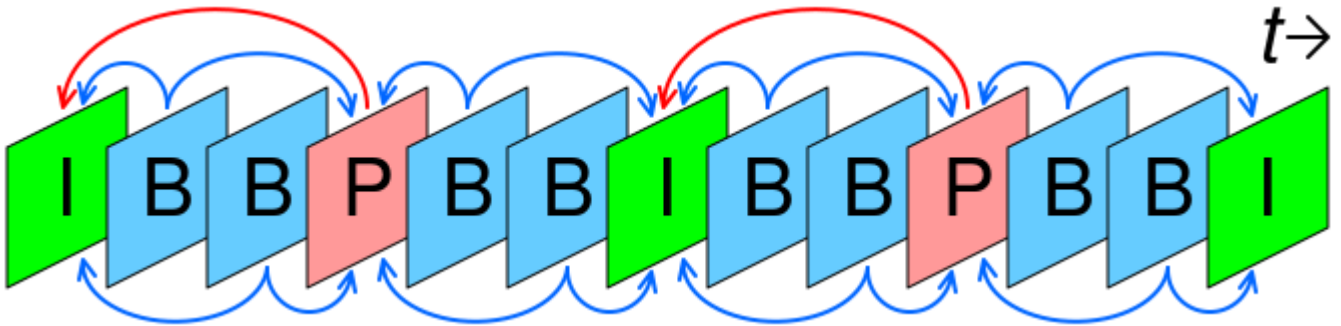
Hans Verkuil
Cisco Systems Norway

Hardware Codec Support

Codec Devices



Codecs: I/B/P-Frames



I-Frame: Intra-coded picture
P-Frame: Predicted picture
B-Frame: Bidirectional predicted picture

Picture from https://en.wikipedia.org/wiki/Inter_frame, Cmglee [CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)]

Stateful Codec Support

- Stateful HW Codecs keep track of the state themselves, the application just provides the raw frames (for the encoder) or bytestream (for the decoder) and it is compressed/decompressed without further interventions.
- Depending on the codec format the application will have to do some basic parsing for the decoder to split the stream into separate frames: while some HW decoders have full bytestream parsing support, most decoders expect all data for one encoded frame in one buffer.
- Detailed decoder and encoder specification are being written. They describe how to use the V4L2 API for stateful encoders and decoders, including information on how to handle seeks and mid-stream resolution changes. Google (Chromium/ChromeOS team) is the main contributor for this. The detailed decoder spec has been merged into the V4L2 Specification, the last remaining issues for the encoder spec will hopefully be resolved soon.
- Testing of this API is important: we created a vicodec driver that we can use for prototyping and testing of stateful encoders and decoders. The v4l2-compliance utility can test codecs and is also used in daily regression testing against vicodec.

Stateless Codec Support

- Stateless HW Codecs do not keep track of the state, instead the application has to provide it together with the raw frame (for the encoder) or compressed frame (for the decoder). No bytestream parsing takes place, and reference frames needed for decoding have to be provided explicitly.
- Currently there are no implementations for stateless encoders, so this is on hold until they appear. For stateless encoders the driver will have to keep track of some state, specifically relating to reference frames which have to be allocated by the driver.
- A detailed specification on how to use the V4L2 API for stateless decoders, including information on how to handle seeks and mid-stream resolution changes, has been merged for 5.5. Google (Chromium/ChromeOS team) is the main contributor for this.
- Testing of this API is important: the vicodec driver also supports a stateless decoder that we can use for prototyping and testing of core frameworks. The stateless decoder support was developed by Dafna Hirschfeld as part of her Outreachy project. The v4l2-compliance utility is unfortunately not yet capable of testing the vicodec stateless decoder.

Stateless Decoder Support

- Stateless decoders use the new Request API framework. There are currently two stateless decoders merged: cedrus for Allwinner SoCs and hantro for Rockchip and i.MX8 SoCs.
- Currently supported codecs are MPEG-2, H.264 and VP8. HEVC support is being worked on. VP9 is planned as well.
- Stateless decoder support is still in staging: the API isn't considered fully stable yet.

Request API

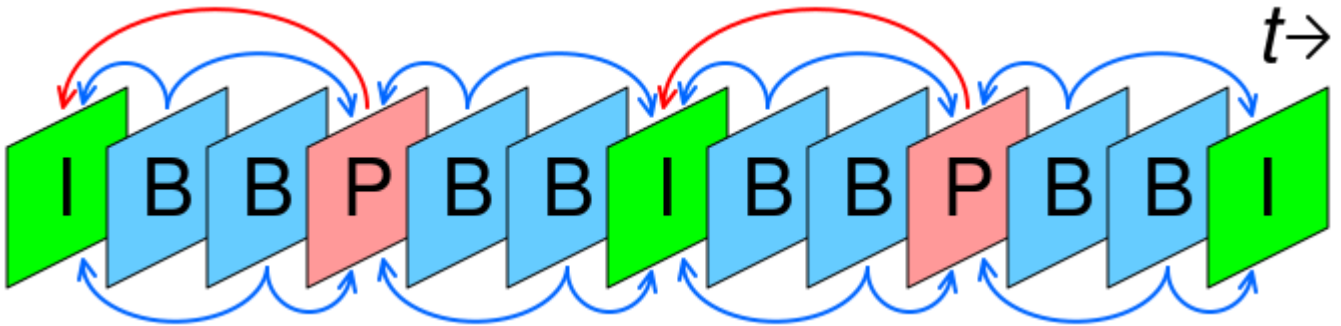
- Stateless decoders need to provide both state information (codec specific) and the compressed frame to the hardware. This means that we need to pass per-frame configuration data. The Request API was designed for that purpose.
- Based on a less-general API called the Configuration Store API: has been used for stateless decoders by ChromeOS for several years, but never made it into the kernel.
- Basic concept (borrowed from Android CameraHAL): create a request object, add any configuration data and video buffers to it, then queue the request to the driver which will then process it. When the request has been processed, an event is generated and the application can retrieve the result from the request object.
- Currently only used by stateless decoders, but will be eventually be extended to camera pipelines as well.
- Stateless decoders expect that the state is set through controls. Different controls are created for different codecs (MPEG-2, H.264, etc.) and are hardware independent.

Userspace API for Stateless Decoders

- Two device nodes are involved: a /dev/mediaX device is used to create Request objects and the /dev/videoX device is used to stream buffers to/from the decoder.
- Userspace is responsible for parsing the headers of the bytestream.
- Create a Request object (usually one for every allocated buffer):

```
ioctl(media_fd, MEDIA_IOC_REQUEST_ALLOC, &request_fd);
```

Userspace API for Stateless Decoders



I-Frame: Intra-coded picture
P-Frame: Predicted picture
B-Frame: Bidirectional predicted picture

Picture from https://en.wikipedia.org/wiki/Inter_frame, Cmglee [CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)]

Userspace API for Stateless Decoders

- Queue the output buffer containing the compressed data of a single frame to the Request object. Tag the output buffer and obtain the reference.

```
struct v4l2_buffer out_buf;
...
out_buf.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
out_buf.request_fd = request_fd;
out_buf.flags = V4L2_BUF_FLAG_REQUEST_FD;
out_buf.timestamp.tv_sec = buf_tag;
out_buf.timestamp.tv_usec = 0;
__u64 buf_ref_ts = v4l2_timeval_to_ns(&out_buf.timestamp);
ioctl(video_fd, VIDIOC_QBUF, &out_buf);
```

Userspace API for Stateless Decoders

```
struct v4l2_mpeg2_sequence {
        /* ISO/IEC 13818-2, ITU-T Rec. H.262: Sequence header */
        __u16   horizontal_size;
        __u16   vertical_size;
        __u32   vbv_buffer_size;

        /* ISO/IEC 13818-2, ITU-T Rec. H.262: Sequence extension */
        __u16   profile_and_level_indication;
        __u8    progressive_sequence;
        __u8    chroma_format;
};

struct v4l2_mpeg2_picture {
        /* ISO/IEC 13818-2, ITU-T Rec. H.262: Picture header */
        __u8    picture_coding_type;

        /* ISO/IEC 13818-2, ITU-T Rec. H.262: Picture coding extension */
        __u8    f_code[2][2];
        __u8    intra_dc_precision;
        __u8    picture_structure;
        __u8    top_field_first;
        __u8    frame_pred_frame_dct;
        __u8    concealment_motion_vectors;
        __u8    q_scale_type;
        __u8    intra_vlc_format;
        __u8    alternate_scan;
        __u8    repeat_first_field;
        __u16   progressive_frame;
};
```

```
struct v4l2_ctrl_mpeg2_slice_params {
        __u32   bit_size;
        __u32   data_bit_offset;
        __u64   backward_ref_ts;
        __u64   forward_ref_ts;

        struct v4l2_mpeg2_sequence sequence;
        struct v4l2_mpeg2_picture picture;

        /* ISO/IEC 13818-2, ITU-T Rec. H.262: Slice */
        __u32   quantiser_scale_code;
};



#define V4L2_CID_MPEG_VIDEO_MPEG2_SLICE_PARAMS \
          (V4L2_CID_MPEG_BASE+250)
```

Userspace API for Stateless Decoders

- All data in the state controls must relate to the corresponding codec standard.
- These controls are currently not part of the public kernel API until we've implemented them in more drivers and have more confidence that we didn't forget anything. H.264 in particular can expect some more revisions. MPEG-2/VP8 seems pretty solid.
- Applications fill in the data based on the contents of the parsed headers from the bytestream.
- The same controls are valid for all stateless decoders.
- If there are HW-decoder-specific controls, then those shall be defined separately.

Userspace API for Stateless Decoders

- Add the state information to the Request object:

```
struct v4l2_ctrl_mpeg2_slice_params params;
struct v4l2_ext_control ctrl = {};
struct v4l2_ext_controls ctrls = {};

// fill in params
params.backward_ref_ts = some_buf_ref_ts;
ctrl.id = V4L2_CID_MPEG_VIDEO_MPEG2_SLICE_PARAMS;
ctrl.ptr = &params;
ctrl.size = sizeof(param);
ctrls.controls = &ctrl;
ctrls.count = 1;
ctrls.which = V4L2_CTRL_WHICH_REQUEST_VAL;
ctrls.request_fd = request_fd;
ioctl(video_fd, VIDIOC_S_EXT_CTRLS, &ctrls);
```

Userspace API for Stateless Decoders

- Queue the capture buffer that will contain the result:

```
ioctl(video_fd, VIDIOC_QBUF, &cap_buf);
```

- Queue the Request for the output buffer:

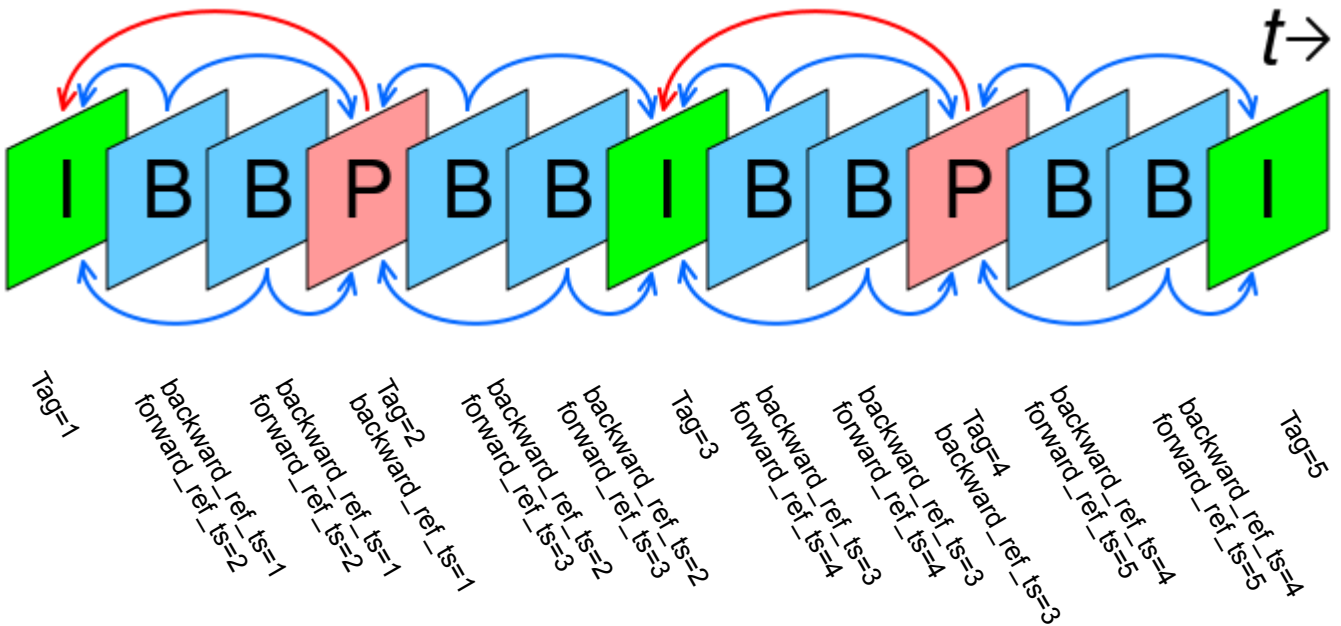
```
ioctl(request_fd, MEDIA_REQUEST_IOC_QUEUE, NULL);
```

- Wait for an event from request_fd signaling that the request has completed.
- Dequeue the buffers:

```
ioctl(video_fd, VIDIOC_DQBUF, &out_buf);
ioctl(video_fd, VIDIOC_DQBUF, &cap_buf);
```

- `cap_buf.timestamp == out_buf.timestamp` and `buf_ref_ts` refers to this capture buffer.

Userspace API for Stateless Decoders



Picture from https://en.wikipedia.org/wiki/Inter_frame, Cmglee [CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)]

H.264/HEVC Decoder Slicing Support

- Slicing: a single frame is divided into N slices, and each slice is encoded/decoded independently. This reduces latency.
- Two types of stateless decoders: those that decode each slice into a buffer and rely on userspace to copy the data into an final frame buffer, combining all slices that build up the frame.
  And those that do this in the driver and can DMA the decoded slice straight into the right position of the capture buffer. This requires API support:
- New capability flag to signal support for this feature (depends on the current output format):

  `V4L2_BUF_CAP_SUPPORTS_M2M_HOLD_CAPTURE_BUF`

  New output buffer flag to hold back the capture buffer until a new frame starts:

  `V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF`

  New decoder command to flush the last held capture buffer:

  `V4L2_DEC_CMD_FLUSH`

Testing

Virtual Drivers

- vivid: emulates webcam/TV/S-Video/HDMI video/vbi capture/output, radio receivers/modulator, software defined radio receiver, HDMI CEC. Under development: metadata capture/output, v4l-touch capture, software defined radio transmitter.
- vim2m: emulates simple video memory-to-memory processing device (such as an RGB to YUV converter).
- vimc: emulates a complex camera pipeline.
- vicodec: emulates a stateful encoder, a stateful decoder and a stateless decoder. Preliminary code for a stateless encoder has been posted, but not merged yet since we do not have a real stateless HW encoder yet.

Testing with Virtual Drivers

- We want to have better quality control over core media changes in order to avoid regressions. Created a new test-media script to do regression tests for V4L2 and CEC. The vivid tests are now run as part of the kernel CI tests (kernelci.org).
- Good compliance tests also make it much easier to write high-quality drivers.
- Developing a new API, the documentation, and testing the new API using virtual drivers is very useful in finding corner cases in the API that you missed.
- syzbot/syzkaller: fuzzing checker from Google. Uses the virtual drivers for testing.

Resources

- Linux Media Infrastructure API: https://linuxtv.org/docs.php
- Version with Codec specification: https://hverkuil.home.xs4all.nl/codec-api/
- Upstream media git repository: http://git.linuxtv.org/media_tree.git
- Cedrus test utility for the stateless decoder: https://github.com/bootlin/v4l2-request-test
- v4l-utils git repository: http://git.linuxtv.org/v4l-utils.git
- linux-media mailinglist & irc channel: http://linuxtv.org/lists.php
- https://linuxtv.org/wiki/index.php/Media_Open_Source_Projects:_Looking_for_Volunteers
- email: hverkuil@xs4all.nl

Questions?