SILICON SIGNALS

IF ZEPHYR WANTS TO POWER AI CAMERAS, WHAT MUST CHANGE?

#OSSUMMIT

About Me...

- **Rutvij Trivedi -** Co-Founder & M.D. Silicon Signals Pvt. Ltd.
- Silicon Signals - Camera ODM
- Built a team of contributors - camera focused
- Hands on - camera products building

Today’s Talk

01 Zephyr Video API Today

02 Gaps in Zephyr's Video API

03 Two Solutions We are Proposing

04 Q&A

What the Zephyr Video API Looks like Today

→ **struct video_buffer** - no flags, no sequence number, no metadata pointer. That is the complete struct today in *video.h*

```c
/**
 * @brief Video buffer structure
 *
 * Represents a video frame.
 */
struct video_buffer {
    /** Pointer to driver specific data. */
    /* It must be kept as first field of the struct if used for @ref k_fifo APIs. */
    void *driver_data;
    /** type of the buffer */
    enum video_buf_type type;
    /** type of the buffer memory, see @ref video_buf_memory */
    uint8_t memory;
    /** pointer to the start of the buffer. */
    uint8_t *buffer;
    /** index of the buffer in the video buffer pool */
    uint16_t index;
    /** size of the buffer in bytes. */
    uint32_t size;
    /** number of bytes occupied by the valid data in the buffer. */
    uint32_t bytesused;
    /** time reference in milliseconds at which the last data byte was
     * actually received for input endpoints or to be consumed for output
     * endpoints.
     */
    uint32_t timestamp;
    /** Line offset within frame this buffer represents, from the
     * beginning of the frame. This offset is given in pixels,
     * so `line_offset` * `pitch` provides offset from the start of
     * the frame in bytes.
     */
    uint16_t line_offset;
};
```

```c
enum video_buf_type {
    /** input buffer type */
    VIDEO_BUF_TYPE_INPUT = 1,
    /** output buffer type */
    VIDEO_BUF_TYPE_OUTPUT = 2,
};
```

Gaps in Zephyr's Video API

- **No metadata path for inference results :** enum video_buf_type has two values. There is no way to carry inference results alongside the frame they came from. Many implementations rely on global variables or application-managed correlation between frames and inference results. Bounding boxes can land on the wrong frame.
- **No streaming states for AI pipelines :** video_stream_stop() cancels everything immediately. If the NPU is mid-inference the buffer is yanked. No graceful drain exists.
- **No per-buffer callback :** The API is pull-based. CPU must wake for every frame. No event-driven pipeline is possible, context switch headache
- **power-aware glue between PM and video subsystems :** *video_set_frmival()* and *pm_notifier* exist but never talk to each other. Camera streams at full 30fps during thermal throttle or idle state. No mechanism to coordinate frame rate with power state safely from driver context.

Gap : Metadata stream

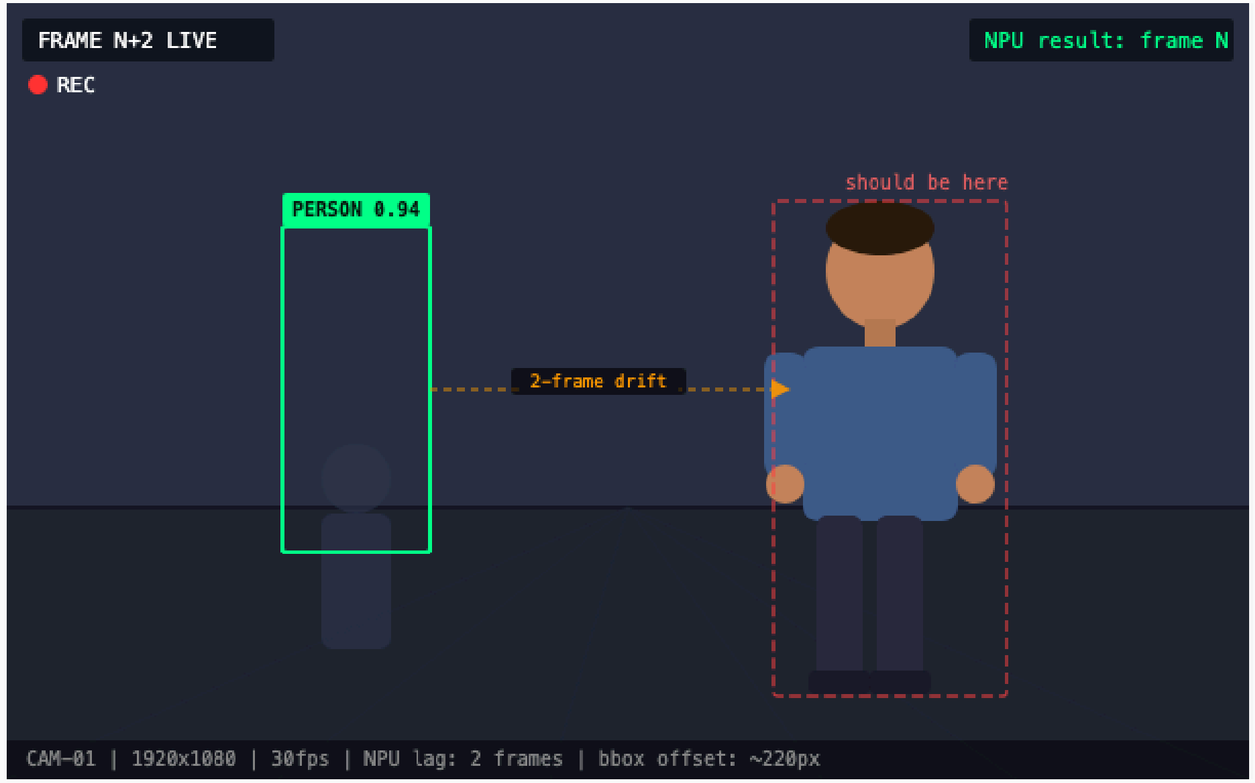
Typical Application Pattern Today

```
static struct infer_result last_result;
static uint32_t last_frame;

void npu_done(struct infer_result *r) {
    last_result = *r;       /* race */
    last_frame  = frame_count;
}

void display(struct video_buffer *buf) {
    if (last_frame == frame_count)
        draw_bbox(buf, &last_result);
        /* pray they match */
}
```

Gap : Metadata stream



- *NPU finishes frame N. Display is on frame N+2. Bounding box on the wrong frame. Every time. Frame and result are strangers - the API never introduced them.*

- *This is not a race condition. It is structural. Frame and result are strangers. The API never introduced them.*

Gap : Metadata stream

- Linux has *V4L2_BUF_TYPE_META_CAPTURE* for ISP statistics.
- ISP statistics travel alongside frames matched by sequence number.
- libcamera's IPA reads those statistics and applies 3A control per frame.
- Zephyr has no equivalent metadata path today.
- We are not aware of an equivalent standard abstraction for AI inference metadata in embedded camera pipelines.

Our Proposal - VIDEO_BUF_TYPE_META + video_infer_result

```
/* new buffer type */
enum video_buf_type {
   VIDEO_BUF_TYPE_INPUT = 1,
   VIDEO_BUF_TYPE_OUTPUT = 2,
   VIDEO_BUF_TYPE_META  = 3, /* NEW */ - RFC #110810 */
};

/* ISP statistics - like Linux IPA via META_CAPTURE */
struct video_isp_stats {
   uint32_t ae_luma_avg;
   uint32_t awb_r_gain;
   uint32_t awb_b_gain;
   uint16_t af_sharpness;
   uint8_t ae_converged;
};
```

Our Proposal - VIDEO_BUF_TYPE_META + video_infer_result

```
/* inference result - does not exist anywhere else */
struct video_infer_result {
   float   confidence;
   uint8_t class_id;
   uint8_t valid;
   uint16_t bbox[4];  /* x, y, width, height */
   void   *userdata; /* extension for richer outputs */
};

/* sidecar - binds both to the frame */
struct video_frame_meta {
   uint32_t sequence;
   struct video_isp_stats   isp;
   struct video_infer_result infer;
};
```

```
/* one pointer added to video_buffer */
struct video_buffer {
   /* existing fields unchanged */
   struct video_frame_meta *meta; /* NULL = unused
};
```

Our Proposal - Driver : binding at DMA completion

```
/* driver - at frame completion, before signalling consumer */

struct video_frame_meta *m = &data->meta_pool[vbuf->index];
m->sequence       = vbuf->sequence;
m->isp.ae_converged = 1;
m->infer.valid     = 0; /* NPU not run yet */
vbuf->meta = m;
/* now hand buffer to consumer */
```

Our Proposal - App: frame and result as the same object

```
video_dequeue(dev, &vbuf, K_FOREVER);

/* AE not settled - skip inference */
if (vbuf->meta && !vbuf->meta->isp.ae_converged) {
   video_enqueue(dev, vbuf);
   return;
}

/* run NPU - result attaches to same object as the frame */
npu_run(vbuf->buffer, &vbuf->meta->infer);
vbuf->meta->infer.valid = 1;
                                        /* frame and bbox are the same object - no global, no race */
                                        if (vbuf->meta->infer.confidence > THRESHOLD)
                                          draw_bbox(vbuf->buffer, vbuf->meta->infer.bbox);

                                        video_enqueue(dev, vbuf);
```

Gap : Per-Buffer Callback

Today - pull-based, CPU wakes for every frame

```
/* today - CPU wakes for every frame */
while (1) {
   video_dequeue(cam, &buf, K_FOREVER);
   /* CPU woke up - 30 times per second */
   process(buf);
   video_enqueue(cam, buf);
}
```

Our Proposal : video_set_buffer_cb()

- typedef void (*video_buffer_cb_t)(
  const struct device *dev,
  struct video_buffer *buf,
  void *user_data);

- int video_set_buffer_cb(const struct device *dev,
  video_buffer_cb_t cb,
  void *user_data);

After : event-driven, CPU sleeps

```
/* after - CPU sleeps */
video_set_buffer_cb(cam, on_frame_ready, &ctx);
video_stream_start(cam, VIDEO_BUF_TYPE_OUTPUT);
k_sleep(K_FOREVER); /* CPU sleeps */

/* driver fires this via k_work */
void on_frame_ready(const struct device *dev,
                    struct video_buffer *buf,
                    void                *user_data) {
  npu_submit(buf);
  video_enqueue(dev, buf);
}
```

Driver implementation : k_work pattern safe from ISR

```
/* driver - DMA completion ISR */
if (pipe->frame_cb) {
   k_work_submit(&pipe->frame_work);
} else {
   k_fifo_put(&pipe->fifo_out, pipe->active);
}

/* work handler - safe thread context */
void dcmipp_frame_work_handler(struct k_work *work) {
   pipe->frame_cb(pipe->dev,
                  pipe->active,
                  pipe->frame_cb_data);
}
```

Why Kernel, Not Application or Library

VIDEO_BUF_TYPE_META + video_infer_result

Today every developer stores the inference result in a global variable and hopes it matches the right frame. That hope breaks at 30fps.

The only way to guarantee the result is bound to the correct frame is to attach it inside the driver at the moment the frame completes - before any application code runs.

Per-buffer callback

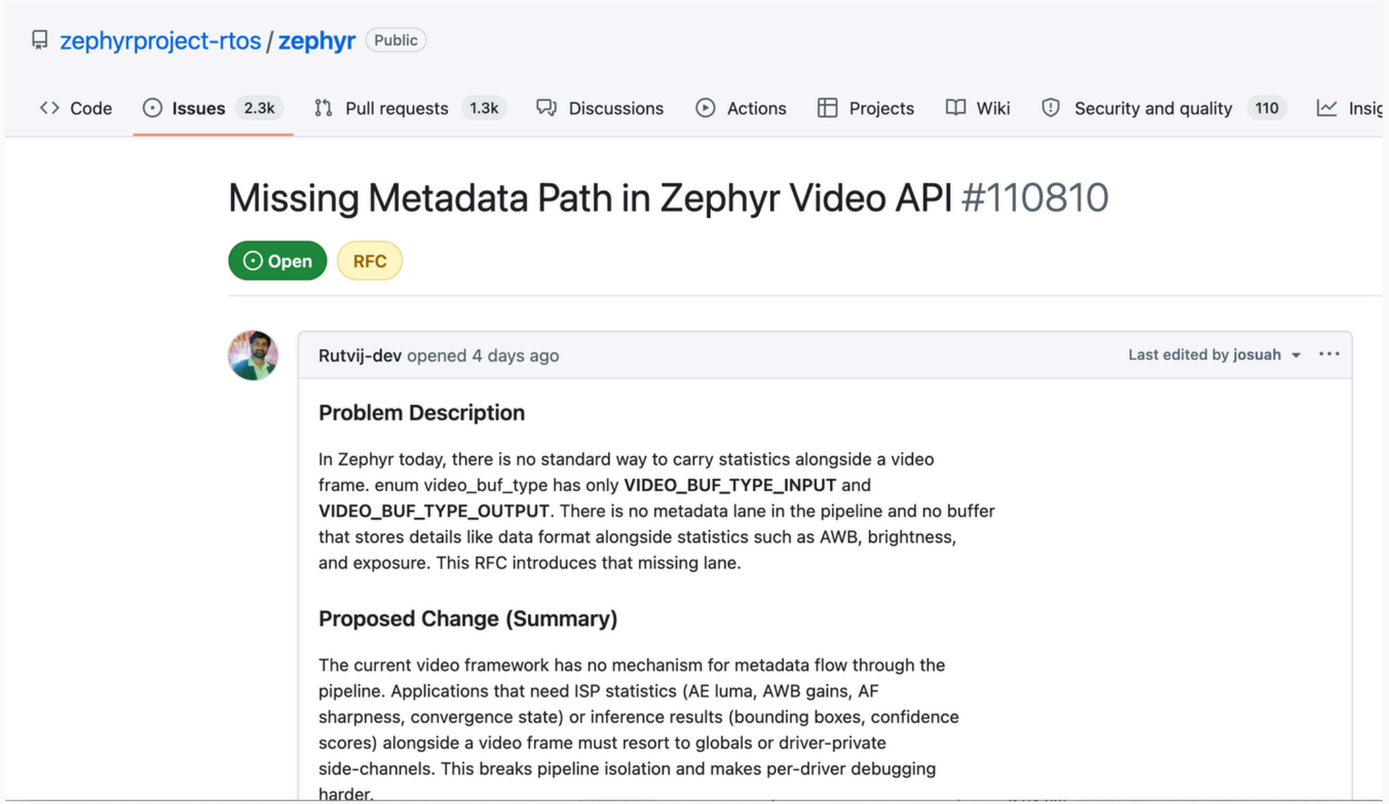
k_poll_signal tells you "a frame is done" - but not which one. With four buffers in flight you are guessing. The callback hands you the exact buffer that just completed.

You always know which frame you are working with.

The k_work deferral keeps it safe - callback runs in thread context, not ISR. Safe for locking, logging, and NPU submission.

RFC - Missing Metadata Path in Zephyr Video API

Step by step - RFC - #110810



zephyrproject-rtos / zephyr Public

Code | Issues 2.3k | Pull requests 1.3k | Discussions | Actions | Projects | Wiki | Security and quality 110 | Insig

Missing Metadata Path in Zephyr Video API #110810

Open RFC

Rutvij-dev opened 4 days ago — Last edited by josuah

Problem Description

In Zephyr today, there is no standard way to carry statistics alongside a video frame. enum video_buf_type has only **VIDEO_BUF_TYPE_INPUT** and **VIDEO_BUF_TYPE_OUTPUT**. There is no metadata lane in the pipeline and no buffer that stores details like data format alongside statistics such as AWB, brightness, and exposure. This RFC introduces that missing lane.

Proposed Change (Summary)

The current video framework has no mechanism for metadata flow through the pipeline. Applications that need ISP statistics (AE luma, AWB gains, AF sharpness, convergence state) or inference results (bounding boxes, confidence scores) alongside a video frame must resort to globals or driver-private side-channels. This breaks pipeline isolation and makes per-driver debugging harder.

Closing & Credits

> ***The goal is not to match Linux feature for feature. The goal is the right abstraction at the right layer - lightweight enough for Zephyr, expressive enough for the next generation of AI cameras.***

- Elgin Perumbilly - elgin.perumbilly@siliconsignals.io
- josuah - josuah.demangeon - Discord - 453536439565680640

Q&A

Any questions?
Ask away!

Give us a chance to address your doubts or concerns. Let’s open up the discussion with your ideas!