Merging the V4L2 streams support

Media Summit 2022
Dublin, Ireland

Laurent Pinchart
laurent.pinchart@ideasonboard.com

Problem Statement

V4L2 lacks support for

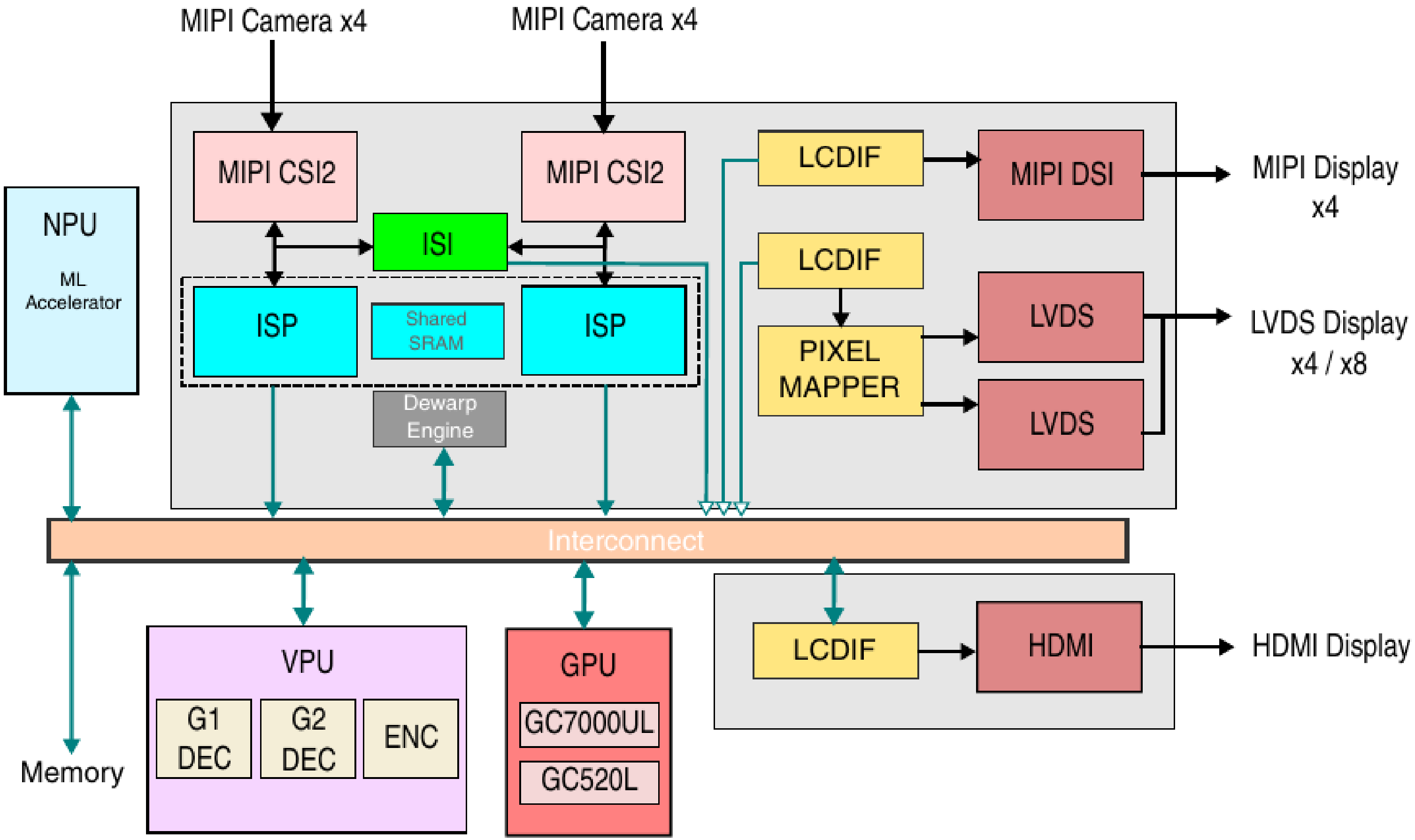
- CSI-2 virtual channels and data types

  Conceptually, this also affects other buses that can carry multiple streams, MIPI CSI-2 is only the most common example.

- Crossbar switches

  This can be generalized as missing support for routing internal to media entities and V4L2 subdevs.

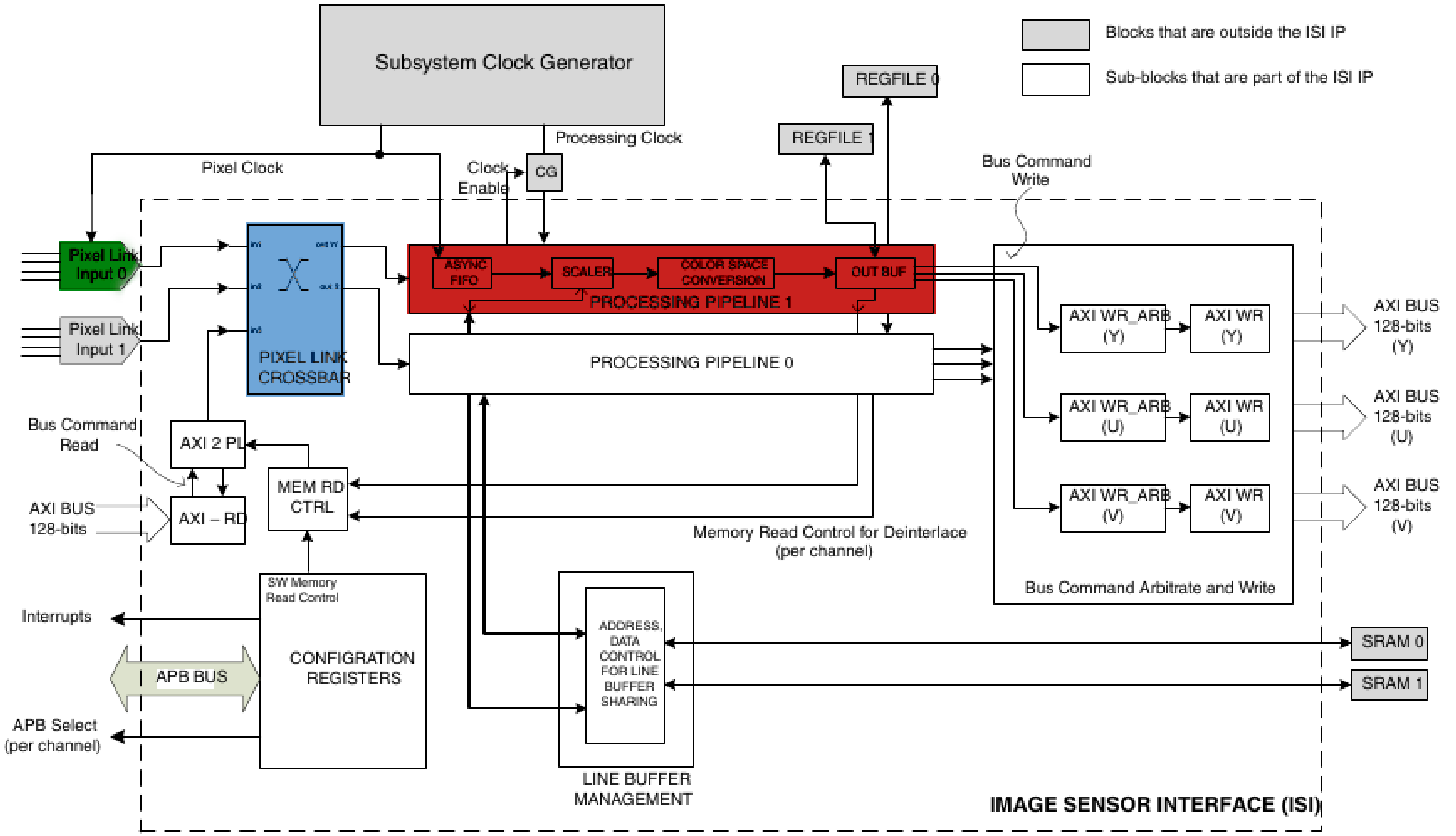
1st – Internal Routing



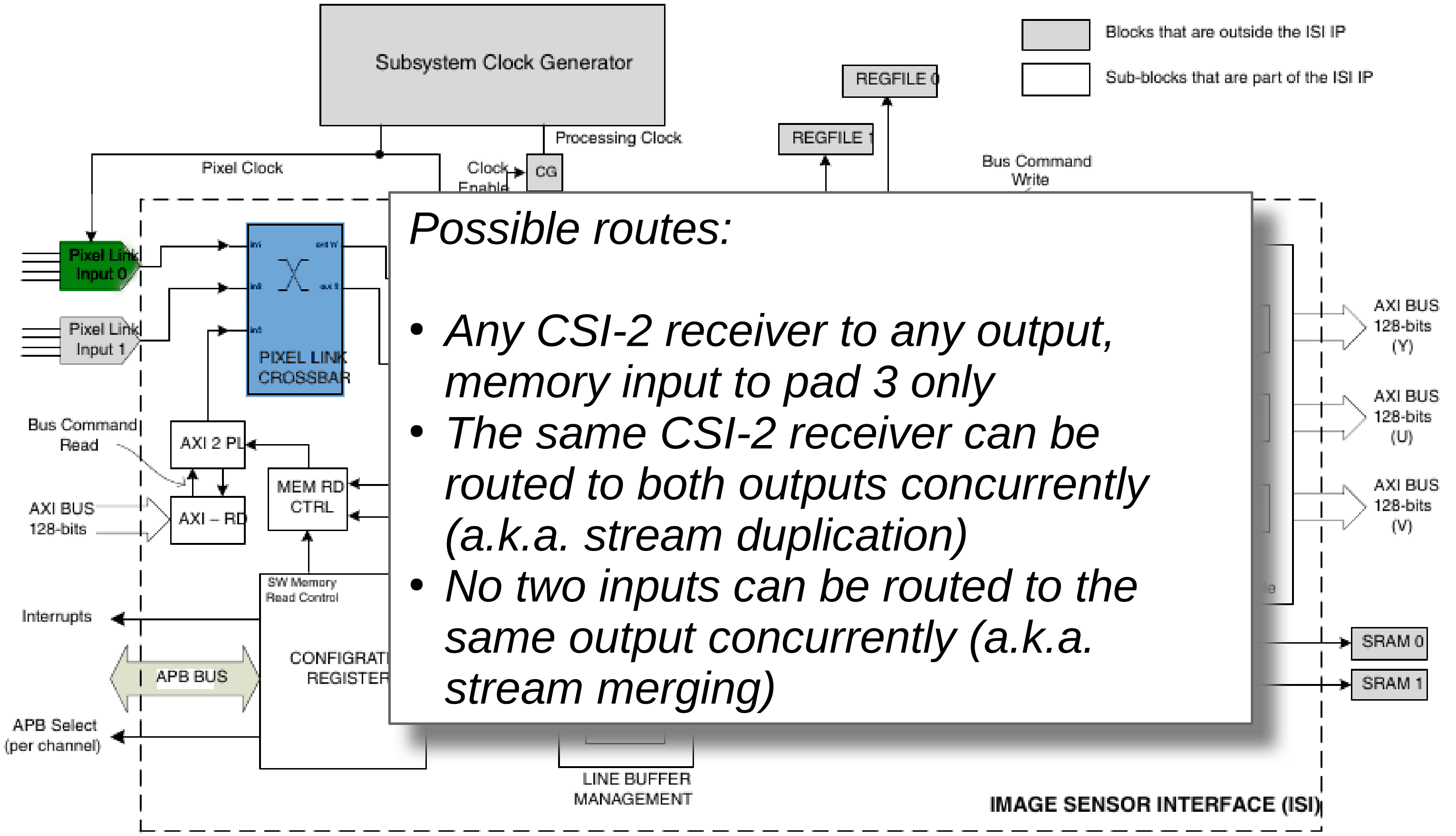
i.MX 8M Plus Applications Processor Reference Manual, Rev. 1, 06/2021

i.MX8MP Hardware Architecture

Image Sensing Interface (ISI)



i.MX 8M Plus Applications Processor Reference Manual, Rev. 1, 06/2021

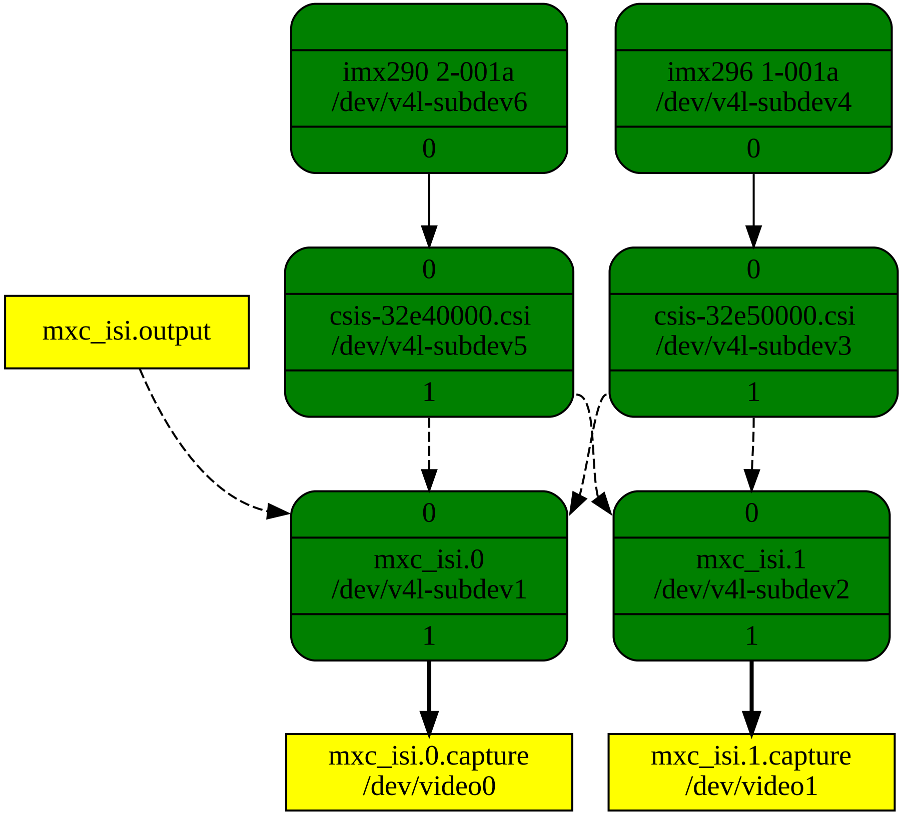


Possible routes:

- *Any CSI-2 receiver to any output, memory input to pad 3 only*
- *The same CSI-2 receiver can be routed to both outputs concurrently (a.k.a. stream duplication)*
- *No two inputs can be routed to the same output concurrently (a.k.a. stream merging)*

i.MX 8M Plus Applications Processor Reference Manual, Rev. 1, 06/2021

Image Sensing Interface (ISI)

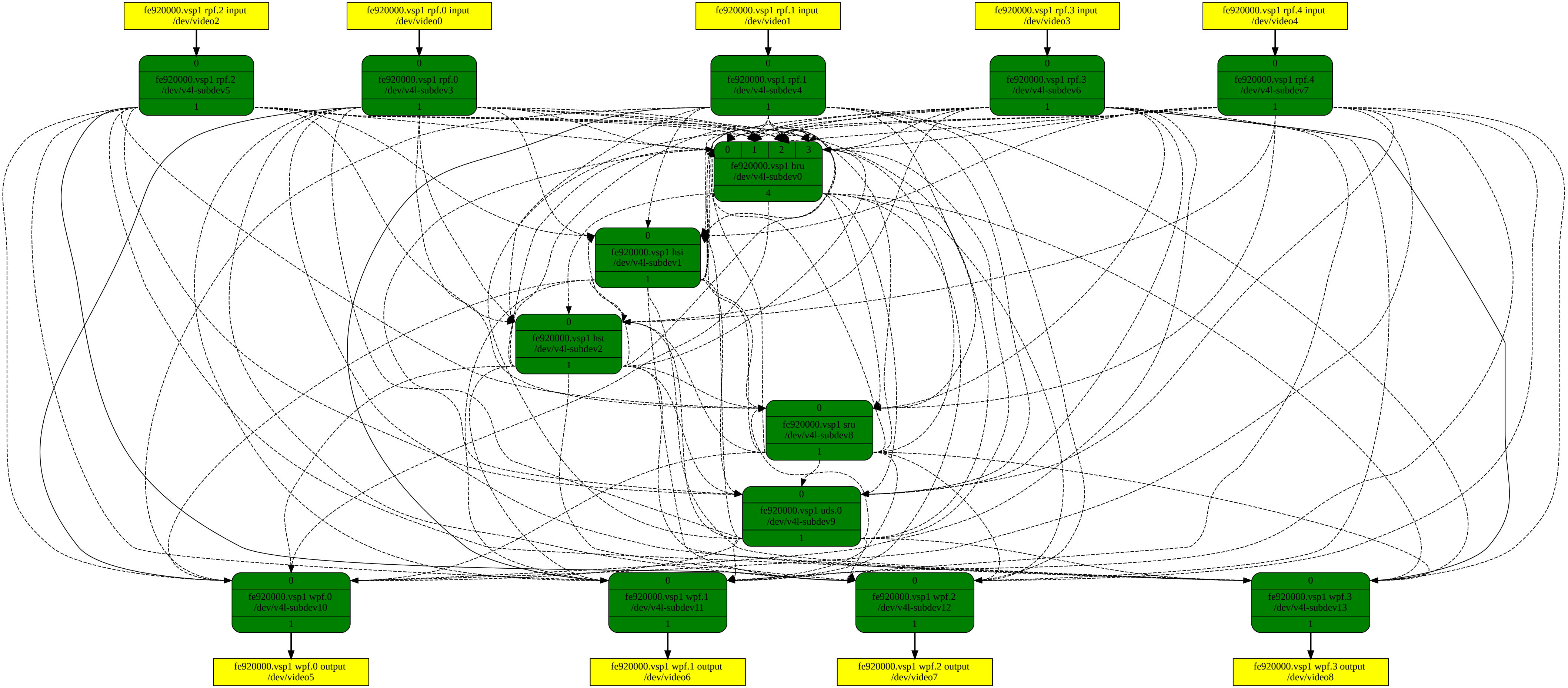


The routing can be modelled with media controller links.

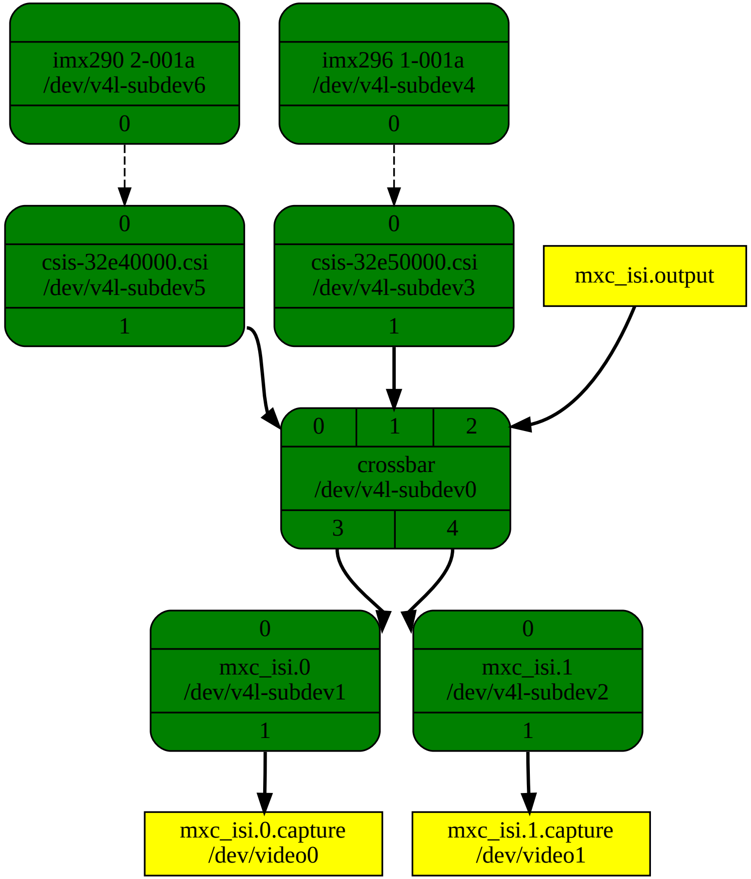
Problems:

- *No way to enumerate supported options*
- *It doesn’t scale*

Crossbar Switch – Links



Told You, It doesn’t scale (and that’s in mainline)



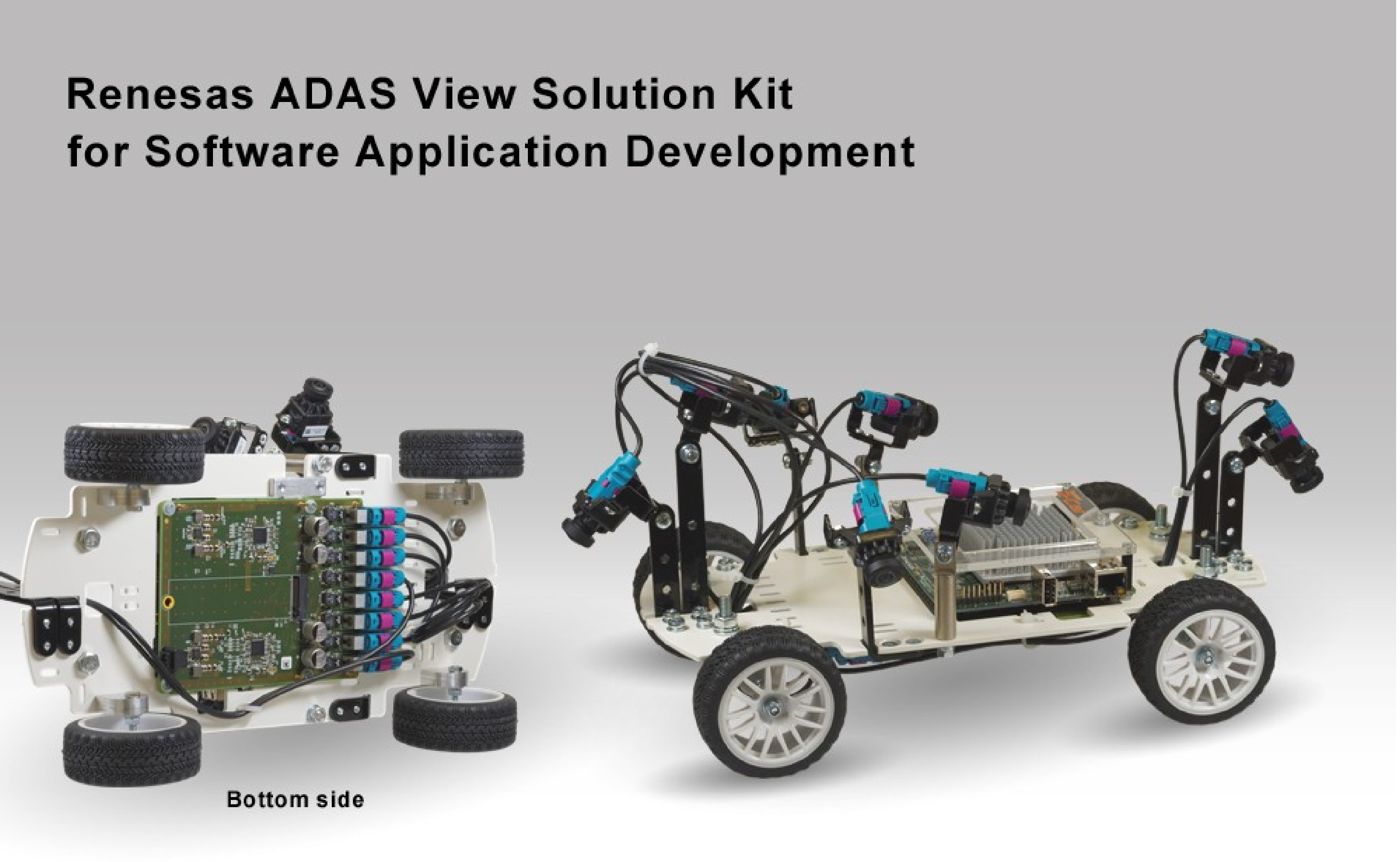
Proposed solution:

- *One entity and V4L2 subdev for the crossbar switch*
- *New ioctls for the V4L2 subdev userspace API to expose internal routing*

Crossbar Switch – Internal Routing

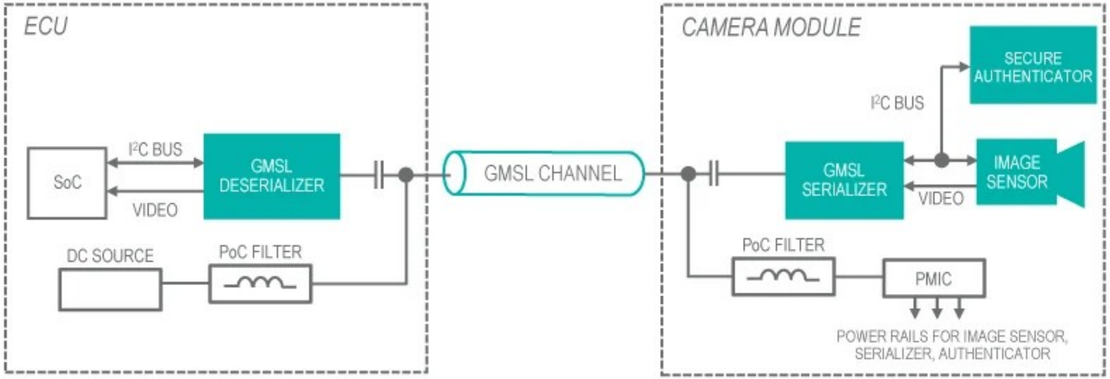
2nd – Streams

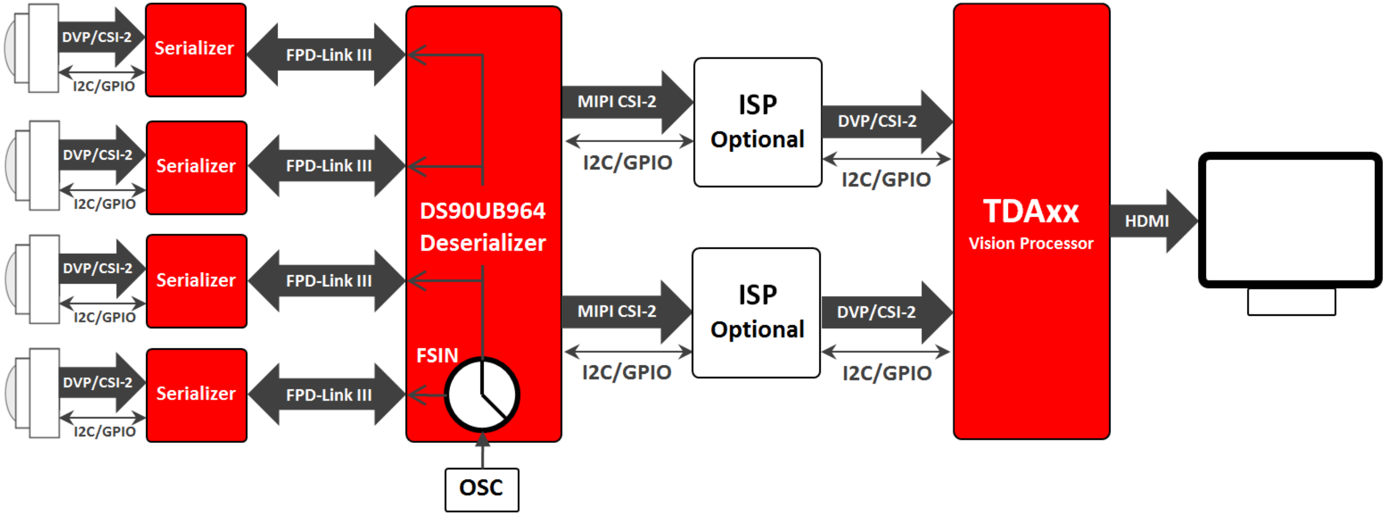
ADAS Surround View – 8 Cameras


Renesas ADAS View Solution Kit
for Software Application Development
Bottom side

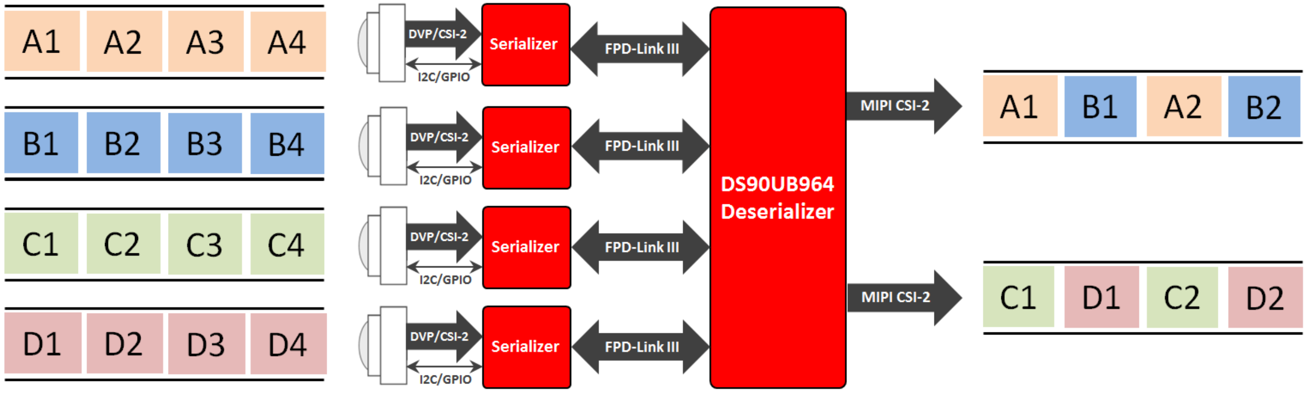
The world is consuming more and more video streams on a single system.

Camera Long Range Connectivity



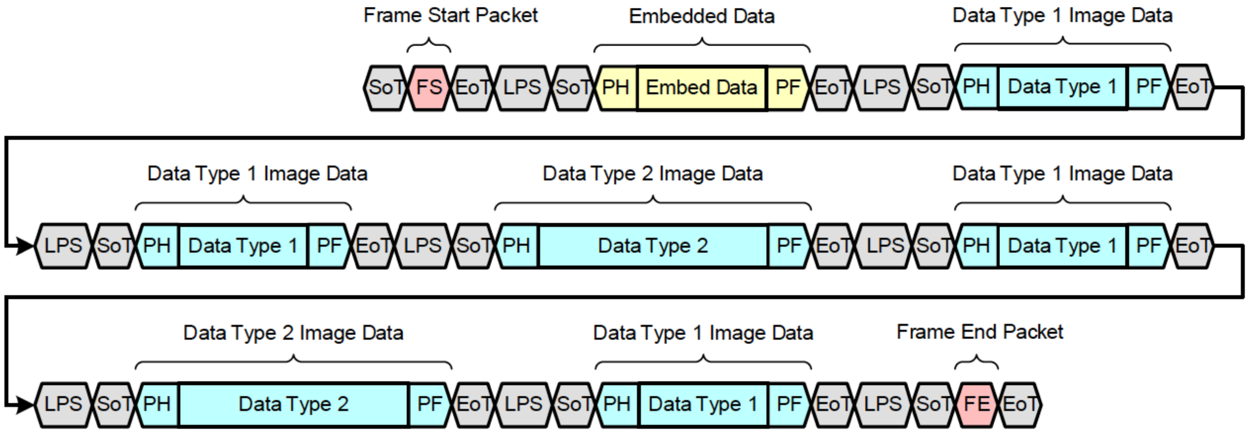


Long range digital connectivity solutions exist (GMSL, FPD-Link, MIPI A-PHY, …).



On the receiving side, MIPI CSI-2 is the most common local bus, simplifying connectivity with its support for multiplexed virtual channels.

MIPI CSI-2 Virtual Channels


Frame Start Packet
Embedded Data
Data Type 1 Image Data
SoT FS EoT LPS SoT PH Embed Data PF EoT LPS SoT PH Data Type 1 PF EoT
Data Type 1 Image Data
Data Type 2 Image Data
Data Type 1 Image Data
LPS SoT PH Data Type 1 PF EoT LPS SoT PH Data Type 2 PF EoT LPS SoT PH Data Type 1 PF EoT
Data Type 2 Image Data
Data Type 1 Image Data
Frame End Packet
LPS SoT PH Data Type 2 PF EoT LPS SoT PH Data Type 1 PF EoT LPS SoT FE EoT

KEY:

LPS – Low Power State
SoT – Start of Transmission
EoT – End of Transmission

PH – Packet Header
PF – Packet Footer

FS – Frame Start Packet
FE – Frame End Packet

Figure 73 PDAF Data with Different Data Type

CSI-2 also supports interleaving multiple data types from one video source. A camera sensor can send image data, PDAF (phase detection auto focus) data and embedded data over the same link.

MIPI CSI-2 Data Types

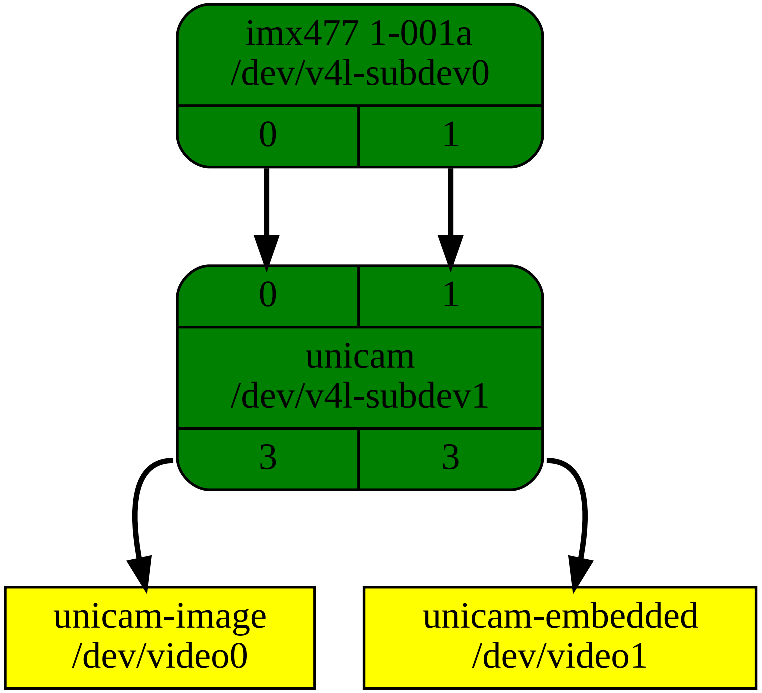
Streams – Device Tree

```
    i2c@xxxx{
        device@xx {
...
            ports {
                #address-cells = <1>;
                #size-cells = <0>;
                port@0 {
                    reg = <0>;
                    dev_out: endpoint {
                        vc-id = <1>;
                        remote-endpoint = <&csi_in0>;
                    };
                };
            };
        };
    };
```

The virtual channel can be selected in the device tree.

Problems:

- *Can’t be selected at runtime*
- *Still supports a single virtual channel only if specified in port node*
- *Doesn’t address exposing streams to userspace*

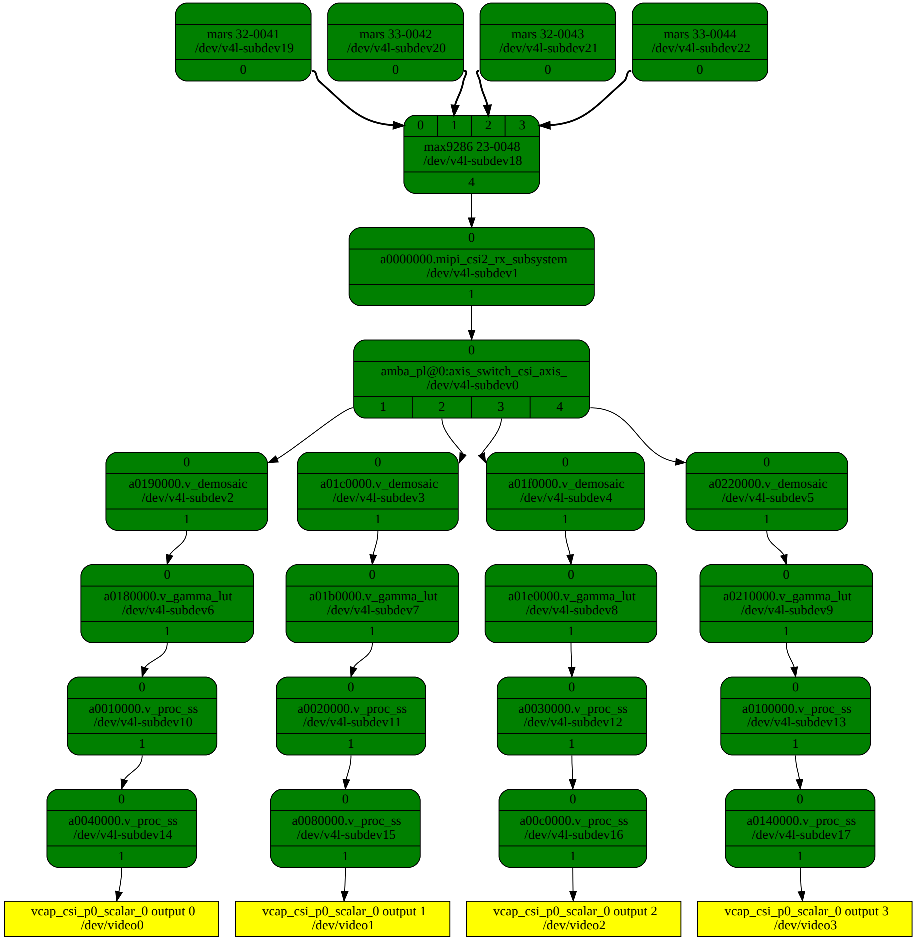


The streams can be modelled with multiple links.

Problems:

- *It doesn’t scale (CSI-2 supports 32 VCs x 64 DTs = 2048 streams)*

Streams – Multiple Links



Proposed solution:

- *Allow links to carry multiple streams*
- *Expose streams to userspace on pads (per-stream formats, selection rectangles, …)*
- *Expose streams to userspace in subdev routing tables*
- *Streams are not dependent on a bus type, keep the streams to VCs/DTs mapping internal to the kernel*

Streams – Native Support Everywhere

Proposal

V4L2 Streams API (v14)

- [PATCH v14 17/34] media: add V4L2_SUBDEV_FL_STREAMS
- [PATCH v14 18/34] media: add V4L2_SUBDEV_CAP_STREAMS
- [PATCH v14 19/34] media: Documentation: Add GS_ROUTING documentation
- [PATCH v14 20/34] media: subdev: Add [GS]_ROUTING subdev ioctls and operations
- [PATCH v14 21/34] media: subdev: add v4l2_subdev_has_pad_interdep()
- [PATCH v14 22/34] media: subdev: add v4l2_subdev_set_routing helper()
- [PATCH v14 23/34] media: subdev: Add for_each_active_route() macro
- [PATCH v14 24/34] media: Documentation: add multiplexed streams documentation
- [PATCH v14 25/34] media: subdev: add stream based configuration
- [PATCH v14 26/34] media: subdev: use streams in v4l2_subdev_link_validate()
- [PATCH v14 27/34] media: subdev: add "opposite" stream helper funcs
- [PATCH v14 28/34] media: subdev: add streams to v4l2_subdev_get_fmt() helper function
- [PATCH v14 29/34] media: subdev: add v4l2_subdev_set_routing_with_fmt() helper
- [PATCH v14 30/34] media: subdev: add v4l2_subdev_routing_validate() helper
- [PATCH v14 31/34] media: v4l2-subdev: Add v4l2_subdev_state_xlate_streams() helper
- [PATCH v14 32/34] media: v4l2-subdev: Add subdev .(enable|disable)_streams() operations
- [PATCH v14 33/34] media: v4l2-subdev: Add v4l2_subdev_s_stream_helper() function
- [PATCH v14 34/34] media: Add stream to frame descriptor

(01/34 to 16/34 already queued for v6.1)

Routing – Userspace API

Internal Routing uAPI (1/3)

```
+/**
+ * struct v4l2_subdev_routing - Subdev routing information
+ *
+ * @which: configuration type (from enum v4l2_subdev_format_whence)
+ * @num_routes: the total number of routes in the routes array
+ * @routes: pointer to the routes array
+ * @reserved: drivers and applications must zero this array
+ */
+struct v4l2_subdev_routing {
+        __u32 which;
+        __u32 num_routes;
+        __u64 routes;
+        __u32 reserved[6];
+};

+#define VIDIOC_SUBDEV_G_ROUTING    _IOWR('V', 38, struct v4l2_subdev_routing)
+#define VIDIOC_SUBDEV_S_ROUTING    _IOWR('V', 39, struct v4l2_subdev_routing)
```

- The API adds support for getting and setting routing tables.
- Setting a routing table overrides the whole configuration (no incremental updates).
- Configurations can be tried using the usual subdev ACTIVE/TRY states.

- Q: Do we need incremental updates ?
- Q: How does userspace enumerate possible routes ?
- Q: More generically, how does userspace query routing restrictions ?

Internal Routing uAPI (2/3)

```
+/**
+ * struct v4l2_subdev_route - A route inside a subdev
+ *
+ * @sink_pad: the sink pad index
+ * @source_pad: the source pad index
+ * @flags: route flags V4L2_SUBDEV_ROUTE_FL_*
+ * @reserved: drivers and applications must zero this array
+ */
+struct v4l2_subdev_route {
+        __u32 sink_pad;
+        __u32 source_pad;
+        __u32 flags;
+        __u32 reserved[5];
+};
```

- A route connects a sink pad to a source pad.

Internal Routing uAPI (3/3)

```
+/* The v4l2 sub-device supports routing and multiplexed streams. */
+#define V4L2_SUBDEV_CAP_STREAMS                        0x00000002
+
+/*
+ * Is the route active? An active route will start when streaming is enabled
+ * on a video node.
+ */
+#define V4L2_SUBDEV_ROUTE_FL_ACTIVE                _BITUL(0)
+
+/*
+ * Is the route a source endpoint? A source endpoint route refers to a stream
+ * generated by the subdevice (usually a sensor), and thus there is no
+ * sink-side endpoint for the route. The sink_pad and sink_stream fields are
+ * unused.
+ * Set by the driver.
+ */
+#define V4L2_SUBDEV_ROUTE_FL_SOURCE                _BITUL(2)
```

- A new capability flag exposes support of the API to userspace.
- A route can be active or inactive (exact meaning not defined yet).
- The source of a route can be a sink pad, or an internal source (e.g. camera sensors).

- Q: What is an inactive route ? (cfr question on previous slide about enumeration)
- Q: Are “source routes” a good idea ?

Routing – Kernel API

Internal Routing kAPI (1/2)

```
+/**
+ * struct v4l2_subdev_krouting - subdev routing table
+ * @num_routes: number of routes
+ * @routes: &struct v4l2_subdev_route
+ *
+ * This structure contains the routing table for a subdev.
+ */
+struct v4l2_subdev_krouting {
+       unsigned int num_routes;
+       struct v4l2_subdev_route *routes;
+};

 struct v4l2_subdev_state {
        /* lock for the struct v4l2_subdev_state fields */
        struct mutex _lock;
        struct mutex *lock;
        struct v4l2_subdev_pad_config *pads;
+       struct v4l2_subdev_krouting routing;
 };
```

- Internal structure to model routing.
- Integrated in v4l2_subdev_state. The whole routing API is heavily based on the subdev state, and requires drivers to use the recent active subdev state API.

Internal Routing kAPI (2/2)

```
 /**
  * struct v4l2_subdev_pad_ops - v4l2-subdev pad level operations
  *
  * [...]
+ * @set_routing: enable or disable data connection routes described in the
+ *               subdevice routing table.
  * [...]
  */
 struct v4l2_subdev_pad_ops {
 [...]
+        int (*set_routing)(struct v4l2_subdev *sd,
+                           struct v4l2_subdev_state *state,
+                           enum v4l2_subdev_format_whence which,
+                           struct v4l2_subdev_krouting *route);
 [...]
 };
```

- New subdev pad operation to set routing.
- Subdev drivers must store the routing table in the state.
- The GET ioctl is fully implemented by the V4L2 subdev core, retrieving the routing table from the state.

Routing – Kernel Helpers

```
+/**
+ * for_each_active_route - iterate on all active routes of a routing table
+ * @routing: The routing table
+ * @route: The route iterator
+ */
+#define for_each_active_route(routing, route) \
+        for ((route) = NULL;                  \
+             ((route) = __v4l2_subdev_next_active_route((routing), (route)));)
```

- Helper to iterate over active routes in a routing table.

Internal Routing Helpers (1/5)

Internal Routing Helpers (2/5)

```
+enum v4l2_subdev_routing_restriction {
+        V4L2_SUBDEV_ROUTING_NO_1_TO_N = BIT(0),
+        V4L2_SUBDEV_ROUTING_NO_N_TO_1 = BIT(1),
+        V4L2_SUBDEV_ROUTING_NO_STREAM_MIX = BIT(2),
+};
+
+/**
+ * v4l2_subdev_routing_validate() - Verify that routes comply with driver constraints
+ * @sd: The subdevice
+ * @routing: Routing to verify
+ * @disallow: Restrictions on routes
+ *
+ * This verifies that the given routing complies with the @disallow constraints.
+ *
+ * Returns 0 on success, error value otherwise.
+ */
+int v4l2_subdev_routing_validate(struct v4l2_subdev *sd,
+                                 const struct v4l2_subdev_krouting *routing,
+                                 enum v4l2_subdev_routing_restriction disallow);
```

- Helper to validate a routing table against common constraints: stream duplication (1:N routing), stream merging (N:1 routing), stream mixing (streams coming on the same pad can be routed to different pads).

Internal Routing Helpers (3/5)

```
+ /**
+ * v4l2_subdev_set_routing() - Set given routing to subdev state
+ * @sd: The subdevice
+ * @state: The subdevice state
+ * @routing: Routing that will be copied to subdev state
+ *
+ * This will release old routing table (if any) from the state, allocate
+ * enough space for the given routing, and copy the routing.
+ *
+ * This can be used from the subdev driver's set_routing op, after validating
+ * the routing.
+ */
+int v4l2_subdev_set_routing(struct v4l2_subdev *sd,
+                            struct v4l2_subdev_state *state,
+                            const struct v4l2_subdev_krouting *routing);
```

- Helper to store a routing table in the state (handles memory allocation).

Internal Routing Helpers (4/5)

```
+/**
+ * v4l2_subdev_set_routing_with_fmt() - Set given routing and format to subdev
+ *                                      state
+ * @sd: The subdevice
+ * @state: The subdevice state
+ * @routing: Routing that will be copied to subdev state
+ * @fmt: Format used to initialize all the streams
+ *
+ * This is the same as v4l2_subdev_set_routing, but additionally initializes
+ * all the streams using the given format.
+ */
+int v4l2_subdev_set_routing_with_fmt(struct v4l2_subdev *sd,
+                                     struct v4l2_subdev_state *state,
+                                     struct v4l2_subdev_krouting *routing,
+                                     const struct v4l2_mbus_framefmt *fmt);
```

- Helper to store a routing table in the state and reset all formats on the corresponding pads (fmt is assumed to be valid).

- Q: How about selection rectangles ? Do we need better helpers ?

```
+/**
+ * v4l2_subdev_routing_find_opposite_end() - Find the opposite stream
+ * @routing: routing used to find the opposite side
+ * @pad: pad id
+ * @stream: stream id
+ * @other_pad: pointer used to return the opposite pad
+ * @other_stream: pointer used to return the opposite stream
+ *
+ * This function uses the routing table to find the pad + stream which is
+ * opposite the given pad + stream.
+ *
+ * @other_pad and/or @other_stream can be NULL if the caller does not need the
+ * value.
+ *
+ * Returns 0 on success, or -EINVAL if no matching route is found.
+ */
+int v4l2_subdev_routing_find_opposite_end(const struct v4l2_subdev_krouting *routing,
+                                          u32 pad, u32 stream, u32 *other_pad,
+                                          u32 *other_stream);
```

- Helper to follow streams inside a subdev.

Internal Routing Helpers (5/5)

Streams – Userspace API

```
 Pipelines and media streams
 ^^^^^^^^^^^^^^^^^^^^^^^^^^^

+A media stream is a stream of pixels or metadata originating from one or more
+source devices (such as a sensors) and flowing through media entity pads
+towards the final sinks. The stream can be modified on the route by the
+devices (e.g. scaling or pixel format conversions), or it can be split into
+multiple branches, or multiple branches can be merged.
+
+A media pipeline is a set of media streams which are interdependent. This
+interdependency can be caused by the hardware (e.g. configuration of a second
+stream cannot be changed if the first stream has been enabled) or by the driver
+due to the software design. Most commonly a media pipeline consists of a single
+stream which does not branch.
```

- In the API, streams are identified by an arbitrary numerical ID. The IDs are link-local, the same stream ID on the source and sink pads of a link refer to the same stream.
- Streams are routed in subdevs using the routing API. The stream ID will typically change when the stream goes through a subdev (no graph-global ID).

Streams uAPI (1/5)

Streams uAPI (2/5)

```
 /**
  * struct v4l2_subdev_route - A route inside a subdev
  *
  * @sink_pad: the sink pad index
+ * @sink_stream: the sink stream identifier
  * @source_pad: the source pad index
+ * @source_stream: the source stream identifier
  * @flags: route flags V4L2_SUBDEV_ROUTE_FL_*
  * @reserved: drivers and applications must zero this array
  */
 struct v4l2_subdev_route {
         __u32 sink_pad;
+        __u32 sink_stream;
         __u32 source_pad;
+        __u32 source_stream;
         __u32 flags;
         __u32 reserved[5];
 };
```

- Streams are created by subdev internal routes. When a route is created with sink and source stream IDs, those streams are implicitly created on the corresponding pads.
- If a subdev doesn’t support the internal routing API, all pads have an implicit stream with ID 0.

- Q: Do we need to support dynamic routing changes (while streaming) ?

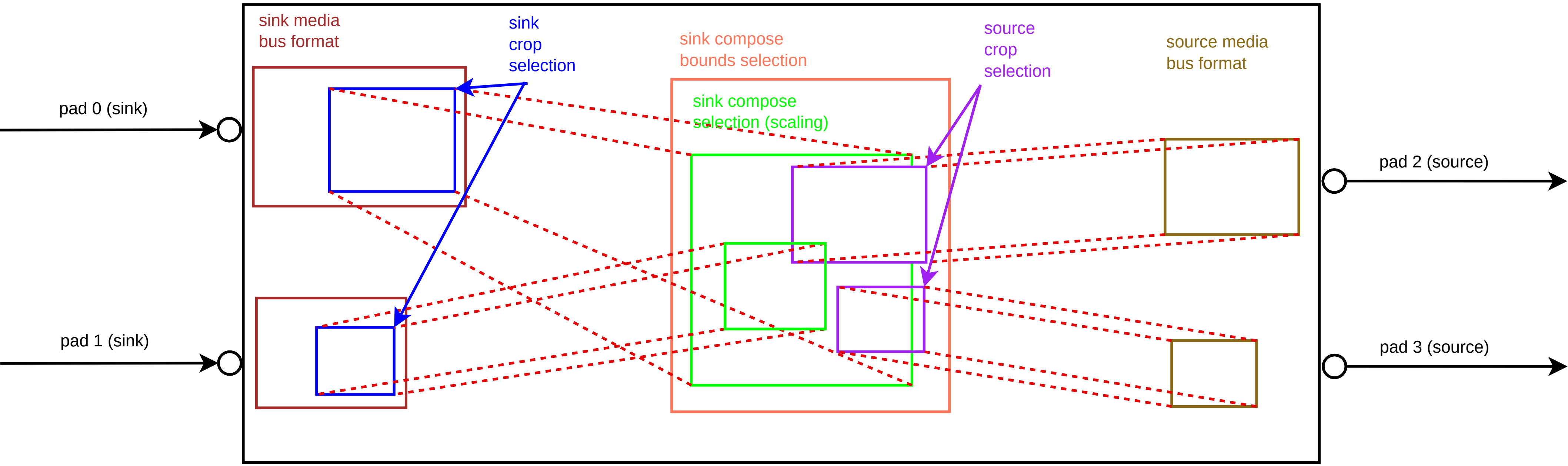
Streams uAPI (3/5)

```
 /**
  * struct v4l2_subdev_format - Pad-level media bus format
  * @which: format type (from enum v4l2_subdev_format_whence)
  * @pad: pad number, as reported by the media API
  * @format: media bus format (format code and frame size)
+ * @stream: stream number, defined in subdev routing
  * @reserved: drivers and applications must zero this array
  */
 struct v4l2_subdev_format {
         __u32 which;
         __u32 pad;
         struct v4l2_mbus_framefmt format;
-        __u32 reserved[8];
+        __u32 stream;
+        __u32 reserved[7];
 };

+ v4l2_subdev_crop, v4l2_subdev_mbus_code_enum, v4l2_subdev_frame_size_enum, v4l2_subdev_frame_interval,
v4l2_subdev_frame_interval_enum and v4l2_subdev_selection
```

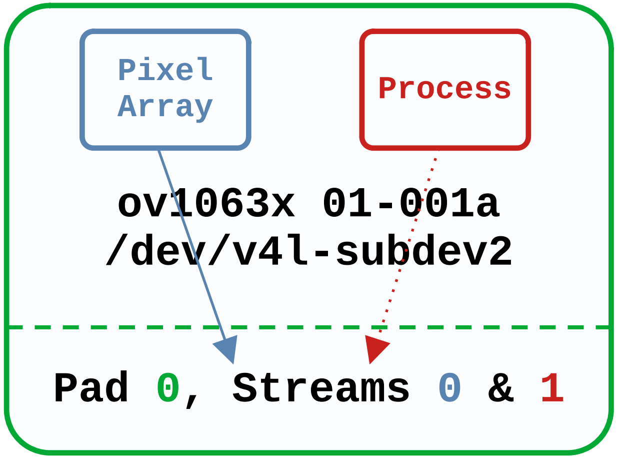
- Streams are exposed to userspace in pad configuration. Pad formats become per-stream.
- Formats are reset when routing is modified.

- Q: Should we skip v4l2_subdev_crop (legacy) ? How about v4l2_subdev_frame_interval (no existing use case) ?
- Q: Should we avoid resetting formats (e.g. to support dynamic routing changes) ?



- Q: Extending the subdev configuration model (Documentation/userspace-api/media/v4l/dev-subdev.rst) for streams hasn't been considered yet. How do we avoid creating a horribly complex monster ?

Streams uAPI (4/5)



```
struct v4l2_subdev_route routes[] = {
        {
                .source_pad = 0,
                .source_stream = 0,
                .flags = V4L2_SUBDEV_ROUTE_FL_SOURCE
                       | V4L2_SUBDEV_ROUTE_FL_ACTIVE,
        },
        {
                .source_pad = 0,
                .source_stream = 1,
                .flags = V4L2_SUBDEV_ROUTE_FL_SOURCE
                    /* | V4L2_SUBDEV_ROUTE_FL_ACTIVE */,
        }
};
```

- A simplified version of the generic subdev model is used with camera sensors.
- The routing table is used to control transmission of embedded data by enabling or disabling the corresponding route.

- Q: Is this the best option ?
- Q: Can it support control of ED when streaming ?
- Q: How does this integrate with other sensor features ?
- Q: Do we need a new model for camera sensors (in-kernel, userspace, or both) ?

Streams uAPI (5/5)

Streams – Kernel API

Streams kAPI (1/4)

```
 /**
  * struct media_entity_operations - Media entity operations
 [...]
+ * @has_pad_interdep:   Return whether two pads of the entity are
+ *                      interdependent. If two pads are interdependent they are
+ *                      part of the same pipeline and enabling one of the pads
+ *                      means that the other pad will become "locked" and
+ *                      doesn't allow configuration changes. pad0 and pad1 are
+ *                      guaranteed to not both be sinks or sources.
+ *                      Optional: If the operation isn't implemented all pads
+ *                      will be considered as interdependent.
 [...]
 */
 struct media_entity_operations {
 [...]
+        bool (*has_pad_interdep)(struct media_entity *entity, unsigned int pad0,
+                                 unsigned int pad1);
 };
```

- Stream-aware .has_pad_interdep() operation (was .has_route() in previous versions) exposes internal routing to the media controller framework, used by media pipeline helpers to walk pipelines based on streams.

Streams kAPI (2/4)

```
+/**
+ * struct v4l2_subdev_stream_config - Used for storing stream configuration.
+ * @pad: pad number
+ * @stream: stream number
+ * @enabled: has the stream been enabled with v4l2_subdev_enable_stream()
+ * @fmt: &struct v4l2_mbus_framefmt
+ * @crop: &struct v4l2_rect to be used for crop
+ * @compose: &struct v4l2_rect to be used for compose
+ *
+ * This structure stores configuration for a stream.
+ */
+struct v4l2_subdev_stream_config {
+        u32 pad;
+        u32 stream;
+        bool enabled;
+        struct v4l2_mbus_framefmt fmt;
+        struct v4l2_rect crop;
+        struct v4l2_rect compose;
+};
```

- Structure to store per-stream pad configuration.

Streams kAPI (3/4)

```
+/**
+ * struct v4l2_subdev_stream_configs - A collection of stream configs.
+ *
+ * @num_configs: number of entries in @config.
+ * @configs: an array of &struct v4l2_subdev_stream_configs.
+ */
+struct v4l2_subdev_stream_configs {
+        u32 num_configs;
+        struct v4l2_subdev_stream_config *configs;
+};

 struct v4l2_subdev_state {
         /* lock for the struct v4l2_subdev_state fields */
         struct mutex _lock;
         struct mutex *lock;
         struct v4l2_subdev_pad_config *pads;
         struct v4l2_subdev_krouting routing;
+        struct v4l2_subdev_stream_configs stream_configs;
 };
```

- Integrated in v4l2_subdev_state. The whole streams API is heavily based on the subdev state, and requires drivers to use the recent active subdev state API.

Streams kAPI (4/4)

```
 /**
  * struct v4l2_subdev_pad_ops - v4l2-subdev pad level operations
  *
  * [...]
+ * @enable_streams: Enable the streams defined in streams_mask on the given
+ *        source pad. Subdevs that implement this operation must use the active
+ *        state management provided by the subdev core (enabled through a call to
+ *        v4l2_subdev_init_finalize() at initialization time). Do not call
+ *        directly, use v4l2_subdev_enable_streams() instead.
+ *
+ * @disable_streams: Disable the streams defined in streams_mask on the given
+ *        source pad. Subdevs that implement this operation must use the active
+ *        state management provided by the subdev core (enabled through a call to
+ *        v4l2_subdev_init_finalize() at initialization time). Do not call
+ *        directly, use v4l2_subdev_disable_streams() instead.
  * [...]
  */
 struct v4l2_subdev_pad_ops {
 [...]
+        int (*enable_streams)(struct v4l2_subdev *sd,
+                              struct v4l2_subdev_state *state, u32 pad,
+                              u64 streams_mask);
+        int (*disable_streams)(struct v4l2_subdev *sd,
+                               struct v4l2_subdev_state *state, u32 pad,
+                               u64 streams_mask);
 [...]
 };
```

- New subdev pad operation to enable and disable streams.
- Replaces .s_stream(), helpers available to enable interop between .s_stream() and new operations in both directions.

Streams – Kernel Helpers

Streams Helpers (1/6)

```
+/**
+ * v4l2_subdev_has_pad_interdep - MC has_pad_interdep implementation for subdevs
+ *
+ * @entity: pointer to &struct media_entity
+ * @pad0: pad number for the first pad
+ * @pad1: pad number for the second pad
+ *
+ * This function is an implementation of the media_entity_operations.has_pad_interdep
+ * operation for subdevs that implement the multiplexed streams API (as
+ * indicated by the V4L2_SUBDEV_FL_STREAMS subdev flag).
+ *
+ * It considers two pads interdependent if there is an active route between pad0
+ * and pad1.
+ */
+bool v4l2_subdev_has_pad_interdep(struct media_entity *entity,
+                                  unsigned int pad0, unsigned int pad1);
```

- Helper to implement .has_pad_interdep() based on active routing table.

Streams Helpers (2/6)

```
+/**
+ * v4l2_subdev_state_get_stream_format() - Get pointer to a stream format
+ * @state: subdevice state
+ * @pad: pad id
+ * @stream: stream id
+ *
+ * This returns a pointer to &struct v4l2_mbus_framefmt for the given pad +
+ * stream in the subdev state.
+ *
+ * If the state does not contain the given pad + stream, NULL is returned.
+ */+struct v4l2_mbus_framefmt *
+v4l2_subdev_state_get_stream_format(struct v4l2_subdev_state *state,
+                                    unsigned int pad, u32 stream);

+v4l2_subdev_state_get_stream_crop(), v4l2_subdev_state_get_stream_compose()
```

- Helpers to retrieve stream format, crop and selection rectangle pointers from the subdev state.

Streams Helpers (3/6)

```
+/**
+ * v4l2_subdev_state_get_opposite_stream_format() - Get pointer to opposite
+ *                                                  stream format
+ * @state: subdevice state
+ * @pad: pad id
+ * @stream: stream id
+ *
+ * This returns a pointer to &struct v4l2_mbus_framefmt for the pad + stream
+ * that is opposite the given pad + stream in the subdev state.
+ *
+ * If the state does not contain the given pad + stream, NULL is returned.
+ */
+struct v4l2_mbus_framefmt *
+v4l2_subdev_state_get_opposite_stream_format(struct v4l2_subdev_state *state,
+                                             u32 pad, u32 stream);
```

- Helper to retrieve stream format on the other end of a stream within a subdev.

Streams Helpers (4/6)

```
+/**
+ * v4l2_subdev_state_xlate_streams() - Translate streams from one pad to another
+ *
+ * @state: Subdevice state
+ * @pad0: The first pad
+ * @pad1: The second pad
+ * @streams: Streams bitmask on the first pad
+ *
+ * Streams on sink pads of a subdev are routed to source pads as expressed in
+ * the subdev state routing table. Stream numbers don't necessarily match on
+ * the sink and source side of a route. This function translates stream numbers
+ * on @pad0, expressed as a bitmask in @streams, to the corresponding streams
+ * on @pad1 using the routing table from the @state. It returns the stream mask
+ * on @pad1, and updates @streams with the streams that have been found in the
+ * routing table.
+ *
+ * @pad0 and @pad1 must be a sink and a source, in any order.
+ *
+ * Return: The bitmask of streams of @pad1 that are routed to @streams on @pad0.
+ */
+u64 v4l2_subdev_state_xlate_streams(const struct v4l2_subdev_state *state,
+                                    u32 pad0, u32 pad1, u64 *streams);
```

- Helper to follow streams within a subdev.

```
+/**
+ * v4l2_subdev_s_stream_helper() - Helper to implement the subdev s_stream
+ *         operation using enable_streams and disable_streams
+ * @sd: The subdevice
+ * @enable: Enable or disable streaming
+ *
+ * Subdevice drivers that implement the streams-aware
+ * &v4l2_subdev_pad_ops.enable_streams and &v4l2_subdev_pad_ops.disable_streams
+ * operations can use this helper to implement the legacy
+ * &v4l2_subdev_video_ops.s_stream operation.
+ *
+ * This helper can only be used by subdevs that have a single source pad.
+ *
+ * Return: 0 on success, or a negative error code otherwise.
+ */
+int v4l2_subdev_s_stream_helper(struct v4l2_subdev *sd, int enable);
```

- Helper to implement legacy .s_stream() operation based on the new .enable_stream() and .disable_stream().
- This allows usage of subdevs that use the new API with drivers that call the legacy .s_stream() operation.

Streams Helpers (5/6)

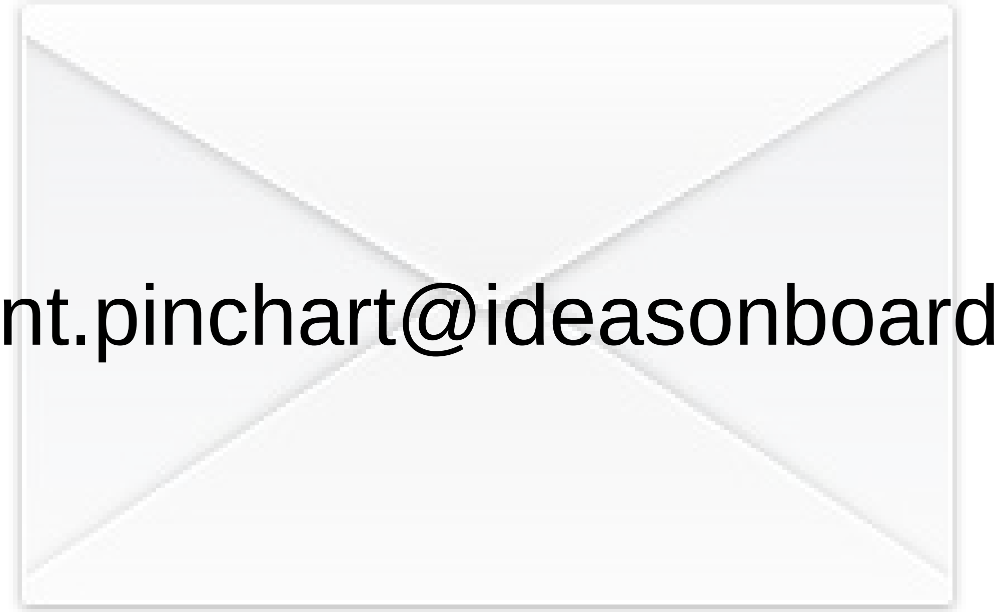
Streams Helpers (6/6)

```
+/**
+ * v4l2_subdev_enable_streams() - Enable streams on a pad
+ * @sd: The subdevice
+ * @pad: The pad
+ * @streams_mask: Bitmask of streams to enable
+ *
+ * This function enables streams on a source @pad of a subdevice. The pad is
+ * identified by its index, while the streams are identified by the
+ * @streams_mask bitmask. This allows enabling multiple streams on a pad at
+ * once.
+ *
+ * Enabling a stream that is already enabled isn't allowed. If @streams_mask
+ * contains an already enabled stream, this function returns -EALREADY without
+ * performing any operation.
+ *
+ * Per-stream enable is only available for subdevs that implement the
+ * .enable_streams() and .disable_streams() operations. For other subdevs, this
+ * function implements a best-effort compatibility by calling the .s_stream()
+ * operation, limited to subdevs that have a single source pad.
+ *
+ * Return:
+ * * 0: Success
+ * * -EALREADY: One of the streams in streams_mask is already enabled
+ * * -EINVAL: The pad index is invalid, or doesn't correspond to a source pad
+ * * -EOPNOTSUPP: Falling back to the legacy .s_stream() operation is
+ *   impossible because the subdev has multiple source pads
+ */
+int v4l2_subdev_enable_streams(struct v4l2_subdev *sd, u32 pad,
+                               u64 streams_mask);

+ v4l2_subdev_disable_streams()
```

- Helpers that wrap .enable_stream() and disable_stream(), falling back to legacy .s_stream().
- This is meant to replace direct calls to subdev operations when enabling or disabling streams, to allow interoperability between old and new subdev drivers.

Contact



laurent.pinchart@ideasonboard.com

? !

Go raibh maith agat