THE LINUX FOUNDATION

Embedded Linux Conference

SAMSUNG

V4L2 Controls
From Perspective of Video Capture Devices

Sathyakam M, Inbaraj E

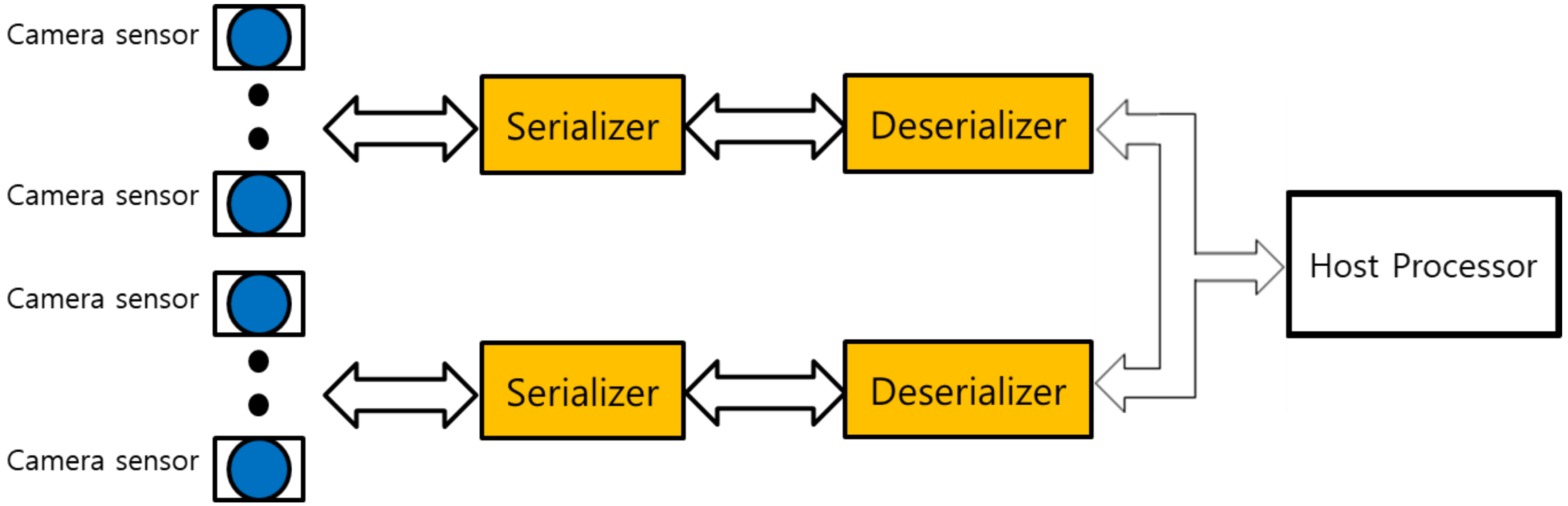
SSIR Samsung Semiconductor India R&D Center

Agenda

- Structure of V4L2 framework
  - V4L2 device
  - V4L2 sub-device
  - Video device nodes
  - V4L2 control objects
- Controlling V4L2 devices
  - Why are controls needed?
  - Control methods provided by V4L2
  - Leveraging controls from User space application
  - Notes on implementing controls
- Sample capture device with controls

Structure of V4L2 framework

- Typical video capture involves multiple devices, which can include a bridge device and one or more sub-devices
  - e.g. Streaming with remote cameras can have multiple sensors, Serializer and Deserializers combination to aggregate camera streams and camera interface controllers



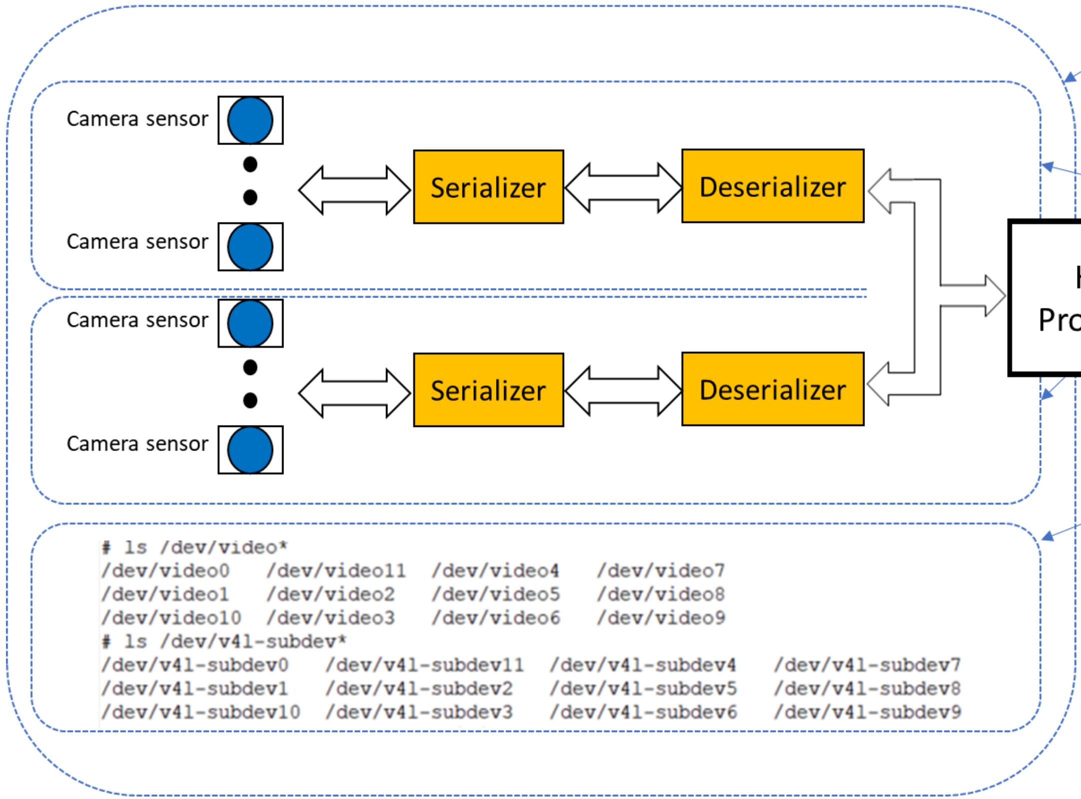
Structure of V4L2 framework (Cont'd.)

V4L2 framework provides kernel data structures for drivers to describe the bridge device and attached sub-devices

- ***struct v4l2_device***: Is the top level data structure acting as root node of streaming device and is responsible for managing the child devices
- ***struct v4l2_subdev***: The sub-devices attached to the bridge device are represented with *struct v4l2_subdev*. These can be camera interface controller in host processor or the camera sensors providing the image stream.
- ***struct video_device***: Abstracts the capture interface and exposes the device nodes under "/dev/" directory
- ***struct v4l2_ctrl***: Describes control properties and tracks control value
- ***struct v4l2_ctrl_handler***: Keeps track of all controls within the device

V4L2 framework - Structure of Driver

❑ Block Diagram and Associated V4L2 objects



```
# ls /dev/video*
/dev/video0    /dev/video11   /dev/video4    /dev/video7
/dev/video1    /dev/video2    /dev/video5    /dev/video8
/dev/video10   /dev/video3    /dev/video6    /dev/video9
# ls /dev/v4l-subdev*
/dev/v4l-subdev0    /dev/v4l-subdev11   /dev/v4l-subdev4    /dev/v4l-subdev7
/dev/v4l-subdev1    /dev/v4l-subdev2    /dev/v4l-subdev5    /dev/v4l-subdev8
/dev/v4l-subdev10   /dev/v4l-subdev3    /dev/v4l-subdev6    /dev/v4l-subdev9
```

```
struct v4l2_device {
        struct device *dev;
        struct media_device *mdev;
        struct list_head subdevs;
        spinlock_t lock;
        char name[V4L2_DEVICE_NAME_SIZE];
        struct v4l2_ctrl_handler *ctrl_handler;
        ...
};
struct v4l2_subdev {
        bool owner_v4l2_dev;
        struct v4l2_device *v4l2_dev;
        struct v4l2_ctrl_handler *ctrl_handler;
        char name[V4L2_SUBDEV_NAME_SIZE];
        struct video_device *devnode;
        struct device *dev;
        ...
};
struct video_device
{
        const struct v4l2_file_operations *fops;
        u32 device_caps;
        struct device dev;
        struct cdev *cdev;
        struct v4l2_device *v4l2_dev;
        struct v4l2_ctrl_handler *ctrl_handler;
        char name[32];
        ...
};
```

Controlling V4L2 devices

- Controls are needed
  - Due to diverse nature of capture devices, most of the devices expose controls that are configurable by user
  - Controls provided can be specific to the device which are vendor specific
  - Device control needs can be application specific as well
- V4L2 framework provides methods to set the controls by user
  - Standard controls
  - Extended controls
  - Custom controls
  - Private controls
- Capturing and streaming the images can implement all or some of the required controls

Controlling V4L2 devices (Cont'd.)

- Controls are accessed using control ID value
- V4L2 framework arranges the controls into classes which serve as base for control IDs

include/uapi/linux/v4l2-controls.h

```
#define V4L2_CTRL_CLASS_USER          0x00980000    /* Old-style 'user' controls */
#define V4L2_CTRL_CLASS_MPEG          0x00990000    /* MPEG-compression controls */
#define V4L2_CTRL_CLASS_CAMERA        0x009a0000    /* Camera class controls */
#define V4L2_CTRL_CLASS_FM_TX         0x009b0000    /* FM Modulator controls */
#define V4L2_CTRL_CLASS_FLASH         0x009c0000    /* Camera flash controls */
#define V4L2_CTRL_CLASS_JPEG          0x009d0000    /* JPEG-compression controls */
#define V4L2_CTRL_CLASS_IMAGE_SOURCE  0x009e0000    /* Image source controls */
#define V4L2_CTRL_CLASS_IMAGE_PROC    0x009f0000    /* Image processing controls */
#define V4L2_CTRL_CLASS_DV            0x00a00000    /* Digital Video controls */
#define V4L2_CTRL_CLASS_FM_RX         0x00a10000    /* FM Receiver controls */
#define V4L2_CTRL_CLASS_RF_TUNER      0x00a20000    /* RF tuner controls */
#define V4L2_CTRL_CLASS_DETECT        0x00a30000    /* Detection controls */
```

Controlling V4L2 devices – Standard controls

- V4L2 framework provides standard controls with predefined control IDs

```
/* User-class control IDs */

#define V4L2_CID_BASE                                                     (V4L2_CTRL_CLASS_USER | 0x900)
#define V4L2_CID_USER_BASE                              V4L2_CID_BASE
#define V4L2_CID_USER_CLASS                             (V4L2_CTRL_CLASS_USER | 1)
#define V4L2_CID_BRIGHTNESS                             (V4L2_CID_BASE+0)
#define V4L2_CID_CONTRAST                               (V4L2_CID_BASE+1)
#define V4L2_CID_SATURATION                             (V4L2_CID_BASE+2)
#define V4L2_CID_HUE                                    (V4L2_CID_BASE+3)
#define V4L2_CID_AUDIO_VOLUME                           (V4L2_CID_BASE+5)
#define V4L2_CID_AUDIO_BALANCE                          (V4L2_CID_BASE+6)
#define V4L2_CID_AUDIO_BASS                             (V4L2_CID_BASE+7)
#define V4L2_CID_AUDIO_TREBLE                           (V4L2_CID_BASE+8)
#define V4L2_CID_AUDIO_MUTE                             (V4L2_CID_BASE+9)
#define V4L2_CID_AUDIO_LOUDNESS                         (V4L2_CID_BASE+10)
#define V4L2_CID_BLACK_LEVEL                            (V4L2_CID_BASE+11) /* Deprecated */
#define V4L2_CID_AUTO_WHITE_BALANCE      (V4L2_CID_BASE+12)
#define V4L2_CID_DO_WHITE_BALANCE        (V4L2_CID_BASE+13)
#define V4L2_CID_RED_BALANCE                            (V4L2_CID_BASE+14)
#define V4L2_CID_BLUE_BALANCE                           (V4L2_CID_BASE+15)
#define V4L2_CID_GAMMA                                                    (V4L2_CID_BASE+16)
#define V4L2_CID_WHITENESS                              (V4L2_CID_GAMMA) /* Deprecated */
#define V4L2_CID_EXPOSURE                               (V4L2_CID_BASE+17)
#define V4L2_CID_AUTOGAIN                               (V4L2_CID_BASE+18)
#define V4L2_CID_GAIN                                                     (V4L2_CID_BASE+19)
#define V4L2_CID_HFLIP                                                    (V4L2_CID_BASE+20)
#define V4L2_CID_VFLIP                                                    (V4L2_CID_BASE+21)
...
#define V4L2_CID_COLORFX_CBCR                                             (V4L2_CID_BASE+42)

/* last CID + 1 */
#define V4L2_CID_LASTP1                (V4L2_CID_BASE+43)
```

Controlling V4L2 devices – Extended controls

- Extended controls for Camera Class

include/uapi/linux/v4l2-controls.h

```
#define V4L2_CID_EXPOSURE_AUTO            (V4L2_CID_CAMERA_CLASS_BASE+1)
enum  v4l2_exposure_auto_type {
      V4L2_EXPOSURE_AUTO = 0,
      V4L2_EXPOSURE_MANUAL = 1,
      V4L2_EXPOSURE_SHUTTER_PRIORITY = 2,
      V4L2_EXPOSURE_APERTURE_PRIORITY = 3
};
#define V4L2_CID_EXPOSURE_ABSOLUTE          (V4L2_CID_CAMERA_CLASS_BASE+2)
#define V4L2_CID_EXPOSURE_AUTO_PRIORITY      (V4L2_CID_CAMERA_CLASS_BASE+3)

#define V4L2_CID_PAN_RELATIVE             (V4L2_CID_CAMERA_CLASS_BASE+4)
#define V4L2_CID_TILT_RELATIVE            (V4L2_CID_CAMERA_CLASS_BASE+5)
#define V4L2_CID_PAN_RESET               (V4L2_CID_CAMERA_CLASS_BASE+6)
#define V4L2_CID_TILT_RESET              (V4L2_CID_CAMERA_CLASS_BASE+7)

#define V4L2_CID_PAN_ABSOLUTE             (V4L2_CID_CAMERA_CLASS_BASE+8)
#define V4L2_CID_TILT_ABSOLUTE            (V4L2_CID_CAMERA_CLASS_BASE+9)

#define V4L2_CID_FOCUS_ABSOLUTE            (V4L2_CID_CAMERA_CLASS_BASE+10)
#define V4L2_CID_FOCUS_RELATIVE           (V4L2_CID_CAMERA_CLASS_BASE+11)
#define V4L2_CID_FOCUS_AUTO             (V4L2_CID_CAMERA_CLASS_BASE+12)
...

#define V4L2_CID_PAN_SPEED              (V4L2_CID_CAMERA_CLASS_BASE+32)
#define V4L2_CID_TILT_SPEED             (V4L2_CID_CAMERA_CLASS_BASE+33)
```

Controlling V4L2 devices – Custom controls

- Often devices provide controls which are specific to that device and not available in standard or extended controls
  - Sensor specific parameters
  - Test pattern generation
  - Set number of data lanes to stream (e.g. for MIPI-CSI2 interface)
  - Set per stream controls
    - Error count threshold
    - DMA controls
  - Any other controls which are driver specific (Implemented in driver using *V4L2_CID_PRIVATE_BASE* or higher values)

Controlling V4L2 devices – Inheriting controls

- Controls from one handler can be added to another

```
include/media/v4l2-ctrls.h

int v4l2_ctrl_add_handler(struct v4l2_ctrl_handler *hdl,
                          struct v4l2_ctrl_handler *add,
                          v4l2_ctrl_filter filter,
                          bool from_other_dev);
```

  - *filter*: function to select the controls to be added
    - controls can also be filtered based on class to which they belong using *V4L2_CTRL_ID2WHICH()* helper
  - *from_other_dev:* controls are defined in another device

- Useful when controls implemented by two devices are same, thus avoids reimplementation
  - e.g. controls like gain, brightness, exposure which are implemented by sensor can be reused in bridge device node

Controlling V4L2 devices – Private controls

- Controls may not be always needed by bridge device node
  - e.g. advanced debug feature particular to sensor is not needed at *video_device* node
- *struct v4l2_ctrl* provides bit mapped variable *is_private* to inform the framework exclude the control to be added to another handler
  - Setting *is_private* prevents the control being added from sub-device to root device during *v4l2_device_register_subdev()* call

Controlling V4L2 devices – Control flags

- Following control flags are available for V4L2 devices

include/uapi/linux/videodev2.h

```
/*  Control flags  */
#define V4L2_CTRL_FLAG_DISABLED         0x0001
#define V4L2_CTRL_FLAG_GRABBED          0x0002
#define V4L2_CTRL_FLAG_READ_ONLY        0x0004
#define V4L2_CTRL_FLAG_UPDATE           0x0008
#define V4L2_CTRL_FLAG_INACTIVE         0x0010
#define V4L2_CTRL_FLAG_SLIDER           0x0020
#define V4L2_CTRL_FLAG_WRITE_ONLY       0x0040
#define V4L2_CTRL_FLAG_VOLATILE         0x0080
#define V4L2_CTRL_FLAG_HAS_PAYLOAD      0x0100
#define V4L2_CTRL_FLAG_EXECUTE_ON_WRITE 0x0200
#define V4L2_CTRL_FLAG_MODIFY_LAYOUT    0x0400
```

Controlling V4L2 devices – V4L2 Control Notification

- Used to notify the change in control's value.
- Notification will be helpful in case of inherited controls.
  - e.g. sometimes the platform or bridge drivers need to be notified when control from sub-device driver changes.
- Only one notify function should be used per control handler.
- You can set a notify callback by calling below function.

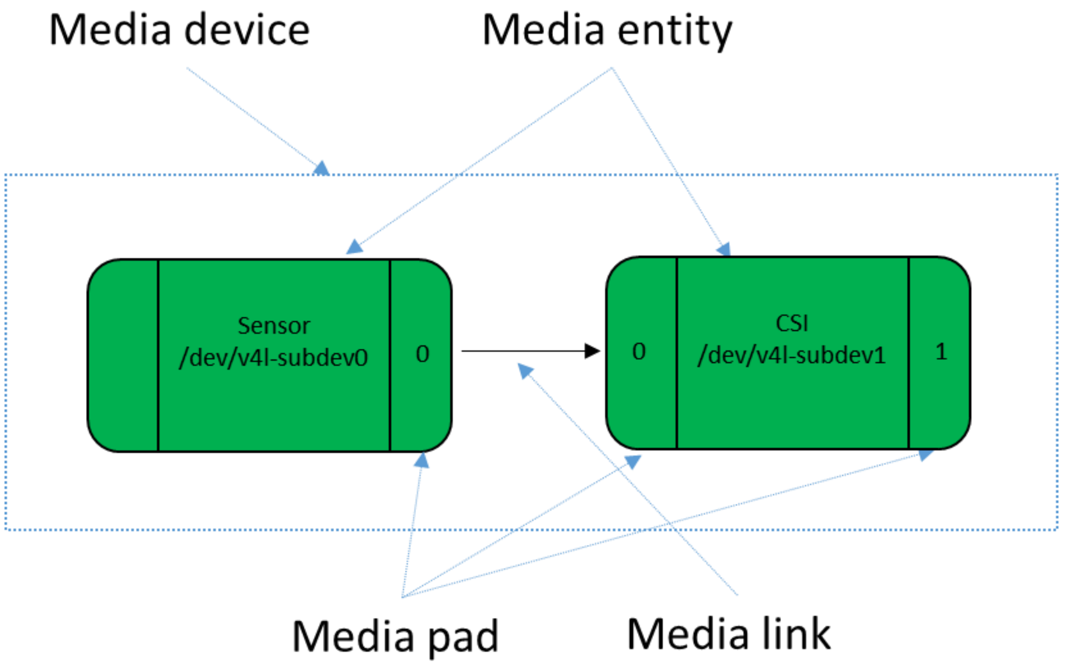
```
void v4l2_ctrl_notify(struct v4l2_ctrl *ctrl, v4l2_ctrl_notify_fnc notify,
                      void *priv);
```

Integration with Linux media controller framework

- Media device consists of media pipeline involving sub-devices (media entities) with links between pads (source, sink)
  - *struct media_device*
- Media Entity
  - Describes basic hardware block
  - Off the chip devices such as sensors or, On chip IPs such as ISP or any logical device participating in strea ming pipeline such as the DMA engine
  - *struct media_entity*
- Media Pad
  - Connection endpoint through which entities transfer data
  - Source and Sink pads
  - *struct media_pad*
- Media Link
  - Connection between two pads, either on same entity or between different entities
  - *struct media_link*

Integration with Linux media controller framework (Cont'd.)

- Diagrammatic representation



Integration with Linux media controller framework (Cont'd.)

- V4L2 device abstraction
  - Uses *struct media_device* to abstract *struct v4l2_device* objects

```
include/media/v4l2-device.h

struct v4l2_device {
        struct device *dev;
        struct media_device *mdev;
        ...
};
```

- V4L2 sub-devices and video devices are observed as media entities

```
include/media/v4l2-subdev.h

struct v4l2_subdev {
#if defined(CONFIG_MEDIA_CONTROLLER)
        struct media_entity entity;
#endif
        ...
};
```

```
include/media/v4l2-dev.h

struct video_device
{
#if defined(CONFIG_MEDIA_CONTROLLER)
        struct media_entity entity;
        struct media_intf_devnode *intf_devnode;
        struct media_pipeline pipe;
#endif
        ...
};
```

  - *type* field of *struct media_entity* is set to *MEDIA_ENTITY_TYPE_V4L2_SUBDEV* or *MEDIA_ENTITY_TYPE_VIDEO_DEVICE*

Integration with Linux media controller framework (Cont'd.)

- V4L2 driver initializes the media device within struct v4l2_device using *media_device_init()*

```
struct v4l2_device {
        struct device *dev;
        struct media_device *mdev;
        ...
};
```

- Each entity driver initializes its entities and pad arrays
    - *v4l2_subdev->entity*
    - *video_device->entity*
    - *media_entity_pads_init()*
- Controls set by V4L2 driver are applicable for each media entity

Leveraging controls from User space application

- V4L2 framework provides IOCTLs to
  - Enumerate the controls provided by driver
    - *VIDIOC_QUERYCTRL*
    - *VIDIOC_QUERY_EXT_CTRL*
  - Get control value
    - *VIDIOC_G_CTRL*
    - *VIDIOC_G_EXT_CTRLS*
  - Set control value
    - *VIDIOC_S_CTRL*
    - *VIDIOC_S_EXT_CTRLS*
    - *VIDIOC_TRY_EXT_CTRLS*
- Drivers must implement these IOCTLs when device has one or more controls
- Custom control IDs must be exposed to applications from header files under *"include/uapi/linux/"*

Leveraging controls from User space application (Cont'd.)

- Enumerate all the User controls

```
for (queryctrl.id = V4L2_CID_BASE;
    queryctrl.id < V4L2_CID_LASTP1;
    queryctrl.id++) {
   if (0 == ioctl(fd, VIDIOC_QUERYCTRL, &queryctrl)) {
      if (queryctrl.flags & V4L2_CTRL_FLAG_DISABLED)
         continue;
…
```

- Enumerate the controls provided by driver

```
for (queryctrl.id = V4L2_CID_PRIVATE_BASE;;
    queryctrl.id++) {
   if (0 == ioctl(fd, VIDIOC_QUERYCTRL, &queryctrl)) {
      if (queryctrl.flags & V4L2_CTRL_FLAG_DISABLED)
         continue;

      printf("Control %s\n", queryctrl.name);
      …
```

Leveraging controls from User space application - Changing controls

- Query the desired user control
- Use *VIDIOC_G_CTRL* and *VIDIOC_S_CTRL* to get or set the control value

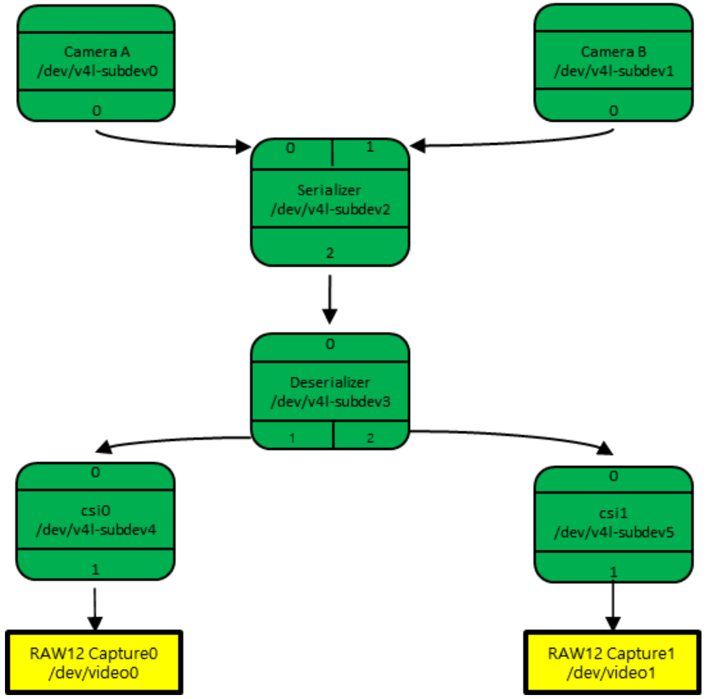
```
struct v4l2_queryctrl queryctrl;
struct v4l2_control control;

memset(&queryctrl, 0, sizeof(queryctrl));
queryctrl.id = V4L2_CID_CUSTOM_CONTROL_1;

if (-1 == ioctl(sd_fd, VIDIOC_QUERYCTRL, &queryctrl)) {
            perror(" V4L2_CID_CUSTOM_CONTROL_1 not supported!\n");
            exit(EXIT_FAILURE);
}

memset(&control, 0, sizeof(control));
control.id = V4L2_CID_CUSTOM_CONTROL_1;

if (0 == ioctl(fd, VIDIOC_G_CTRL, &control)) {
   /* set the desired value */
   control.value = x;

if (-1 == ioctl(fd, VIDIOC_S_CTRL, &control)
      && errno != ERANGE) {
      perror("VIDIOC_S_CTRL");
      exit(EXIT_FAILURE);
   }
} else if (errno != EINVAL) {
   perror("VIDIOC_G_CTRL");
   exit(EXIT_FAILURE);
}
```

Notes on Implementing Controls

- Driver implementation of controls need to be documented
  - *"Documentation/userspace-api/media/drivers/"*
  - Useful for custom controls
- *v4l2_subdev* controls can be overwritten by *v4l2_dev* during sub-device registration
  - Decide which controls need protection
  - Set *is_private* flag for sub-device controls which need protection
  - Adding controls to *v4l2_subdev* after the device is registered will not have any effect
    - Add the required controls for the sub-device prior to *v4l2_device_register_subdev()*
- Set the controls to default value
  - Call *v4l2_ctrl_handler_setup()* for the control handler and initialize any hardware control values
  - Helps avoiding setting the hardware device to default values from user application

Sample capture device with controls

- 2 Camera sensors connected to Serializer
- Image streams aggregated by Serializer, Deserializer and received by Host at MIPI CSI2 Rx port
- Streaming RAW12 Bayer images



How to add controls?

- Driver needs to fill *struct v4l2_ctrl_config* and *struct v4l2_ctrl_ops*
- After initializing above structures, now controls are ready to get initialized

```
struct v4l2_ctrl *v4l2_ctrl_new_custom(struct v4l2_ctrl_handler *hdl,
                                       const struct v4l2_ctrl_config *cfg,
                                       void *priv);
struct v4l2_ctrl *v4l2_ctrl_new_std(struct v4l2_ctrl_handler *hdl,
                                    const struct v4l2_ctrl_ops *ops,
                                    u32 id, s64 min, s64 max, u64 step,
                                    s64 def);

struct v4l2_ctrl *v4l2_ctrl_new_std_menu(struct v4l2_ctrl_handler *hdl,
                                         const struct v4l2_ctrl_ops *ops,
                                         u32 id, u8 max, u64 mask, u8 def);
```

- Driver should use these functions to register for custom or standard controls
  - Depending on the type of the control registration function will change

Structure v4l2_ctrl_config

- Exploring *v4l2_ctrl_config* structure

include/media/v4l2-ctrls.h

```
struct v4l2_ctrl_config {
        const struct v4l2_ctrl_ops *ops;
        u32 id;
        const char *name;
        enum v4l2_ctrl_type type;
        s64 min;
        s64 max;
        u64 step;
        s64 def;
        unsigned int is_private:1;
        ...
};
```

ops – Callback used to handle the controls.
id – Control id.
name – Driver will set the name.
min – Minimum value the control allows.
max – Maximum value the control allows.
def – *v4l2_ctrl_handler_setup* function will set control to default value.
is_private – set this to prevent control used by other control handler.

Additional fields which can be used depending on control type implementation.

Structure v4l2_ctrl_ops

include/media/v4l2-ctrls.h

```
struct v4l2_ctrl_ops {
	int (*g_volatile_ctrl)(struct v4l2_ctrl *ctrl);
	int (*try_ctrl)(struct v4l2_ctrl *ctrl);
	int (*s_ctrl)(struct v4l2_ctrl *ctrl);
};.
```

- ***g_volatile_ctrl***: Get a new value for this control. Generally only relevant for volatile (and usually read-only) controls such as a control that return the current signal strength which changes continuously. If not set, then the currently cached value will be returned
- ***try_ctrl***: Test whether control's value is valid. Only relevant when the usual min/max/step checks are not sufficient
- ***s_ctrl***: Actually set the new control value .*s_ctrl* is compulsory. The *ctrl->handler-> lock* is held when these ops are called, so no one else can access controls owned by that handler

v4l2_ctrl_ops – An Example

- Driver should initialize *v4l2_ctrl_ops* and define *s_ctrl* callback to implement different control functionality

```
sample_capture_ip.c

int capture_ip_s_ctrl(struct v4l2_ctrl  *ctrl)
{
        switch (ctrl->id) {
        case V4L2_CID_USER_NO_OF_LANE:
                /*handles HW or SW related functionality */
                ...
                break;
        default:
                break;
        }
        return 0;
}

static const struct v4l2_ctrl_ops capture_ctrl_ops = {
        .s_ctrl = capture_ip_s_ctrl,
};
```

Structure and example for adding controls

- Example for setting number of lanes used to receive the frames
- min and max are minimum and maximum number of lanes used to receive the frames
- step is used to set intermediate value between min and max no of lanes

```
sample_capture_ip.c

static const struct v4l2_ctrl_config capture_ip_set_nb_lane = {
        .ops    = &capture_ctrl_ops,
        .id     = V4L2_CID_USER_NO_OF_LANE,
        .type   = V4L2_CTRL_TYPE_INTEGER,
        .name   = "no of lanes",
        .min    = 1,
        .step   = 1,
        .max    = 4,
        .def    = 1,
};

static int capture_probe(struct platform_device *pdev)
{
        ...
        v4l2_ctrl_new_custom(&dev->ctrl_handler, &capture_ip_set_nb_lane, NULL);
        ...
}
```

Steps to implement control handler in driver

- ❑ Add the *v4l2_ctrl_handler* structure to the top level structure or driver's private structure
- ❑ For V4L2 driver add at the same level where V4L2 device is present
- ❑ Initialize the control handler
- ❑ Add all the necessary controls to your device as discussed in previous slides
- ❑ Optionally force initialization of all the controls
- ❑ Free the control handler when device is leaving or removed

Adding Control Handler – An Example

- Implementation of steps discussed in previous slide

```
sample_capture_ip.c

struct capture_dev {
        ...
        struct v4l2_device v4l2_dev;
        ...
        struct v4l2_ctrl_handler ctrl_handler;
        ...
};

struct capture_dev *cap_dev

static int capture_probe(struct platform_device *pdev)
{
        ...
       v4l2_ctrl_handler_init(&cap_dev->ctrl_handler, nr_of_controls);
        cap_dev->v4l2_dev.ctrl_handler = &cap_dev->ctrl_handler;
        v4l2_ctrl_new_custom(&cap_dev->ctrl_handler, &capture_ip_set_nb_lane, NULL);
        /* Add control like above as discussed in how to add control section */
       ...
        /* this step complety optional */
        v4l2_ctrl_handler_setup(&dev->ctrl_handler);
        ...
}
```

How to Implement Control Notification?

```
static void sensor_ctrl_notify(struct v4l2_ctrl *ctrl, void *priv)
{

    switch(ctrl->id)
    {
    case V4L2_CID_USER_NO_OF_LANE:
        /* Sync the data structures */
        break;
    }
}

static int sensor_probe(struct platform_device *pdev)
{


    ...
    /* After adding control to the control handler */
    v4l2_ctrl_notify(v4l2_ctrl_find(hdl, V4L2_CID_USER_NO_OF_LANE,
        sensor_ctrl_notify, priv);

}
```

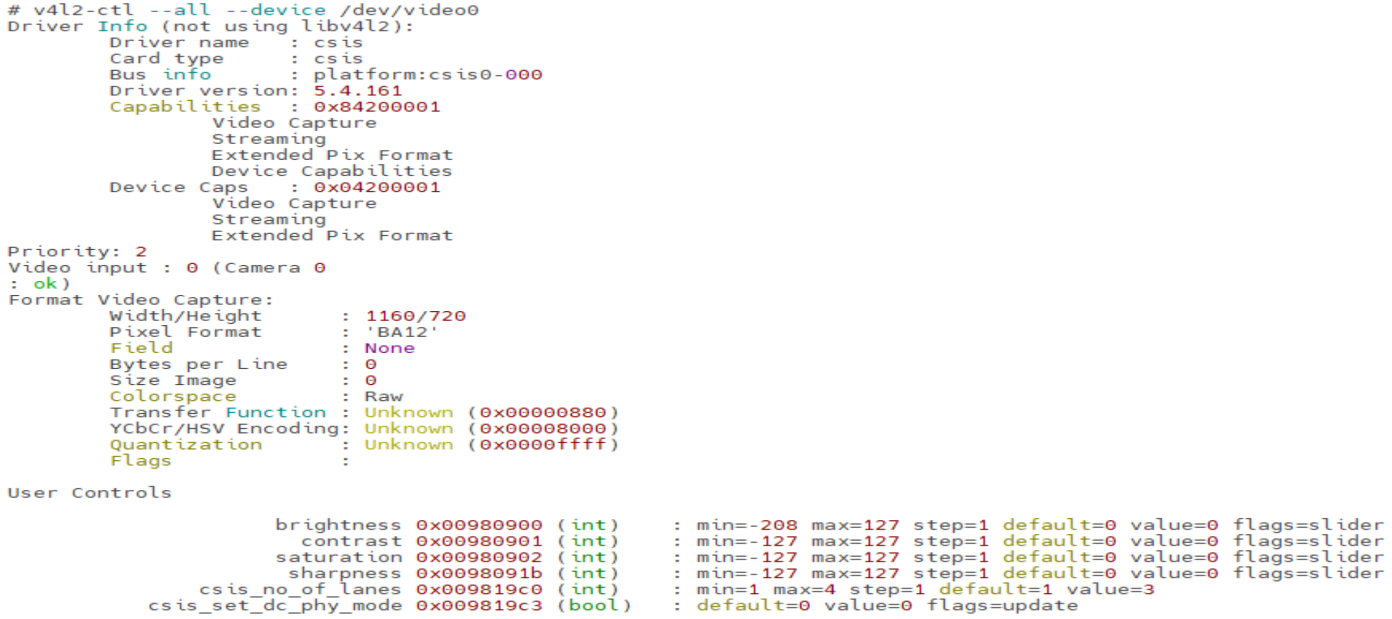
V4L2-ctl utils

- v4l2-ctl --list-device
  - List all the v4l2-device with video device number and corresponding name

```
# v4l2-ctl --list-device

csis (platform:csis0-000):
        /dev/video0

csis (platform:csis0-001):
        /dev/video1
```

V4L2-ctl utils (Cont'd.)

- v4l2-ctl --all --device /dev/video{n}
  - Gives information about driver name, card type, bus info, driver version and device capabilities (video capture and streaming)
  - Default width, height and pixel format



```
# v4l2-ctl --all --device /dev/video0
Driver Info (not using libv4l2):
        Driver name   : csis
        Card type     : csis
        Bus info      : platform:csis0-000
        Driver version: 5.4.161
        Capabilities  : 0x84200001
                Video Capture
                Streaming
                Extended Pix Format
                Device Capabilities
        Device Caps   : 0x04200001
                Video Capture
                Streaming
                Extended Pix Format
Priority: 2
Video input : 0 (Camera 0
: ok)
Format Video Capture:
        Width/Height       : 1160/720
        Pixel Format       : 'BA12'
        Field              : None
        Bytes per Line     : 0
        Size Image         : 0
        Colorspace         : Raw
        Transfer Function  : Unknown (0x00000880)
        YCbCr/HSV Encoding: Unknown (0x00008000)
        Quantization       : Unknown (0x0000ffff)
        Flags              :

User Controls

                     brightness 0x00980900 (int)    : min=-208 max=127 step=1 default=0 value=0 flags=slider
                       contrast 0x00980901 (int)    : min=-127 max=127 step=1 default=0 value=0 flags=slider
                     saturation 0x00980902 (int)    : min=-127 max=127 step=1 default=0 value=0 flags=slider
                      sharpness 0x0098091b (int)    : min=-127 max=127 step=1 default=0 value=0 flags=slider
                csis_no_of_lanes 0x009819c0 (int)    : min=1 max=4 step=1 default=1 value=3
            csis_set_dc_phy_mode 0x009819c3 (bool)   : default=0 value=0 flags=update
```

V4L2-ctl utils (Cont'd.)

- v4l2-ctl --list-ctrls --device /dev/video{n}
  - List all the control owned by the device.
  - Gives information about control name, control id, type of control, min, max, default, current value and flag.

```
# v4l2-ctl --list-ctrls --device /dev/video0

User Controls

                     brightness 0x00980900 (int)    : min=-208 max=127 step=1 default=0 value=0 flags=slider
                       contrast 0x00980901 (int)    : min=-127 max=127 step=1 default=0 value=0 flags=slider
                     saturation 0x00980902 (int)    : min=-127 max=127 step=1 default=0 value=0 flags=slider
                      sharpness 0x0098091b (int)    : min=-127 max=127 step=1 default=0 value=0 flags=slider
                csis_no_of_lanes 0x009819c0 (int)    : min=1 max=4 step=1 default=1 value=3
            csis_set_dc_phy_mode 0x009819c3 (bool)   : default=0 value=0 flags=update
```

V4L2-ctl utils (Cont'd.)

- v4l2-ctl --device /dev/video{n} --get-ctrl={control_name}
  - Used to get current value of the control for given device.

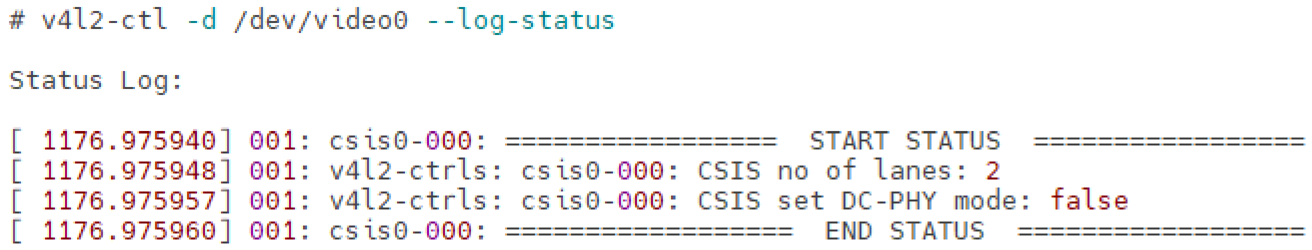
```
# v4l2-ctl --device /dev/video0 --get-ctrl=csis_no_of_lanes
csis_no_of_lanes: 1
```

- v4l2-ctl --device /dev/video{n} --set-ctrl={control_name} ={value}
  - Used to set the control to new value for given device.

```
# v4l2-ctl --device /dev/video0 --set-ctrl=csis_no_of_lanes=2
# v4l2-ctl --device /dev/video0 --get-ctrl=csis_no_of_lanes
csis_no_of_lanes: 2
```

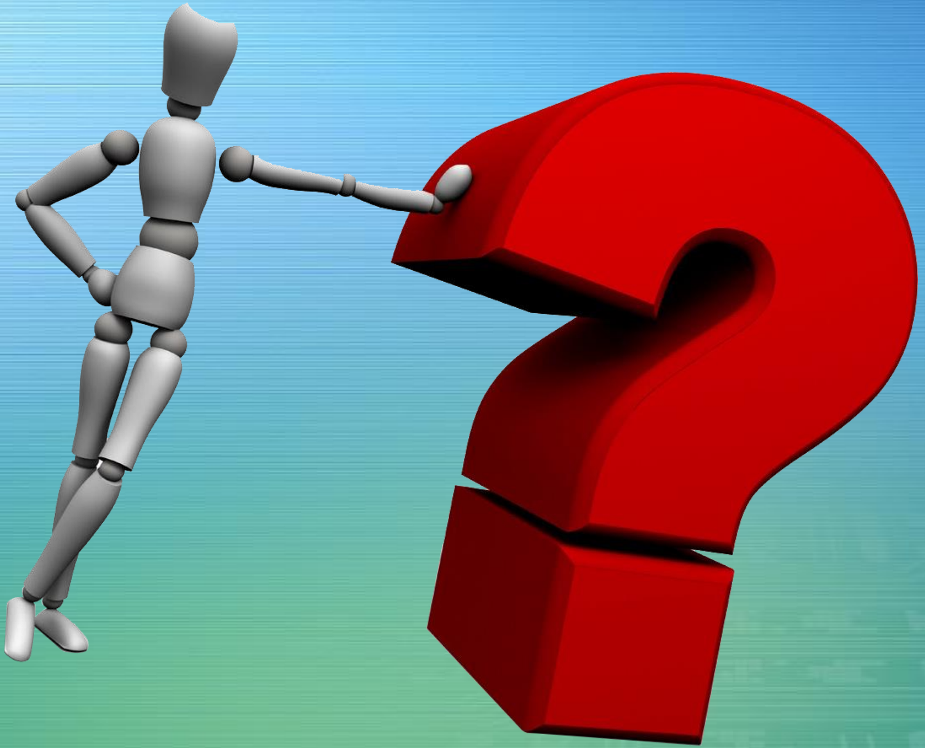
V4L2-ctl utils (Cont'd.)

- v4l2-ctl --d /dev/video{n} --log-status
  - Gives the current value of all the controls owned by the device



```
# v4l2-ctl -d /dev/video0 --log-status

Status Log:

[ 1176.975940] 001: csis0-000: =================  START STATUS  =================
[ 1176.975948] 001: v4l2-ctrls: csis0-000: CSIS no of lanes: 2
[ 1176.975957] 001: v4l2-ctrls: csis0-000: CSIS set DC-PHY mode: false
[ 1176.975960] 001: csis0-000: ==================  END STATUS  ==================
```

Any Questions ?



THANK YOU