Advanced Camera Support on Allwinner SoCs with Mainline Linux

Paul Kocialkowski
paul@bootlin.com
Paul Kocialkowski

- Embedded Linux engineer at Bootlin
  - Embedded Linux expertise
  - Development, consulting and training
  - Strong open-source focus
- Open-source contributor
  - Initial author of the cedrus VPU driver in V4L2
  - Contributor to the sun4i-drm DRM driver
  - Contributing the logicvc-drm DRM driver
  - Developed the displaying and rendering graphics with Linux training
- Living in Toulouse, south-west of France
An Introduction to Image Capture Technology
Overview of the Digital Image Capture Chain

- **Optics**: shape light rays
- **Sensor**: convert light to digital values
- **Interface**: transport values
- **Processing**: produce good-looking pictures
- **Display/encoding**: show/store pictures (*out of the scope of this talk*)
Data coming from a sensor ADC needs processing:

- Data corresponds to a **bayer pattern**, not pixels
- Brightness is linear, not adapted for display
- Sensors have a non-zero **dark-level current**
- Noise is present, color is off, image looks bad

Enhancement takes place in **Image Signal Processors (ISPs)**

Three distinct domains are involved:

1. **Bayer domain**, ends with debayering step
2. **RGB domain**, ends with YUV conversion
3. **YUV domain**, ends with final picture
Various enhancements are usually applied to the image:

- **Dead pixel correction**: discard invalid values
- **Black level correction**: remove dark level current
- **White balance**: adjust R-G-B balance with coefficients/offsets
- **Noise filtering**: remove electronic noise
- **Color matrix**: adjust colors for fidelity
- **Gamma**: adjust brightness curve for non-linearity
- **Saturation**: adjust colorfulness
- **Brightness**: adjust global luminosity
- **Contrast**: adjust bright/dark difference
Image Enhancements in ISPs

More advanced enhancements may also be applied:

- **Lens shading**: correct lens irregular brightness
- **Lens dewarp**: correct lens geometry distortion effect
- **Stabilization**: crop to remove shaking
- **Color LUT**: Translate colors with a specific style

Hardware implementations:

- ISPs embedded in sensors tend to be **simple**
  - Provide YUV data to the camera interface
- Multimedia Systems on a Chip often have an **advanced ISP**
  - Require raw bayer data on the camera interface
  - Require specific calibration data for the sensor/lens
Processing RAW Images: Illustration

- Bayer step
- RGB step
- YUV step
Parameters to Adjust

Some parameters depend on the situation:
- **Focus** depends on the area of interest
- **White balance** depends on the light source(s)
- **Exposure** depends on the amount of light

Exposure depends on a few parameters:
- Diaphragm **aperture** (f-number)
- **Exposure time** (shutter speed)
- **Amplifier gain** (ISO number equivalent)

Advanced users will set parameters manually, with artistic implications
In other cases, automatic parameters control is desirable:

- **Automatic exposition**: manage exposure time and gain (optionally diaphragm)
- **Auto-focus**: detect blurry and sharp areas, adjust with focus coil
- **Auto white balance**: detect dominant lighting and adjust

Implemented using 3A algorithms:

- General algorithms described in academic literature
- Involve a feedback loop system, using statistics
- Implementations are usually hardware specific (ISP and sensor), often considered to be the secret sauce!
Hardware Interfaces for Capture

Sensors need to transmit data:

▶ Analog interfaces (CVBS, etc) are mostly deprecated
▶ **Parallel** digital interfaces: basic, BT.656
  *typically used with old and low-end sensors*
▶ **Serial** digital interfaces: MIPI CSI-2, LVDS, SDI, HiSPi
  *typically used with high-end sensors*

**Basic parallel** interface:

▶ One TTL signal per bit, usually 8/10/12/16/24 bits width
▶ Pixel clock and sync signals (hsync, vsync)

**MIPI CSI-2** serial interface:

▶ Differential pairs, using double data rate (DDR)
▶ One clock lane (high rates) and 1-4 data lanes
Scope and Use Case
Scope and Use Case: Allwinner + MIPI CSI-2 + ISP

**Allwinner platforms** (V3 and A83T):
- Systems on a Chip with ARM CPUs
- MIPI CSI-2 receiver
- Camera interface (CSI)
- Image Signal Processor (ISP)

**Image sensors** (OV8865, OV5648):
- I2C control interface
- MIPI CSI-2 transmitter
- Bayer RAW formats (10/12 bits)
- Minimal to inexistent onboard ISP

The BananaPi-M3 with OV8865 connected
Status of Allwinner Camera Support in Mainline Linux
Mainline Linux Support and Allwinner Camera Support

Allwinner platform support in mainline Linux:

- Long-time effort from the **sunxi community**, very active
  https://linux-sunxi.org/Linux_mainlining_effort
- Multimedia areas are often the last missing parts
- Allwinner started contributing (more or less) very recently

Camera support in mainline Linux:

- `sun4i-csi` driver for first generation CSI
- `sun6i-csi` driver for second generation CSI
- Third generation CSI support is missing
- MIPI CSI-2 and ISP support was entirely missing
  *non-free blobs for ISP support and A80 MIPI CSI-2 in SDK*
Video4Linux2 (V4L2) is the subsystem/API for media support in Linux

- Supports various types of **pixel-related devices**
  
  *basically anything that is not a display or gpu*

- Provides userspace with **video devices** (e.g. /dev/video0)

- Implements a generic **userspace API** including:
  
  - Format negotiation, implemented in `struct v4l2_ioctl_ops`
  
  - Memory management (alloc, free, mmap), implemented in `struct vb2_mem_ops`
  
  - A queue interface for buffers of a given type (output, capture...), implemented in `struct vb2_ops`

  - A control interface for configuration

- Good fit for **all-in-one devices** (e.g. USB UVC cameras) 
  
  *assumes that a memory (DMA) interface is available*
Complex systems bring the need for more refinement:

- Internal blocks with FIFOs
- External devices with interfaces (e.g. sensors)
- Possibility to configure each block and the topology

Hence the notion of subdevs was introduced to V4L2:

- Represent a single block (usually not DMA-capable)
- Exposed to userspace via dedicated nodes /dev/v4l-subdev0
- Dedicated format configuration, implemented in struct v4l2_subdev_pad_ops
- Dedicated stream management, implemented in struct v4l2_subdev_video_ops
- Called by video devices with v4l2_subdev_call
Subdevs need to be parented to a v4l2 device (controlling entity)

Simple case: the all-in-one driver

- A single driver may register a parent v4l2 device, a video device and subdev(s)
- The subdev can be registered directly:
  
  ```
  v4l2_device_register_subdev(v4l2_dev, subdev);
  ```

Complex case: multiple drivers involved

- The video device driver will typically register a v4l2 device
- Each subdev driver will register its subdev asynchronously:
  
  ```
  v4l2_async_register_subdev(subdev);
  ```
- A driver that needs a subdev needs to identify and wait for it
The fwnode graph represents the connection between different blocks:

- Typically described in device-tree with port/endpoint
- The meaning of each port is described in the device-tree bindings
- Endpoints are retrieved by the driver and parsed with a helper:
  
  ```c
  fwnode_graph_get_endpoint_by_id()
  v4l2_fwnode_endpoint_parse()
  ```

- May contain an indication of the bus type:
  ```c
elem v4l2_mbus_type, e.g. V4L2_MBUS_CSI2_DPHY
  ```

- As well as bus-specific information:
  ```c
  e.g. struct v4l2_fwnode_bus_mipi_csi2
  ```
Device-tree example for camera to MIPI CSI-2 bridge:

```
imx219: camera@10 {
    compatible = "sony,imx219";
    ...
    port {
        camera_to_bridge: endpoint {
            data-lanes = <1 2>;
            link-frequencies = /bits/ 64 <456000000>;
            remote-endpoint = <&bridge_from_camera>;
        }
    }
};

mipi_csi2: csi@1cb1000 {
    compatible = "allwinner,sun8i-v3s-mipi-csi2";
    ...
    ports {
        ...
        port@0 {
            reg = <0>;
            bridge_from_camera: endpoint {
                data-lanes = <1 2>;
                remote-endpoint = <&camera_to_bridge>;
            }
        }
        ...
    }
};
```
Async registration allows other drivers to use the subdev:

- A link between devices is described with `fwnode graph`
- An async notifier will match and notify when the subdev is available:
  ```c
  v4l2_async_notifier_add_fwnode_remote_subdev
  ```
- The async notifier can be used by the driver with a v4l2 device:
  ```c
  v4l2_async_notifier_register(v4l2_dev, notifier);
  ```
- Or by a subdev that needs another subdev (e.g. a bridge):
  ```c
  v4l2_async_subdev_notifier_register(subdev, notifier);
  ```
- A callback gives the requesting driver a `struct v4l2_subdev`
The **media controller** API provides coordination between blocks:

- Each block is an **entity** with sink/source **pads**
derivated from a video device or a subdev

- Entities declare a particular function
e.g. `MEDIA_ENT_F_PROC_VIDEO_PIXEL_FORMATTER`

- **Links** between pads of entities are created by drivers,
  may allow userspace to enable/disable them

- Grouped in a media device (tied to a v4l2 device)

- Performs **runtime validation** for links, implemented in
  `struct media_entity_operations`'s `link_validate`

- Topology is **exposed to userspace**, usually controlled with `media-ctl`:
  `media-ctl -l "sun6i-csi-bridge":1 -> "sun6i-csi-capture":0[1]`
The i.MX capture driver’s media topology
Specific aspects related to ISPs:

- Usually have an internal pipeline with **multiple blocks**
- Parameters are **highly specific** (not a good fit for V4L2 controls)
- Provide stats **information buffers** (3A, histogram)
- Exposes one or multiple **capture interfaces**

ISPs integration in V4L2:

- Processor represented by a subdev/media entity: `MEDIA_ENT_F_PROC_VIDEO_ISP`
- **Capture video devices** for pixels: queues with type `V4L2_BUF_TYPE_VIDEO_CAPTURE`
- **Meta output video devices** for parameters: queue with type `V4L2_BUF_TYPE_META_OUTPUT` with dedicated (struct) buffer type
- **Meta capture video devices** for stats: queue with type `V4L2_BUF_TYPE_META_CAPTURE` with dedicated (struct) buffer type
Example driver: **rkisp1**

- `rkisp1_isp` subdev device to coordinate
- `rkisp1_mainpath`, `rkisp1_selfpath` giving pixels, with resizers
- `rkisp1_params` taking `struct rkisp1_params_cfg`
- `rkisp1_stats` giving `struct rkisp1_stat_buffer`
Accomplished Work for Advanced Camera support on Allwinner
A31/V3 and A83T MIPI CSI-2 Support

- MIPI CSI-2 controllers feed (raw) data to the CSI controller
- Represented as bridges (subdevs) between CSI and the sensor
- Requires adaptation to the CSI code to select interface
- Needs to get sensor pixel rate from dedicated control: V4L2_CID_PIXEL_RATE
- Using a D-PHY block with the generic Linux PHY API
  - phy_mipi_dphy_get_default_config helper not accounting for DDR

A83T Support:

- Reference source code in Allwinner SDK:
  drivers/media/video/sunxi-vfe/mipi_csi/bsp_mipi_csi.c
- Some magic values in registers (undocumented)
- D-PHY is mixed with controller registers
  - In-driver PHY provider and consumer
A31/V3 Support:

- **Reference source code** in Allwinner SDK:
  drivers/media/video/sunxi-vfe/mipi_csi/{protocol,dphy}

- **Documentation** available in A31 user manual

- Same D-PHY block used for MIPI DSI, in Rx mode instead of Tx

- Driver already exists for Tx, needs direction selection:
  - Describe with submode? Not a run-time decision...
  - Describe with different compatible? Same hardware block...
  - Describe with optional device-tree property
V3 and A83T MIPI CSI-2 Support: Patch Series

- First iteration sent out in October 2020
- Merged in v6.0 in June 2022

```plaintext
25 files changed, 2633 insertions(+), 141 deletions(-)
```
ISP Support and Integration

Input/output aspects:
- ISP takes (raw) data from one of the **CSI controller(s)**
- DRAM input exists in theory but unable to make it work
- Input/interface part of CSI controller needs to be configured
- **Internal mux** routes data to ISP instead of CSI DMA
  - Impossible to switch back to CSI DMA without reboot
- **Two outputs** available: main-channel and sub-channel

**Major CSI rework** required:
- Separate bridge from DMA engine (subdev and video device)
- Register with ISP’s v4l2/media devices for common topology
- Allow standalone use (both with and without ISP enabled):
  `sun6i_csi_isp_detect` helper
ISP Support and Integration: Topology

CSI components:
- sun6i-csi-bridge
- sun6i-csi-capture

ISP components:
- sun6i-isp-proc
- sun6i-isp-params
- sun6i-isp-capture

MIPI CSI-2 interface:
- sun6i-mipi-csi2
- sun8i-a83t-mipi-csi2
Parameters configure **modules of the ISP**:

- Passed via `sun6i-isp-params` video device
- uAPI structure: `struct sun6i_isp_params_config`
- Applied to next load buffer update

**Supported features:**

- **Bayer coefficients**, with R/GR/GB/B gain/offset:
  `struct sun6i_isp_params_config_bayer`
- **2D noise filtering** (BDNF) coefficients for G and R/B:
  `struct sun6i_isp_params_config_bdnf`
- Submitted to **staging** since a stable uAPI needs all features covered
ISP Driver and Integration: Patch Series

- First iteration sent out in September 2021
- Merged in v6.2 in November 2022

drivers/media/platform/sunxi/sun6i-csi/sun6i_csi.c | 1051 ++++++++++++++++++++++
drivers/media/platform/sunxi/sun6i-csi/sun6i_csi.h | 155 ++---
drivers/media/platform/sunxi/sun6i-csi/sun6i_csi_bridge.c | 895 ++++++++++++++++++++4
drivers/media/platform/sunxi/sun6i-csi/sun6i_csi_bridge.h | 64 ++
drivers/media/platform/sunxi/sun6i-csi/sun6i_csi_capture.c | 1094 ++++++++++++++++++++4
drivers/media/platform/sunxi/sun6i-csi/sun6i_csi_capture.h | 73 +++
drivers/media/platform/sunxi/sun6i-csi/sun6i_csi_reg.h | 364 ++++++++++
drivers/staging/media/sunxi/sun6i-isp/sun6i_isp.c | 577 +++++++++++++++++++
drivers/staging/media/sunxi/sun6i-isp/sun6i_isp.h | 86 +++
drivers/staging/media/sunxi/sun6i-isp/sun6i_isp_capture.c | 759 +++++++++++++++++++
drivers/staging/media/sunxi/sun6i-isp/sun6i_isp_capture.h | 79 +++
drivers/staging/media/sunxi/sun6i-isp/sun6i_isp_params.c | 571 +++++++++++++++++++
drivers/staging/media/sunxi/sun6i-isp/sun6i_isp_params.h | 53 ++
drivers/staging/media/sunxi/sun6i-isp/sun6i_isp_proc.c | 598 +++++++++++++++++++
drivers/staging/media/sunxi/sun6i-isp/sun6i_isp_proc.h | 61 ++
drivers/staging/media/sunxi/sun6i-isp/sun6i_isp_reg.h | 275 ++++++++
drivers/staging/media/sunxi/sun6i-isp/uapi/sun6i-isp-config.h | 43 ++

51 files changed, 8702 insertions(+), 1808 deletions(-)
Future Work and Improvements
Remaining Features to Implement

Roadmap for ISP driver completeness:

- Support **more platforms** (at least A83T)
- Declare **hardware revisions** (modules availability):
  `media_dev->hw_revision`
- Support for **stats** (hist/ae/awb/af/afs)
- Support for **sub-channel, scaling and rotation**
- **Complete uAPI** that describes all modules
- Support for **all available modules**
  - Start with black level correction, color matrix and gamma
- Userspace **3A algorithms** support
Integration with libcamera

- Community-driven project for advanced camera support: **libcamera**
- Provides **abstraction** for applications, GStreamer, Android
- Implements **complex pipeline support**
- Implements **hardware-specific 3A algorithms**
- Good fit for Allwinner A31 ISP userspace support
We’re hiring!

- Embedded Linux / Linux kernel engineers
  - Linux BSPs, Linux kernel drivers, U-Boot, Buildroot, Yocto
  - Open-source contributions
  - Conferences
  - Offices in Lyon, Toulouse, or full remote
  - 2+ years of experience

- Internships for computer science engineering students

- [https://bootlin.com/company/careers/](https://bootlin.com/company/careers/)
Questions? Suggestions? Comments?

Paul Kocialkowski
paul@bootlin.com

Slides under CC-BY-SA 3.0
https://bootlin.com/pub/conferences/
Extra Slides
Optical systems have multiple elements and purposes:

- **Lens** to make light converge towards sensor surface
  - **Focal length** \( f \) indicates the amount of convergence
  - Sets the angle of view, results in magnification/zoom effect
  - Optional moving elements to define focus plane
- Optional **focus coil** to electrically control focus adjustment
- Optional **diaphragm** to control aperture
  - **F-number** \( f/1.8 \) indicates how open the diaphragm is
  - Aperture decreases with f-number (diaphragm closes)
Camera Optical Systems: Illustration

Camera optical system

Diaphragm aperture variation (CC BY-SA 3.0, KoeppiK, Wikimedia Commons)
Components of an image sensor:

1. Color Filter Array (CFA) following a **Bayer pattern** (R/G/G/B)
2. Photo-sensitive cells (**photosites**) in CMOS or CCD technology
3. **Amplifier and ADC** to produce digital values
   - Generally 8, 10 or 12-bit data
4. Configurable **shutter speed** (exposure time)
5. **Clocks and timings** for frame rate
   - Capture cycle repeatedly following precise timings
   - External clock reference for internal PLLs
   - Limits exposure time
6. **Processing** (more or less advanced)
7. Control and **configuration** interface
   - Usually configured via I2C or SPI
8. Data **transmission** interface
Image Sensors: Illustration

OV5648 block diagram (Omnivision)

Bayer pattern (CC BY-SA 3.0, Cburnett, Wikimedia Commons)
Parallel and MIPI CSI-2 interfaces on the S3-OLinuXino