Camera Sensors Integration and Tuning in Linux Systems

Presented at Open Source Summit Japan 2025

2025-12-10, 東京, 日本

こんにちは。

My name is Laurent Pinchart

- Founder and CEO of **Ideas on Board**
- Electrical engineer and software developer
- 20+ years of experience with **Linux** and cameras
- Linux kernel core contributor and maintainer
- Lead architect of the **libcamera** project

I am  and live in 

✉ laurent.pinchart@ideasonboard.com

9423 1B98 0100 EC61 9AC1 0E10 F045 C2B9 6991 256E

Introduction

We make cameras work

Ideas on Board provides embedded development services for camera and multimedia within the Linux open source ecosystem.

- 11 developers in 8 countries
- Recognized core contributor to the Linux kernel
- Open, upstream-first philosophy

Introduction

Where this talk comes from

Introduction

Today we will discuss

- Camera modules and hardware interfaces
- Linux kernel drivers
- Devicetree and ACPI integration
- Clocks configuration
- Testing and troubleshooting
- Image capture (libcamera, GStreamer and PipeWire)
- Camera tuning

Agenda

Do not hesitate to ask questions

If anything is unclear in this presentation, please feel free to ask questions during the talk. You will be be rewarded with Finnish chocolates.

If you are too shy, you can ask questions at the end, or come and talk to me after the talk. There will be chocolates too.

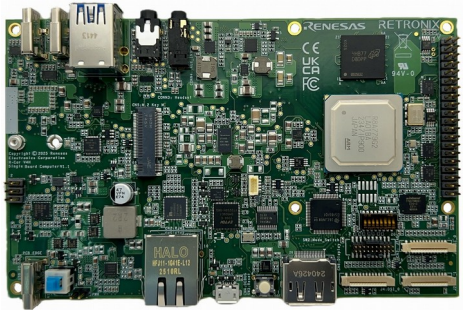
Agenda

Problem statement

To promote a fully open ecosystem around the **R-Car V4H** SoC, **Renesas** wants to enable usage of the Raspberry Pi v3 camera module based on a **Sony IMX708** image sensor with a **Sparrow Hawk** SBC.

SONY

RENESAS



RETRONIX
The Solutions for Connected World

Camera integration

IDEAS ON BOARD

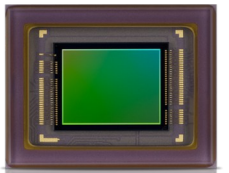
Camera Modules and Hardware Interfaces

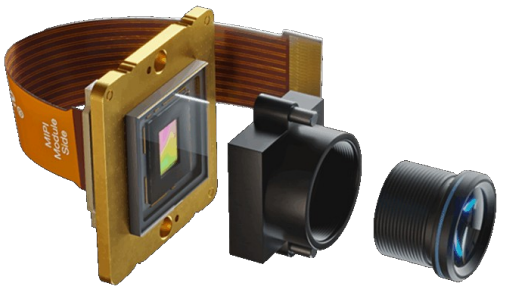
Terminology

An **image sensor**, also known as camera sensor, is an integrated circuit that includes a pixel array sensitive to light.

A **camera module** is an image sensor integrated with a lens, focus motor, IR filter, control electronics and a circuit board.

Camera modules come in different shapes and sizes, with different features.





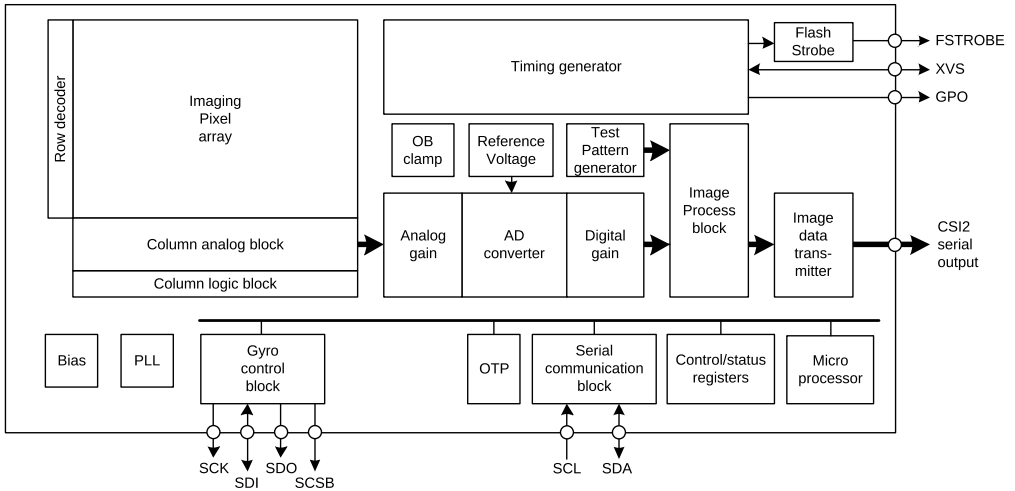
Camera Modules

IDEAS ON BOARD

Hardware interfaces

An image sensor typically has

- Power supplies
- An input clock
- Control GPIOs
- A control interface
- A data interface



Camera Modules

Hardware interfaces

The **control interface** is a low-speed connection to access sensor registers.

- I²C, a two-wire bi-directional interface.
- SCCB, a variant of I2C from OmniVision.
- SPI, a four-wire bi-directional interface.
- MIPI I³C, a higher speed successor to I²C with limited market adoption.
- MIPI CSI-2 Unified Serial Link (USL), a CSI-2 feature that allows carrying control data over CSI-2 data lanes during blanking intervals.

Other options likely exist. **I²C** (including SCCB) is by far the most common control interface.

Camera Modules

Hardware interfaces

The **data interface** is a high-speed connection that carries image and ancillary data.

- MIPI CSI-2, a serial interface with differential (D-PHY) or trio (C-PHY) signalling.
- “Parallel”, a de facto standard with discrete synchronization signals.
- BT.656, a parallel interface with synchronization embedded in the data stream.
- SLVS-EC, a proprietary serial interface from Sony.

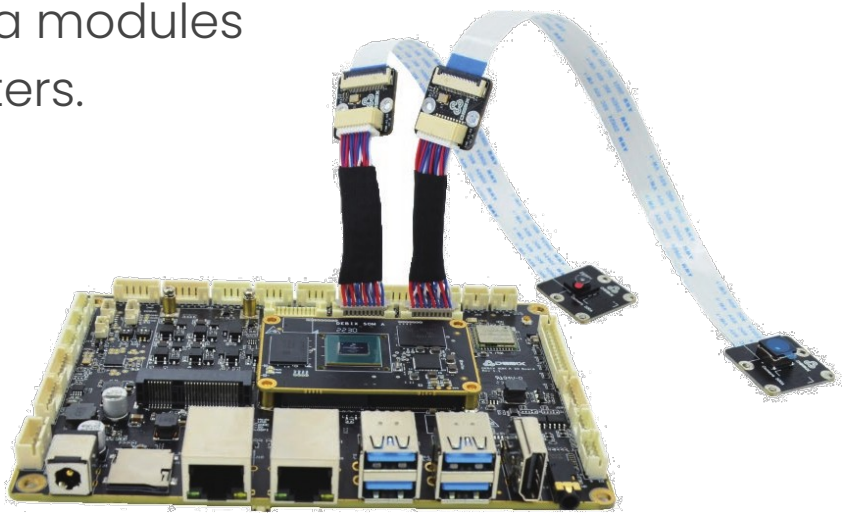
Most sensors in embedded devices use **MIPI CSI-2** (with D-PHY or C-PHY).

A number of interfaces have been standardized and deprecated over time, including SMIA CCP2, MIPI CSI-1 (a defunct predecessor to CSI-2) and MIPI CSI-3 (a defunct successor to CSI-2).

Camera Modules

Connectors

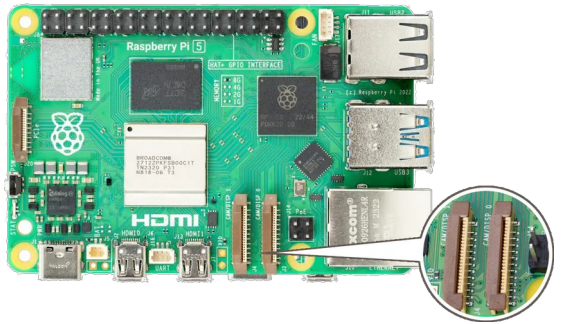
The industry has not standardized any connector for MIPI CSI-2 camera modules. While most vendors use **FPC cables**, the number of pins, pitch and pinout differ. This hinders interoperability between camera modules and boards, often requiring the use of adapters.

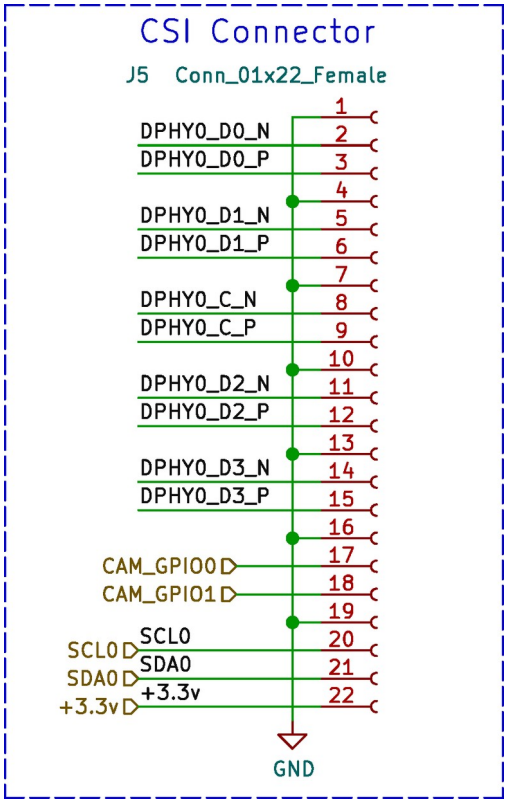


Camera Modules

Connectors

The Raspberry Pi 22-pin, 4 data lanes camera connector is becoming a de facto standard for many popular development boards.



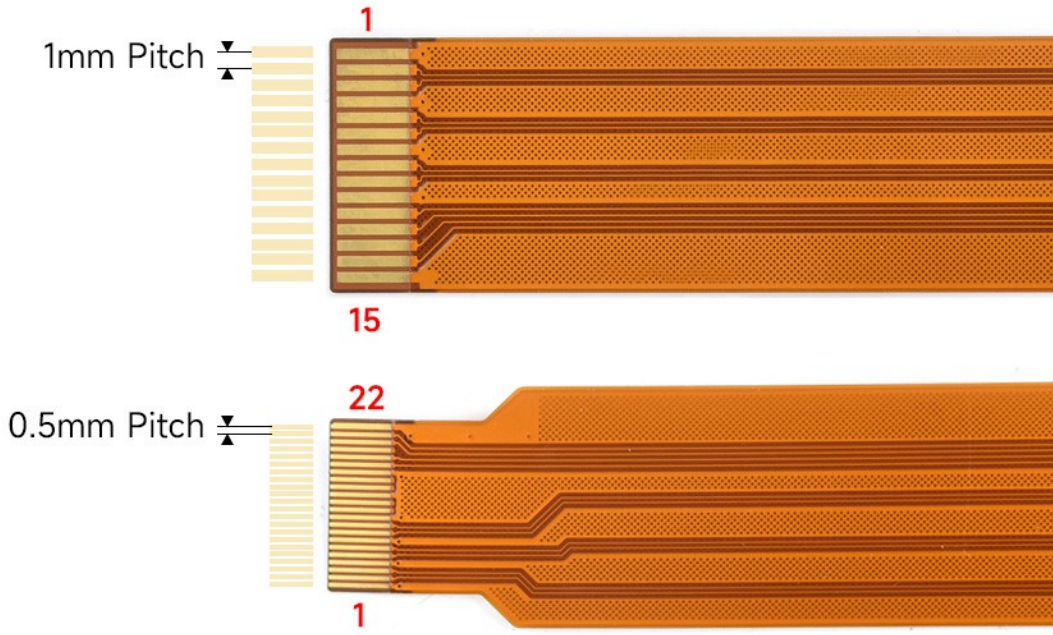


Mind the pinout: The Raspberry Pi schematics pin numbers are inverted compared to the connector vendor pin numbers.

Camera Modules

Connectors

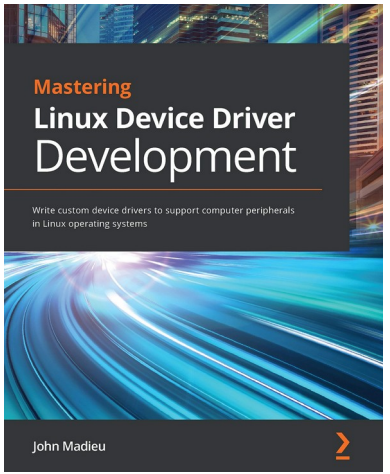
Camera modules sold by Raspberry Pi use the 15-pin, 2 data lanes connector. They can be interfaced using an off-the-shelf 15-to-22 pins adapter cable.



Camera Modules

Linux Kernel Drivers

Driver availability



Depending on the image sensor, drivers can be found

- In the mainline kernel.
- In a BSP kernel (same or different platform vendor).
- In an obscure unmaintained github repository.
- Nowhere.

The further a driver is from mainline, the more it risks being incompatible with the target platform or camera module.

The **Sony IMX708** has a driver in the **Raspberry Pi kernel**. The driver is close to mainline and is easy to port to the latest upstream kernel version. Ideally, image sensor vendors should upstream drivers for their devices.

Driver development

How to develop a new driver for an image sensor is out of scope for this presentation. The topic has been covered in multiple presentations, including **"Bring Your Camera into 2018: Forward Porting Image Sensor Driver"**[1] and **"Camera Sensor Drivers Compliance"**[2] by Jacopo Mondi.

The Linux media community is ready to help, do not hesitate to reach out to the **linux-media@vger.kernel.org** mailing list, or the **#linux-media** IRC channel on oftc.net.

1. https://www.youtube.com/watch?v=PJVlvUf0gP4
2. https://www.youtube.com/watch?v=Cn1cHguVaLk

Devicetree and ACPI Integration

System description

I²C and MIPI CSI-2 are non-discoverable buses. The Linux kernel needs to be told what and how devices are connected. This information is provided by **platform firmware**. The two most common types of platform firmware for devices that integrate image sensors are **Devicetree** (Arm, RISC-V, …) and **ACPI** (x86).

Devicetree bindings

The structure and syntax of the devicetree is defined in the **Devicetree specification**[1]. It is supplemented by **devicetree bindings** that define device-specific properties.

Devicetree bindings are OS-agnostic, but are typically developed and upstreamed at the same time as the corresponding Linux kernel driver. They are found in the Linux kernel source tree[2].

1. https://github.com/devicetree-org/devicetree-specification/releases/tag/v0.4
2. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/devicetree/bindings

Devicetree overlay

A devicetree overlay is a devicetree **fragment** that can be applied to a base devicetree. Overlays are used to describe **external modules** that are not part of the base board, such as camera modules or extension hats.

To describe a camera module, a corresponding overlay needs to be written, compiled, and loaded. Overlays are typically loaded by the **boot loader** before the kernel is started. Support for loading them in Linux at runtime is limited but progressing[1].

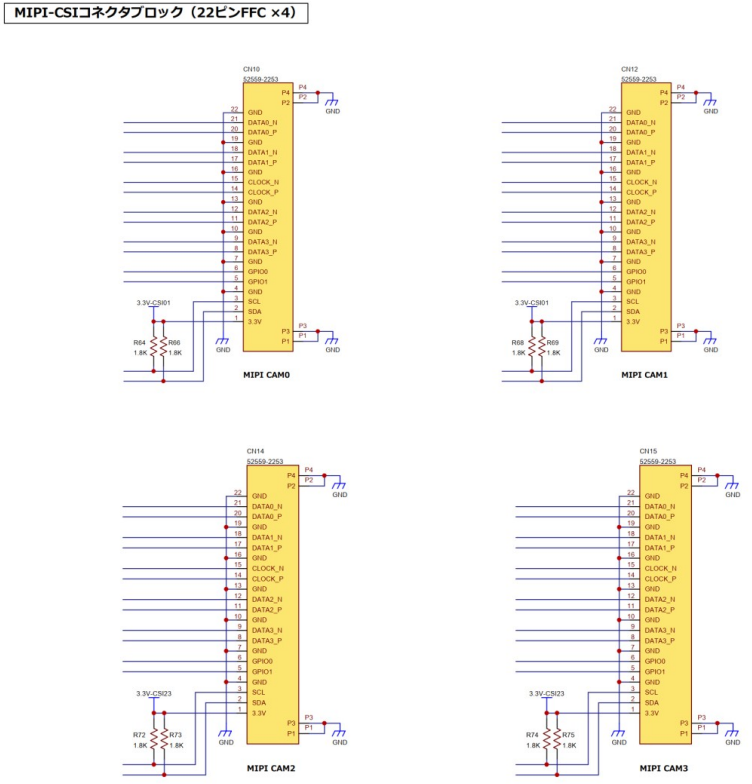
1. https://www.youtube.com/watch?v=msmBQBSyZZ4

Schematics

Connections between the SoC and the camera connectors are found in the board's **schematics**. The full schematics is not always available, reduced schematics that focus on connectors is often provided in user manuals.

CSI x 4

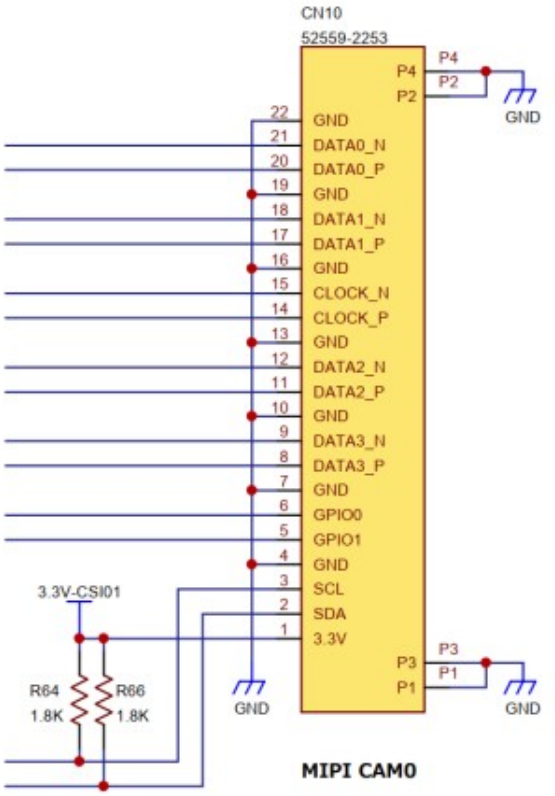
Pin	Name
1	3.3V
2	SDA
3	SCL
4	GND
5	GPIO1
6	GPIO0
7	GND
8	DATA3_P
9	DATA3_N
10	GND
11	DATA2_P
12	DATA2_N
13	GND
14	CLOCK_P
15	CLOCK_N
16	GND
17	DATA1_P
18	DATA1_N
19	GND
20	DATA0_P
21	DATA0_N
22	GND



Firmware Integration

Schematics

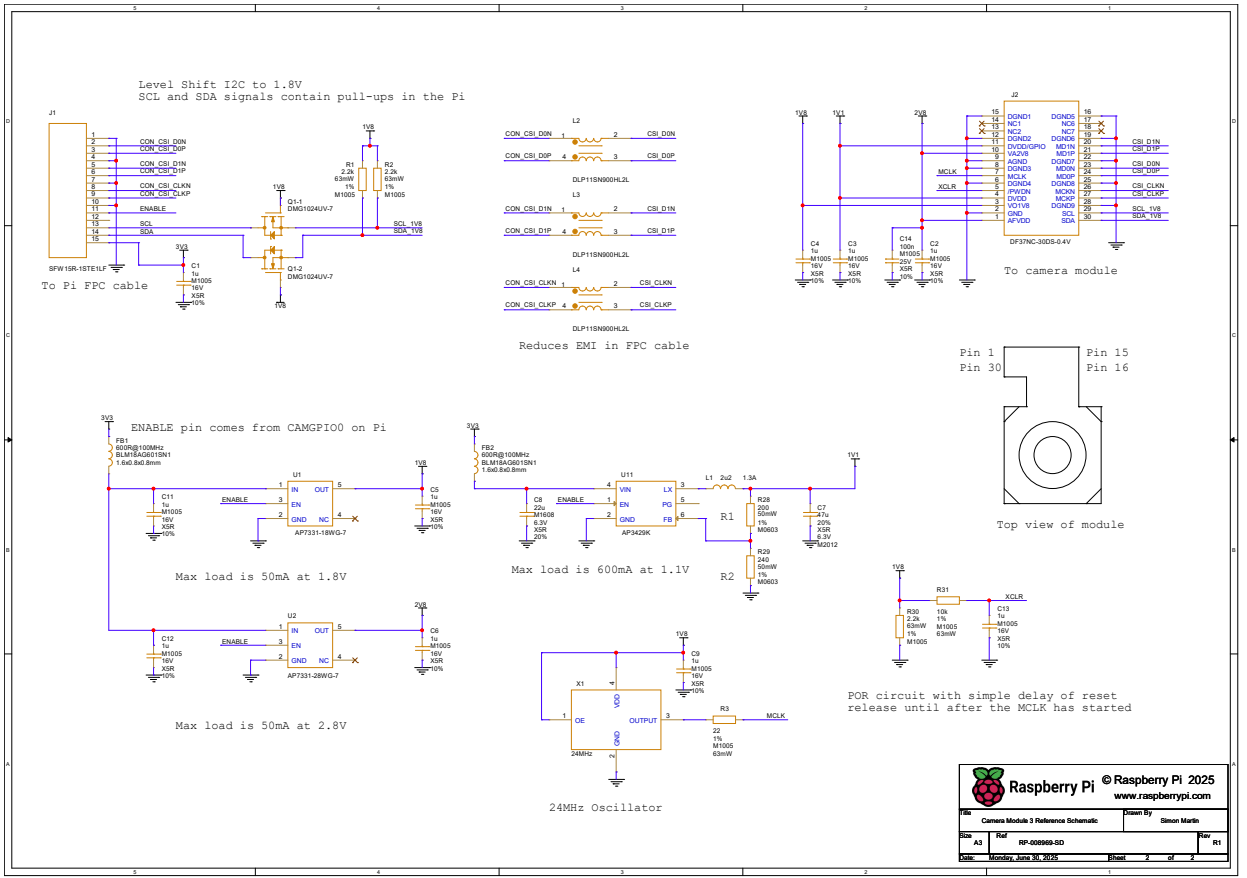
Even reduced schematics don't always provide the required information. Alternative strategies include searching for devicetree sources in **BSP kernels**, asking information by **contacting the vendor**, visually or electrically **tracing signals** on the board, or using **brute-force** by trying all possible option.



Firmware Integration

Schematics

The Raspberry Pi camera module v3 schematics is public. It shows that the on-board clock generator and power supplies.



https://datasheets.raspberrypi.com/camera/camera-module-3-schematics.pdf

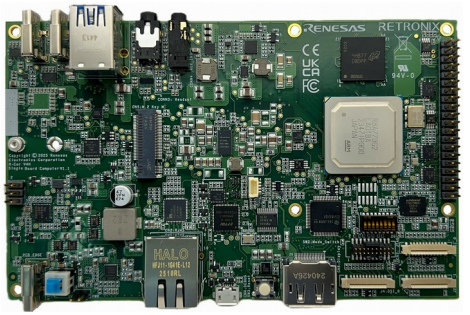
Firmware Integration

Connection to the SoC

More information is required to write a devicetree overlay:

- I²C bus number
- GPIO connected to the power regulator
- CSI-2 port

The Sparrow Hawk schematics and user manual are not currently public. The board vendor needs to provide this information.



Firmware Integration

IDEAS ON BOARD

```
/dts-v1/;
/plugin/;

#include <dt-bindings/gpio/gpio.h>
#include <dt-bindings/media/video-interfaces.h>

&{/} {
        clk_cam_j1: clk-cam-j1 {
                compatible = "fixed-clock";
                #clock-cells = <0>;
                clock-frequency = <24000000>;
        };

        /* Page 29 / CSI_IF_CN / J1 */
        reg_cam_j1: reg-cam-j1 {
                compatible = "regulator-fixed";
                regulator-name = "cam-J1";
                enable-active-high;
                gpios = <&gpio0 1 GPIO_ACTIVE_HIGH>;
        };
};
```

The clock and power supplies are declared at the root of the device tree. The enable GPIO comes from the Sparrow Hawk schematics.

Firmware Integration

IDEAS ON BOARD

```
23 &i2c1 {
24         #address-cells = <1>;
25         #size-cells = <0>;
26         status = "okay";
27
28         imx708: sensor@1a {
29                 compatible = "sony,imx708";
30                 reg = <0x1a>;
31
32                 clocks = <&clk_cam_j1>;
33                 clock-names = "inclk";
34                 vana1-supply = <&reg_cam_j1>;
35
36                 orientation = <2>;
37                 rotation = <0>;
38
39                 port {
40                         imx708_j1_out: endpoint {
41                                 remote-endpoint = <&csi40_in>;
42                                 clock-lanes = <0>;
43                                 data-lanes = <1 2>;
44                                 link-frequencies = /bits/ 64 <450000000>;
45                         };
46                 };
47         };
48 };
```

The sensor is declared as a child of the I²C bus. The connection to the CSI-2 interface is described using an endpoint. The I²C bus number and CSI-2 interface port come from the Sparrow Hawk schematics.

Firmware Integration

```
/* Page 29 / CSI_IF_CN */
&csi40 {
        status = "okay";

        ports {
                #address-cells = <1>;
                #size-cells = <0>;

                port@0 {
                        reg = <0>;

                        csi40_in: endpoint {
                                bus-type = <MEDIA_BUS_TYPE_CSI2_DPHY>;
                                clock-lanes = <0>;
                                data-lanes = <1 2>;
                                remote-endpoint = <&imx708_j1_out>;
                        };
                };
        };
};
```

The CSI-2 receiver is enabled and the connection to the image sensor also described by an endpoint.

Firmware Integration

```
73 &isp0 {
74         status = "okay";
75 };
76
77 &vin00 {
78         status = "okay";
79 };
80
81 &vin01 {
82         status = "okay";
83 };
84
85 &vin02 {
86         status = "okay";
87 };
88
89 &vin03 {
90         status = "okay";
91 };
92
```

Other IP cores of the camera pipeline need to be enabled. This information comes from the R-Car V4H manual.

Firmware Integration

ACPI

In ACPI-based systems, the hardware is described in a firmware table named **DSDT**. The ACPI specification standardizes how I^2C devices are enumerated, but this alone does not provide all the information necessary to enable the imaging hardware.

The **MIPI DisCo for Imaging**[1] specification fills the gap by standardizing what information is conveyed from the ACPI firmware and how it is encoded. Many vendors however still implement vendor-specific ACPI properties.

1. https://www.mipi.org/mipi-disco-for-imaging-download

Firmware Integration

Clocks Configuration

Clock tree

Image sensors include a clock tree with one or multiple PLLs that generate internal and output clocks from an input clock.

The **input clock** is generated by a fixed or controllable oscillator, and its frequency is typically set to a fixed value.

The **output clock** frequency needs to accommodate **bandwidth** requirements and **EMC** constraints, and multiple frequencies can be used in a system for different sensor modes.

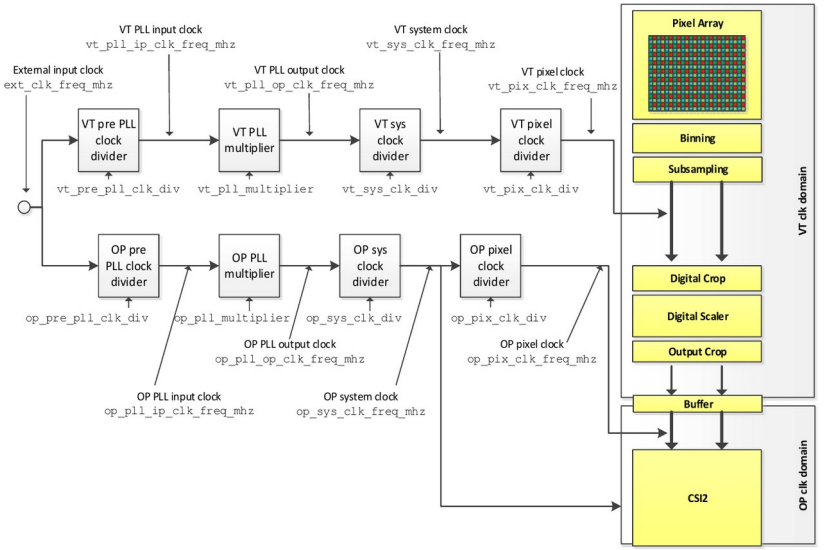


Figure 47 Two PLL with Lane Speed Model

Clocks Configuration

```
 7 &{/} {
 8         clk_cam_j1: clk-cam-j1 {
 9                 compatible = "fixed-clock";
10                 #clock-cells = <0>;
11                 clock-frequency = <24000000>;
12         };
..
21 };
..
23 &i2c1 {
..
28         imx708: sensor@1a {
..
39                 port {
40                         imx708_j1_out: endpoint {
41                                 remote-endpoint = <&csi40_in>;
42                                 clock-lanes = <0>;
43                                 data-lanes = <1 2>;
44                                 link-frequencies = /bits/ 64 <450000000>;
45                         };
46                 };
47         };
48 };
```

The input clock frequency is specified in the DT clock node, using the **clock-frequency** property for fixed clocks or the **assigned-clock-rates** property for configurable clocks,

Clocks Configuration

```
 7 &{/} {
 8         clk_cam_j1: clk-cam-j1 {
 9                 compatible = "fixed-clock";
10                 #clock-cells = <0>;
11                 clock-frequency = <24000000>;
12         };
..
21 };
..
23 &i2c1 {
..
28         imx708: sensor@1a {
..
39                 port {
40                         imx708_j1_out: endpoint {
41                                 remote-endpoint = <&csi40_in>;
42                                 clock-lanes = <0>;
43                                 data-lanes = <1 2>;
44                                 link-frequencies = /bits/ 64 <450000000>;
45                         };
46                 };
47         };
48 };
```

The output clock frequencies are specified in the DT image sensor endpoint node using the **`link-frequencies`** property.

Clocks Configuration

Clock tree

Choosing appropriate link frequencies needs to take into account the capabilities of the sensor. The values specified in the device tree must **match exactly** frequencies that can be produced by the PLLs.

PLL calculations are complex, so many drivers **hardcode** register values for one or a few input and output frequencies. Better sensor drivers compute the PLL parameters **dynamically**.

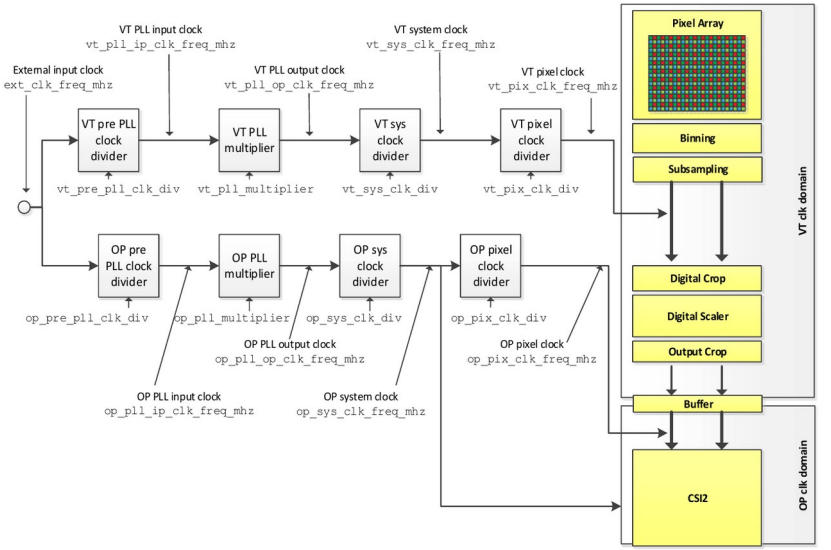


Figure 47 Two PLL with Lane Speed Model

Clocks Configuration

IDEAS ON BOARD

Testing and Troubleshooting

Media graph

The **media graph** is a representation of the camera pipeline hardware topology. It can be retrieved from the kernel drivers with the media-ctl tool, and converted to a diagram.

```
$ media-ctl -d /dev/media0 --print-dot | dot -Tpdf > graph.pdf
```

A human-readable detailed description can also be retrieved with

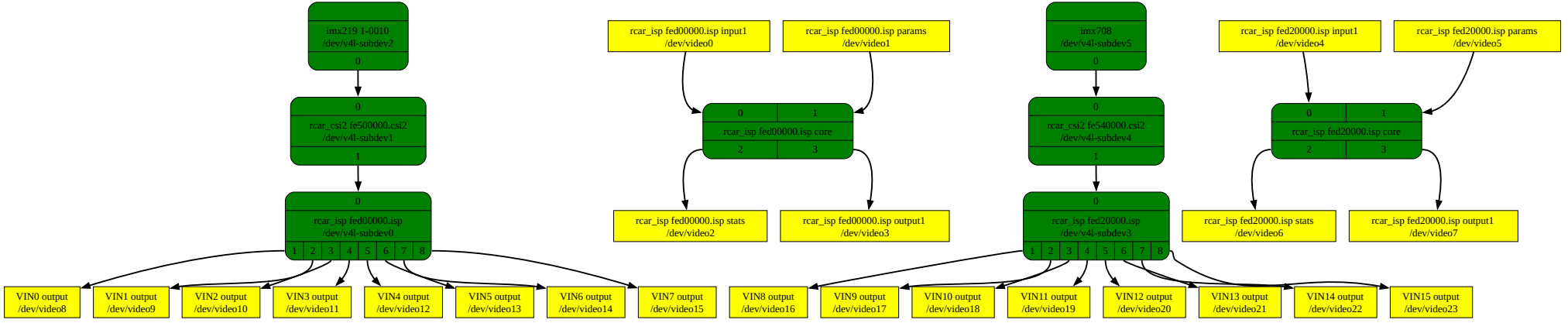
```
$ media-ctl -d /dev/media0 -p
```

The name of the media device node corresponding to the camera pipeline may vary depending on the other media devices present in the system.

Testing and Troubleshooting

When everything goes right

If the image sensor is successfully probed, it will appear in the **media graph** of the camera pipeline.



Testing and Troubleshooting

When something goes wrong

If the image sensor does not appear in the media graph, troubleshooting involves checking if

- The devicetree node is present and available.
- The Linux kernel device has been created.
- The driver has been loaded and the device probed.

Devicetree node

The devicetree structure is exposed by the kernel in sysfs. Each DT node is modelled by a directory, and each property by a file.

```
$ ls -l /sys/firmware/devicetree/base/soc/i2c@e6510000/sensor@1a
clock-names
clocks
compatible
name
orientation
phandle
port
reg
rotation
status
vana1-supply
```

Verify that the node corresponding to the image sensor is present and that its status property is not “disabled”.

Testing and Troubleshooting

Kernel device

If the devicetree node is present and not disabled, the kernel will create a corresponding device, also exposed in sysfs.

```
$ ls -l /sys/bus/i2c/devices/2-001a
driver -> ../../../../../../bus/i2c/drivers/imx708
modalias
name
of_node -> ../../../../../../firmware/devicetree/base/soc/i2c@e6510000/sensor@1a
power
subsystem -> ../../../../../../bus/i2c
supplier:platform:reg-cam-j2 -> ../../../../../virtual/devlink/platform:reg-cam-j2--i2c:2-001a
supplier:regulator:regulator.0 -> ../../../../../virtual/devlink/regulator:regulator.0--i2c:2-001a
supplier:regulator:regulator.4 -> ../../../../../virtual/devlink/regulator:regulator.4--i2c:2-001a
uevent
video4linux
```

Verify that the device is present, and that it has a link named “driver” pointing to the correct driver.

Testing and Troubleshooting

Device driver

If no driver link is present, check for the most common problems:

- Is the expected driver loaded and does it appear in /sys/module ?
- Does the DT compatible string, as read from the “compatible” property in sysfs, match the driver?
- Has probe failed (usually reported by a message in the kernel log)?
- Does /sys/kernel/debug/devices_deferred report probe deferral?

If none of these help, ask for support with a clear description of what you have tried.

Image Capture

Manual capture

Capturing images from the camera involves setting up the complete camera pipeline. Depending on the hardware architecture, this can be a simple or complex operation.

If no ISP is present in the pipeline, manual set up and image capture can be performed respectively with the media-ctl and v4l2-ctl command line utilities. The procedure is detailed in **“Application usage of media controller devices”**[1].

1. https://git.ideasonboard.org/doc/mc-v4l2.git/

libcamera

libcamera is a framework and library to support complex cameras in Linux-based systems. It ties together all the components in the camera pipeline, and implements control algorithms for the camera sensor and ISP.

The “cam” utility can detect cameras and capture images from the command line.

```
$ cam -l
1: 'imx708' (imx708 2-001a)
```

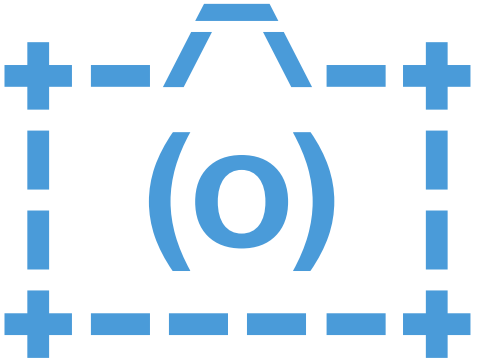


Image Capture

IDEAS ON BOARD

Camshark

Camshark is a test, debug and development application for libcamera. It runs on a host machine and connects to the target device through ssh.

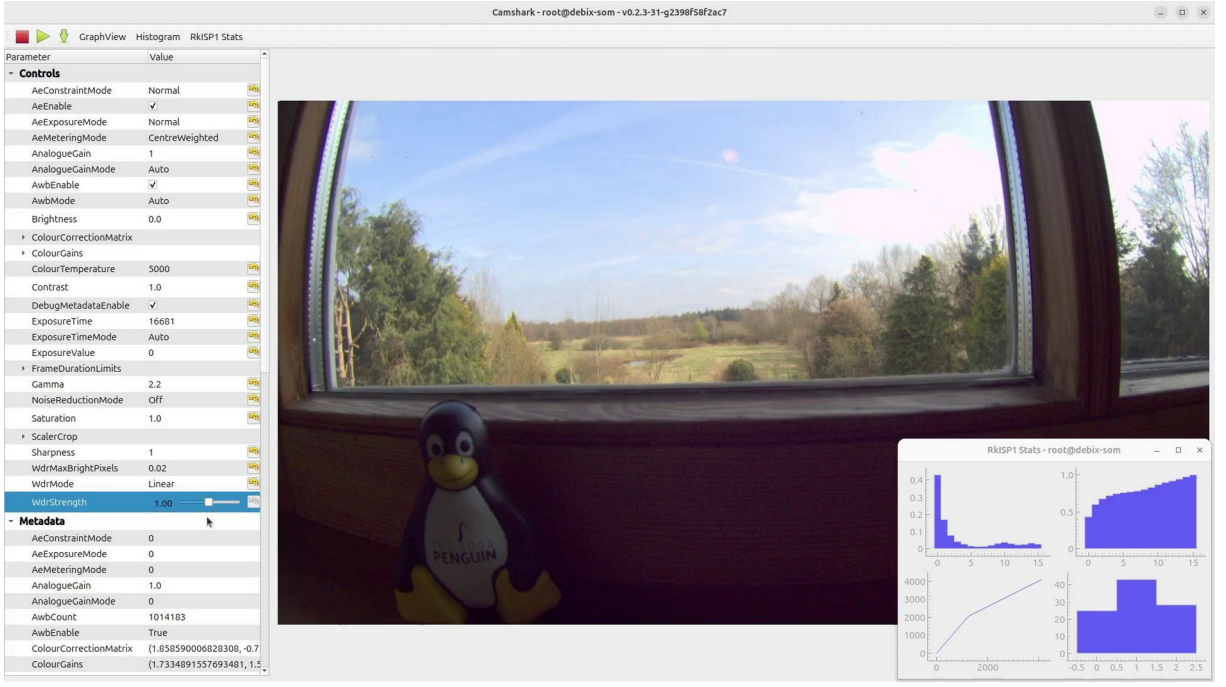


Image Capture

GStreamer

libcamera includes a GStreamer source element named libcamerasrc. Using GStreamer is recommended for applications that do not need the low-level access provided by the libcamera native API.


```
$ gst-launch-1.0 libcamerasrc camera-name="imx708" ! queue ! glimagesink
```

More information about GStreamer usage is available in the libcamera documentation.

Image Capture

PipeWire

PipeWire is multimedia framework for handling audio and video streams on Linux. It is the recommended framework to access cameras on modern Linux desktop systems and provides advances features such as device sharing between applications or permission management.

PipeWire includes a libcamera backend for seamless integration with libcamera-based cameras.

Camera Tuning

Image quality tuning

Camera tuning is an required step to achieve good image quality.

libcamera provides a fully open-source tuning infrastructure. Work is ongoing to extend it to all the ISPs that libcamera supports, with multiple improvements, and a new tuning guide.

For more information about tuning, attend the **“Raw to Real and Green to Great: Open Source Camera Tuning for Linux Devices with libcamera”**[1] talk that Kieran Bingham has submitted for FOSDEM 2026.

1. https://pretalx.fosdem.org/fosdem-2026/talk/review/8VM8YDCUL97PBNVL88RURGCEYVZVPEL3

Conclusion

Summary

Camera integration is

- Hard when you don’t know what to do.

Conclusion

Summary

Camera integration is

- Hard when you don’t know what to do.
- Easier when drivers are available in mainline. Tell your hardware vendors.

Conclusion

Summary

Camera integration is

- Hard when you don’t know what to do.
- Easier when drivers are available in mainline. Tell your hardware vendors.
- Traditionally closed and proprietary, but improving with libcamera.

Conclusion

Summary

Camera integration is

- Hard when you don’t know what to do.
- Easier when drivers are available in mainline. Tell your hardware vendors.
- Traditionally closed and proprietary, but improving with libcamera.
- Supported by community and experts. Ask for help.

Conclusion

Summary

Camera integration is

- Hard when you don’t know what to do.
- Easier when drivers are available in mainline. Tell your hardware vendors.
- Traditionally closed and proprietary, but improving with libcamera.
- Supported by community and experts. Ask for help.
- Better with chocolates.

Conclusion

ご清聴ありがとうございました。