Image Signal Processor (ISP) Drivers & How to merge one upstream

Helen Koike
Senior Software Engineer
About me

- @ Collabora since 2016
- Mostly working on the kernel – media subsystem:
  - Maintainer of rkisp1 driver
  - Maintainer of vimc driver
- Outreachy intern in 2015 – vimc projet
- Co-coordinator of Linux Kernel project in Outreachy
Main goal of this presentation

- Overview of Camera → ISP → Memory pipeline
- Overview of Media Framework
- Design choices when implementing a driver
- Lessons learned when upstreaming rkisp1 driver
- Userspace tools (libcamera)
Camera → ISP → Memory
Camera sensor

The Bayer arrangement of color filters on the pixel array of an image sensor

Application
What is an ISP?

- Image signal processor

Common use case:

- ISP receives the reading all those small color sensors
- Transforms in an image usable for userspace

Performs several other image transformations
Image Processing

- Format conversion (debayering, RGB, YUV)
- Crop / Resize
- White balance
- Compose
- Image stabilization
- Effects / filters
- Flip / Rotate
- etc

- Hardware accelerated image processing
- Offloads the CPU
Statistics

- ISP can generate statistics:
  - Histograms
  - Area contrast
  - etc

- Used by userspace to implement algorithms such as:
  - Histogram equalization
  - 3A (auto-focus, auto-exposure, auto-white balance)
What an ISP is not

- ISP is not a codec
- ISPs work with raw/uncompressed images
- Codecs:
  - Encoders: raw image → compressed image format
  - (such as H.264, JPEG, VP9)
  - Decoders: compressed image → raw image
ISP architectures

Inline vs Offline
Offline

• 2 phases:
  - Sensor → Memory
  - Memory → ISP → Memory

• Usually implemented in two separate drivers
  - Coordinated by userspace
  - Example Intel IPU3:
    • IPU3 CIO2 (camera interface) driver: gets the image from the sensor
    • IPU3 ImgU driver: process this image and sends to userspace
Data reaches memory only in the end:

- Sensor → ISP → Memory

Example: rkisp1 driver
Hybrid

- Can get the image directly from the sensor or from memory
- Can behave as inline, or perform the second phase of offline
- Ex: MT8183 P1
MIPI DPHY (quick overview)
**Bus – MIPI DPHY**

- Very common bus used in the market for cameras and displays
- Specified by MIPI Alliance
- Physical layer with high data-rate
- 4k images with a good frame rate
Bus - MIPI DPHY

• Up to 4 data lanes

• I2C bus for configuration

• On top of this bus there can be two protocols:
  - MIPI DSI-2: Display Serial Interface, to output images
  - MIPI CSI-2: Camera Serial Interface, to capture images

• MIPI DPHY/CSI-2 → frequent term in ISP land
Study case - RKISP1
Rockchip RK3399 ISP

• rkisp1 is the driver of the ISP block present in Rockchip RK3399 SoCs

• RK3399 SoC can be found in devices such as:
  - Scarlet Chromebooks
  - RockPi boards
  - Pinebook Pro laptops
Rockchip RK3399 ISP

- Originally written by Rockchip
- Merged in kernel 5.6
- `drivers/staging/`
- 9k+ lines of code
Rkisp1 hw architecture
Rkisp1 hw architecture

- ISP Comprises with:
  - Image Signal Processing
  - Many Image Enhancement Blocks
  - Crop
  - Resizer
  - RBG display ready image
  - Image Rotation

- Self-path: preview
- Main-path: picture
Kernel media framework
Media topology

- Linux kernel exposes a topology to userspace
- Userspace can query `/dev/mediaX`
  - Retrieve how inner blocks are interconnected
  - Order of image processing
Media topology

- Two types of nodes:
  - subdevices: inner parts of the hardware
  - video devices: dma engine, where userspace queues and de-queues buffers, containing images or metadata to/from the hardware

- Connected by links between pads

- NOTE: sensor is usually a separated driver
IPU3 CIO2 – offline – 1st phase

- imx355 10-001a
  - 0
  - ipu3-csi2 0
    - 0
    - ipu3-cio2 0
      - /dev/video0
  - 1
- ipu3-csi2 1
  - 0
  - ipu3-cio2 1
    - /dev/video1
- ipu3-csi2 2
  - 0
  - ipu3-cio2 2
    - /dev/video2
- ipu3-csi2 3
  - 0
  - ipu3-cio2 3
    - /dev/video3
IPU3 ImgU – offline – 2nd phase

- ipu3-imgu 0 input /dev/video4
- ipu3-imgu 0 parameters /dev/video5

- ipu3-imgu 0 output /dev/video6
- ipu3-imgu 0 viewfinder /dev/video7
- ipu3-imgu 0 3a stat /dev/video8

- ipu3-imgu 1 input /dev/video9
- ipu3-imgu 1 parameters /dev/video10

- ipu3-imgu 1 output /dev/video11
- ipu3-imgu 1 viewfinder /dev/video12
- ipu3-imgu 1 3a stat /dev/video13
RKISP1 - inline

ov5695 7-0036
/dev/v4l-subdev3
0

ov2685 7-003e
/dev/v4l-subdev4
0

rkisp1_isp
/dev/v4l-subdev0
0 1

rkisp1_resizer_mainpath
/dev/v4l-subdev1
0 1

rkisp1_resizer_selfpath
/dev/v4l-subdev2
0 1

rkisp1_stats
/dev/video2

rkisp1_mainpath
/dev/video0

rkisp1_selfpath
/dev/video1

rkisp1_params
/dev/video3
Driver config architecture
Auto vs Manual config propagation
Auto config propagation

Set resolution

Auto-propagation
Manual config propagation

Set resolution

ov5695 7-0036
/dev/v4l-subdev3
0

ov2685 7-003c
/dev/v4l-subdev4
0

rkisp1_params
/dev/video3

Set resolution

rkisp1_isp
/dev/v4l-subdev0
0 1
2 3

Set resolution

rkisp1_resizer_mainpath
/dev/v4l-subdev1
0

rkisp1_resizer_selfpath
/dev/v4l-subdev2
0

rkisp1_stats
/dev/video2

Set resolution

rkisp1_mainpath
/dev/video0

rkisp1_selfpath
/dev/video1
Manual config propagation

- Increases complexity for userspace

- If formats don’t match → fail on STREAMON

- Finer grain configuration in inner blocks of the hardware

- More blocks exposed, more complex

- Extendable
Why rkisp1 is manual?
Crop

- Specify a sub-rectangle in the image
Crop - rkisp1

Set sub-rectangle?
Crop - rkisp1

- rkisp1 allows cropping the image from the sensor
- rkisp1 allows cropping the image before resizing
- Exposing crop once in the video node would be confusing
Crop - rkisp1
Image stabilizer

- “Lock” sub-rectangle in the picture
- Shaking the phone won’t shake the image much
Setting sub-rectangles

- ov5695 7-0036
  /dev/v4l-subdev3
  0

- ov2685 7-003c
  /dev/v4l-subdev4
  0

- rkisp1_params
  /dev/video3

- rkisp1_isp
  /dev/v4l-subdev0
  0  1

- rkisp1_resizer_mainpath
  /dev/v4l-subdev1
  0
  1

- rkisp1_resizer_selfpath
  /dev/v4l-subdev2
  0
  1

- rkisp1_mainpath
  /dev/video0

- rkisp1_selfpath
  /dev/video1

- rkisp1_stats
  /dev/video2

Set sub-rectangle (crop)
Set sub-rectangle (imag-stab)
Phy subsystem
Rkisp1 – original topology

- ov2685 7-003c
  /dev/v4l-subdev2
  0

- ov5695 7-0036
  /dev/v4l-subdev3
  0

- rockchip-sy-mipi-dphy
  /dev/v4l-subdev1
  1

- rkisp1-input-params
  /dev/video3

- rkisp1-ispc-subdev
  /dev/v4l-subdev0

- rkisp1_selfpath
  /dev/video1

- rkisp1_mainpath
  /dev/video0

- rkisp1_statistics
  /dev/video2

Removed
Phy Abstraction Layer

- Manual config propagation → more subdevices, more complex for userspace
- Re-think exposed blocks
- Phy block → no image configuration exposed
- Topology → image processing steps
- Same processing steps can be used on top of different buses
  - ex. rkisp1: parallel (not implemented), MIPI-DPHY/CSI2
Phy – Lessons learned

• Lessons learned:

  - Migrate bus code to PHY Abstraction Layer (drivers/phy/)

  - Generic topology for any bus – less complex for userspace

  - ISP driver is much cleaner

  - Phy driver can be used for DSI
Lessons learned
Updating to staging

• V4L2 community is open to accept drivers in staging

• (with the condition that you work on it to move it out asap)

• Detailed TODO list

• Make it available to other people to use

• Improve workflow, easier to get contributions from others, testing, bug reports

• Decrease maintenance cost → no need to keep rebasing
More lessons learned

• Don't be afraid to re-organize the code (files, namings, code order, re-writing functions)

• Split the code between different files per implementation node, at least between video nodes and subdevice nodes

• Separate the code that configures the hardware, from the code that deals with the V4L2 API

• Remove code you are not using, you that you can't test, for example:
  - rk3288 support
  - phy driver ports (SoC has 2 MIPI-DPHY/CSI2 ports, I had was only using one)
  - Simplify the code – but keep extendable
  - Lots of macros in headers
Userspace support
Libcamera
Complex topologies

- Not all features are auto discoverable

- Examples (rkisp1):
  - sub-rectangle for cropping
  - vs sub-rectangle for image stabilizer
  - Meta-data buffers structure:
    - rkisp1_stats
    - rkisp1_params
Complex topologies

- Requires userspace specific implementation for specific drivers
- Specific applications to specific hardware
- Not very reusable code
- Hard to test
Libcamera

• Open source camera stack for many platforms with a core userspace library

• Userspace drivers

• Image processing algorithms
Architecture

---< \textit{libcamera Public API} >---

\begin{itemize}
\item Camera Devices Manager
\item Camera Device
\item \begin{itemize}
  \item Device-Agnostic
  \item \{
    \begin{itemize}
    \item \begin{itemize}
      \item \{ Image \}
      \item \{ Processing \}
      \item \{ Algorithms \}
    \end{itemize}
    \end{itemize}
  \end{itemize}
\end{itemize}
\item Device-Specific
\item \begin{itemize}
  \item Pipeline
  \item Handler
\end{itemize}
\end{itemize}

\begin{itemize}
\item \textit{Helpers and Support Classes}
\item \begin{itemize}
  \item MC & V4L2 Support
  \item Buffers Allocator
  \item Sandboxing IPC
  \item Plugins Manager
  \item Pipeline Runner
\end{itemize}
\end{itemize}

\begin{itemize}
\item \{ Device-Specific Components \}
\item \textit{Sandboxing}
\end{itemize}
Tips

• Add/push/update support for your hardware in Libcamera

• Easier to test

• More users

• More developers involved

• Contribute with the project
Thank you

Message {
  config {
    priority: "high"
    body: "Collabora is hiring"  // Many open positions
    recipient: "you"              // Please join us
    calltoaction: "http://col.la/join"
  }
}

Helen Koike
helen.koike@collabora.com