Supporting the Camera Interface on the C.H.I.P

Maxime Ripard
maxime@bootlin.com

© Copyright 2004-2018, Bootlin.
Creative Commons BY-SA 3.0 license.
Corrections, suggestions, contributions and translations are welcome!
Maxime Ripard

- Embedded Linux engineer and trainer at Bootlin
  - Embedded Linux development: kernel and driver development, system integration, boot time and power consumption optimization, consulting, etc.
  - Embedded Linux, Linux driver development, Yocto Project / OpenEmbedded and Buildroot training, with materials freely available under a Creative Commons license.
  - http://bootlin.com

- Contributions
  - Co-maintainer for the sunXi SoCs from Allwinner
  - Contributor to a couple of other open-source projects, Buildroot, U-Boot, Barebox

- Living in Toulouse, south west of France
Introduction
C.H.I.P.?

- 9$ SBC
- Based on an Allwinner R8 (equivalent to A13)
- 1GHz Cortex-A8 CPU
- Mali 400 GPU
- Plenty of GPIOs to bitbang stuff (and real controllers too!)
- Running mainline-ish Linux kernel (4.4 at the moment)
Development effort

- A significant part of the work already done
- But key features for a desktop-like application were missing
  - NAND support
  - Display, GPU
  - Audio, Camera, VPU
- Plus board specific developments
  - WiFi regulators
  - DIP
Video Capture in Linux
Introduced in 2002, in 2.5.46

Supports a wide range of devices

- Video Capture (Camera, tuners)
- Memory to memory devices (Hardware codecs, scalers, deinterlacers)
- Radio receivers and transceivers
- SDR
V4L2: A dumb pipeline

RAM ➔ Controller ➔ Camera Bus ➔ Camera

RAM ➔ Controller ➔ Camera Bus

Camera Bus ➔ I2C

I2C ➔ Camera
There’s a wide range of video formats...

... And even weird variations of them

Most of the time, the controller and the sensor don’t support the same set of formats

Some negotiation needs to happen between the controller and the camera to agree on a common format.
You also need to implement the streaming hooks

Addresses two things:
  - Memory Management: Buffer allocation, queuing and dequeuing
  - Streaming control

With the formats, the only really needed operations
Streaming modes

User

Other device

RAM

Controller

Camera

Camera Bus

I2C

Allocated by the driver
- Generic implementation of that streaming API
- Relies on a smaller, simpler set of callbacks to implement
- Different videobuf implementations, depending on your setup (backed by vmalloc, scatter gather DMA or contiguous DMA)
- Also has a notion of streaming modes, which control the source of the buffers, among
  - The driver
  - The user-space (if the device supports it)
  - Some other device (through DMA-BUF)
- The new callbacks are only there to tell videobuf the size and number of buffers to allocate, insert new buffers in a DMA chain, or start and stop the streaming
Your device might need additional set up for things like
- White balance
- Saturation
- Brightness
- etc.

By default, no controls are implemented, but the driver needs to declare them during probe, and handle them in a dedicated callback.
You’ll usually have two drivers:
  - One for the controller, usually in `drivers/media/platform`
  - And one for the camera, in `drivers/media/i2c`

By default, exposed to the userspace as one single device `/dev/videoX`

You need some synchronization between the two: `v4l2-async`

Very similar to what is found in ASoC or DRM

Basically a two-stage probe
State machine

- Set format
- Set controls
- Allocate buffers
  - Queue Buffers
  - Start Streaming
  - Flip Buffers
  - Stop Streaming
- Dequeue Buffers
  - Interrupt
Some formats require multi-plane support
- Depending on the format, it might need 1 to 3 buffers
- Supported in v4l through a different capture type
- The callbacks are different too, but very similar
- You basically just have to deal with more buffers
More complicated setup
When the pipeline gets more complicated, the amount of controls to expose in the video device starts to be impossible to deal with.

The media controller API allows to expose one device file per component in the pipeline.

Each of them can be accessed independently, for example with media-ctl.

It might even simplify your driver, because all the format negotiation will not be relevant anymore.
Tests!

- v4l2-compliance is awesome
- v4l2-info
- yavta
- Any v4l enabled application (Cheese?)
Future developments

Maxime Ripard
maxime@bootlin.com
Integration with DRM

- Our camera and display engines can work in the same format (but no driver support for it yet in the DRM driver).
- The display engine is even able to re-scale the video coming from the camera (but there’s no driver support for it yet).
- Finding which component in userspace could do that. Gstreamer? Something a la ALSA cards configuration files?
We have some work on-going to support the VPU on the Allwinner SoCs

- Reverse engineering
- Decoding works for some codecs and image formats
- Encoding is not really understood right now
- Figure it out and support encoding through the VPU
Questions? Suggestions? Comments?

Maxime Ripard
maxime@bootlin.com

Slides under CC-BY-SA 3.0
http://bootlin.com/pub/conferences/2016/elce/ripard-v4l