Image Capture On Embedded Linux Systems

Jacopo Mondi

FOSDEM 2018
Hello, I’m Jacopo

- jacopo@jmondi.org
- irc: jmondi freenode.net

- Linux kernel and embedded software engineer
- Renesas Linux kernel team
Light, color, pixels

Image sensor
- Anatomy
- Integration

Image Data transmission

Video4Linux2
- Basic architecture
- Memory management
- Image streaming
- Media controller APIs
Light, colors, pixels

*Color is not an absolute value*

- Neural response to physical properties of electromagnetic radiations
- Visible light has a well defined interval (390 nm to 700 nm)
- Not all species and not all humans perceive colors in the same way
The human eye is more sensitive to three frequencies:

- **short**: blue color
- **medium**: green color
- **long**: red color

Mathematical correlation between photo-optic properties and perceived color.
Light, colors, pixels

*Spectral power distribution*

- We can describe a radiant emission of visible light as the intensity of a photo-optic property in function of the frequency of its component.

**Figure:** Spectral power distribution of standard illuminants. From: commons.wikimedia.org
Light, colors, pixels

The (very simple) **LMS** color space

- Samples of neural stimulus received by the human eye
- Samples on well-known wavelength
- The **Long Medium Short** color space

**Figure:** Long-Medium-Short wavelength sampling
Light, colors, pixels

**LMS Colorspace**
- LMS is theoretical tool too simple scheme to represent real use cases

**CIE 1931** defined colorspaces:
- **RGB color space:**
  - Red Green Blue primary colors
- **XYZ color space:**
  - Luma component $Y$ and associated chrominances $x$ and $z$

Notable color spaces: sRGB, Adobe RBG, CYMK...
Colors, colors, pixels

Colorspaces and color encodings

Figure: From: commons.wikimedia.org
Color encoding schemes

- We have a mathematical model to represent values of a "color" with a tuple.
- As we live in a digital world, we can now use those values to transmit the most basic information an image is composed of: a **pixel**

**Question:** if we have to describe a single pixel with at least 3 digital values, how big would an image composed be by 1280x800 pixels?
Color encoding schemes

- We have a mathematical model to represent with a tuple of values a "color"
- As we live in a digital world, we can now use those values to transmit the most basic information an image is composed of: a pixel

Answer: 24,5Mbit with a very limited color resolution (0-255) → That’s bad
It is highly unpractical to sense all 3 color components for each pixel of a sensor’s pixel matrix

- Image resolution vs sensor size ratio
- Required bandwidth for digital information transmission
- Production costs and dimension not justified by resulting performances for most use cases
A *Bayer filter* is an arrangement of light filters on top of a CMOS sensor photo-receptors. Each ‘pixel’ transports a single color information.

**Figure:** From: http://www.cambridgeincolour.com
Light, colors, pixels

*Bayer filter*

- The full pixel color is re-constructed by *demosaic* and *interpolation* with neighbor pixels.
- Reduces the required transmission bandwidth and sensor size.

*Figure:* From: http://www.cambridgeincolour.com
Integration diagram

Diagram showing the integration of a sensor and a SoC (System on Chip) through a communication interface and a data bus. The sensor is connected to the SoC via an interface and the data bus, enabling communication and data transfer between them.
Image sensors
- Grid of CMOS photo-detectors: *Pixel Grid Array*
- Bayer pattern: RGB color filter array
- Color filter disposition: RGGB - BGGR - GRBG - RGGB etc.
Image sensors

- **Image format control:**
  - Pixel encoding: RGB555, YUV422, YUV420 etc
  - Image manipulation: Cropping, binning, zoom
  - Advances features (ie. 3A), mirroring, flipping etc
Mainly two data bus categories:
- Parallel (BT.601 - BT.656)
- MIPI Serial camera interface (MIPI CSI-2)

Parallel bus:
- Lower data rate, lower resolutions, more wires.
- Easier integration, cheaper, "easy" debug
- Usually found in industrial/automation contexts, hobbyist projects, older system in general

MIPI Serial bus:
- Higher data rate, highly integrated, less wires
- Hard to integrate, hard to debug, more expensive
- Mobile devices, cameras, and new designs in general
- **VSYNC - HSYNC/HREF**
  - vertical/horizontal synchronization signals
- **PCLK**: pixel clockout reference clock
- **8+ parallel data lanes**
MIPI CSI-2 specifications not only define the physical layer, image formats and data transmission protocol.

- Physical layers: D-PHY, C-PHY
- Data transmission protocol:
  - Media bus image formats: RGB or YUYV permutations
  - Short packets for synchronization signals (line/frame start/end)
  - Long packets for actual data with header for data description
Image sensors data interface: MIPI CSI-2

Physical layers

- **MIPI D-PHY**
  - Differential lines signals
  - Up to 1Gbps per lane
  - 1 clock signal and 1 to 4 data lanes

- **MIPI C-PHY**
  - Differential data lanes with embedded clock (3 pin)
  - Up to three "trios"
  - Up to 5.7Gbps per "trio"
Packet-oriented protocol
- Short packets for synchronization: Frame start/end - Line start/end
- Long packets for data

A data stream is a sequence of pixel data enclosed in a <FS> <FE> sequence
Each data packet is identified by a DT and a VC specified in its header
Different data stream can be multiplexed on the same physical bus

- **Data type (DT)**
  - Data type identifier as defined by CSI-2 specs
  - Different image formats interleaved in the same stream

- **Virtual channel (VC) interleaving**
  - Channel identifier: [0-3 or 0-6]
  - Multiple streams interleaved
  - Each stream can be data type interleaved
Image sensors data interface: MIPI CSI-2

Capture
DT=0x18

Metadata
DT=0x12
Image sensors data interface: MIPI CSI-2

CSI-2 MUX

VC = 0

CSI-2

capture

sensor

DT = 0x18

VC = 1

CSI-2

metadata

sensor

DT = 0x12

VC = 2

CSI-2

preview

sensor

DT = 0x1e

depth map

sensor
Video4Linux2: a basic use case

Basic use case: single sensor connected to a video receiver
Preliminary operations

- Open the video device node
- Control the video device through V4L2 IOCTL:
  - Query capabilities to make sure the device can stream
  - Set image format on platform and sensor drivers
  - Set image size on platform and sensor drivers
  - Set stream parameters (frame rate) on platform and sensor drivers
Video memory requirements

- DMA capable memory (often implies contiguity if DMA engine do not support s/g operations)
- Accessible by CPU and devices (in case of IOMMU)
- Possibly shared between different subsystems to reduce userspace copies
Three memory allocation model

- Kernel uses pointers to userspace buffers
- mmap of kernel buffers in userspace land
- Buffer sharing through DMABUF
Video4Linux2: memory allocation - mmap
Video4Linux2: memory allocation - mmap
Video4Linux2: memory allocation - mmap

- Video4Linux2 Core
- VideoBuf2
- Video4Linux2 Async
- DMA memory area
- Platform driver
- Receiver
- Sensor
- Pixel data
- i2c
- i2c messages
- DMA allocation
- User application

Diagram showing the flow of data from hardware to user application through the Video4Linux2 modules and DMA allocation.
Video4Linux2: memory allocation - mmap

User application

/dev/video0

VIDIOC_REQBUFS

Platform driver

receiver

dma_alloc()

dma memory area

v4l2 core

videobuf2

v4l2-sync

Kernel

i2c

Sensor

Pixel data

Hardware

i2c messages
Video4Linux2: memory allocation - dmabuf

User application

```
int dmabuf_fd[]
```

/dev/video0

```
drivers/media/v4l2-core/
```

v4l2 core    videobuf2    v4l2-async

Platform driver

```
receiver
```

Pixel data

```
sensor
```

```
i2c
```

```
i2c messages
```

```
kernel
```

```
DRM
```

```
alloc()
```

```
hardware
```

Jacopo Mondi - FOSDEM 2018
Video4Linux2: memory allocation - dmabuf

user application

int dmabuf_fd[]

/export fds()

drivers/media/v4l2-core/

v4l2 core  videobuf2  v4l2-async

platform driver

receiver

pixel data

sensor

i2c

i2c messages

KERNEL

DRM

alloc()

HARDWARE
Video4Linux2: memory allocation - dmabuf

Diagram showing the memory allocation process in Video4Linux2, with user application access to dmabuf через /dev/video0.
Video4Linux2: memory allocation - dmabuf

user application
int dmabuf_fd[]

export fds()

KERNEL

platform driver

receiver

sensor

drivers/media/v4l2-core/

/v4l2 core

videobuf2

/v4l2-async

DRM

alloc()

HARDWARE

pixel data

i2c

i2c messages
Video4Linux2: zero copy image streaming

```
user application
int dmabuf_fd[]

/dev/video0
```

```
drivers/media/v4l2-core/
```

```
v4l2 core    |    videobuf2    |    v4l2-async
```

```
platform driver
```

```
capture_queue
```

```
receiver
```

```
pixel data
```

```
sensor
```

```
i2c messages
```

```
DRM
```

```
KERNEL
```

```
HARDWARE
```

Jacopo Mondi - FOSDEM 2018 Image Capture On Embedded Linux Systems (40/63)
Video4Linux2: zero copy image streaming

Diagram illustrating the architecture of Video4Linux2, showing the interaction between different components such as user application, /dev/video0, VIDIOC_QBUF, v4l2-core, videobuf2, v4l2-sync, DRM, kernel, hardware, receiver, sensor, pixel data, i2c, and dmabuf_fd[].
Video4Linux2: zero copy image streaming

user application
int dmabuf_fd[]

/dev/video0

VIDIOC_QBUF

drivers/media/v4l2-core/

v4l2 core  videobuf2  v4l2-async

platform driver

capture_queue

receiver

pixel data

sensor

i2c

i2c messages

drm

KERNEL

HARDWARE

Jacopo Mondi - FOSDEM 2018

Image Capture On Embedded Linux Systems (42/63)
Video4Linux2: zero copy image streaming

```
/dev/video0

user application
int dmabuf_fd[]

drivers/media/v4l2-core/
  v4l2 core
  videobuf2
  v4l2-async

platform driver
  capture_queue

receiver

pixel data

sensor

i2c

i2c messages

KERNEL

DRM

HARDWARE

Jacopo Mondi - FOSDEM 2018
Image Capture On Embedded Linux Systems (43/63)
Video4Linux2: zero copy image streaming

user application

int dmabuf_fd[]

/dev/video0

drivers/media/v4l2-core/

v4l2 core

videobuf2

v4l2-async

platform driver

capture_queue

receiver

pixel data

sensor

i2c

i2c messages

DRAM

KERNEL

HARDWARE
Video4Linux2: zero copy image streaming

/drive/video0

users application
int dmabuf_fd[]

drivers/media/v4l2-core/
- v4l2 core
- videobuf2
- v4l2-async

platform driver
capture_queue

receiver
IRQ

sensor
pixel data

i2c

i2c messages

KERNEL

HARDWARE

DRM
Video4Linux2: zero copy image streaming

User application
int dmabuf_fd[]
/dev/video0

drivers/media/v4l2-core/
v4l2 core videobuf2 v4l2-async

platform driver
 capture_queue

IRQ
receiver

pixel data
sensor

i2c

i2c messages

DRM
KERNEL
HARDWARE
Video4Linux2: zero copy image streaming

User application
int dmabuf_fd[]

/dev/video0

VIDIOC_QBUF

drivers/media/v4l2-core/

v4l2 core  videobuf2  v4l2-async

platform driver

capture_queue

receiver

pixel data

sensor

i2c

i2c messages

KERNEL

DRM

HARDWARE
Video4Linux2: zero copy image streaming

/dev/video0

VIDIOC_DQBUF

user application
int dmabuf_fd[]

Drivers/media/v4l2-core/

v4l2 core  videobuf2  v4l2-async

platform driver
capture_queue

DRM

KERNEL

HARDWARE

receiver

pixel data

sensor

i2c

i2c messages
Video4Linux2: platform drivers

drivers/media/platform/

- Transfer image data from internal buffers to system memory
- Perform transformations on the received images before presenting them to userspace
- Implement user space API through video device abstraction (or through media controller...)
- Handle IRQs and program receiver interface DMA to actually capture images
Video4Linux2: image sensor drivers

drivers/media/i2c/

- Control the image sensor through I2c transaction
- Respond to platform driver calls to set/get streaming parameters
- Start/stop sensor when platform driver requires data
Video4Linux2: Device tree bindings

Documentation/devicetree/bindings/

- Bindings defines a driver/subsystem ABI
- How drivers expects to be instantiated
- Which and how supply parameters to a driver

Documentation/devicetree/bindings/media/

- Video devices: how do they connect to each other
Platform drivers - Image receivers

.. /bindings/media/video-interfaces.txt

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

    port@0 {
        reg = <0>;
        endpoint {
            "endpoint properties";
            remote-endpoint = <&phandle-0>;
        }
    }
}
Endpoint properties

- Parallel input bus
  
  \[
  \text{vsync-active} = \langle 1 \rangle / \langle 0 \rangle; \\
  \text{hsync-active} = \langle 1 \rangle / \langle 0 \rangle; \\
  \text{data-active} = \langle 1 \rangle / \langle 0 \rangle; \\
  \ldots
  \]

- CSI-2
  
  \[
  \text{data-lanes} = \langle 1 \ 2 \ldots \rangle \\
  \text{clock lanes} = \langle 0 \rangle \\
  \text{link-frequencies} = \langle 210000000 \rangle
  \]
static int platform_probe(struct i2c_client *client) {
    priv = devm_kzalloc(sizeof(*priv));
    devm_ioremap(resources);
    devm_request_irq(irq);
    pm_runtime_enable();
    platform_parse_dt(dev->of_node);
    v4l2_async_notifier_register(priv->notifier);
    return 0;
}
The V4L2 video device APIs are not enough to represent the complexity in modern SoCs

- Dedicated IP blocks on the SoC for image transformation
- media-controller: graph of media entities with sink and source pads
- Each entity can be linked to another entity
- media-controller allows the creation of image manipulation pipelines
Video4Linux2: the media controller
Video4Linux2: the media controller

[Diagram of the media controller system with various components and connections such as receiver, ISP driver, pixel formatter, resizer, converter, system memory, sensor, and i2c messages.]
Video4Linux2: the media controller
Video4Linux2: the media controller

[Diagram of Video4Linux2 media controller with components like receiver driver, isp driver, pixel formatter, resizer, converter, formatter, and system memory connected through arrows and labels for pixel data, sensor, i2c messages, and HARDWARE.]
The media controller drawbacks

- Pipeline creation and management all in userspace
- System boots in a non-usable way
- Video devices vs video sub-devices APIs
- Complexity sometimes not justified for simple use cases
Thank you!
Questions?