

# libcamera

Devendra Tewari

August 9, 2022

libcamera

# Introduction

- ▶ Cameras have gone from highly complex black boxes to highly configurable sensors
- ▶ Complexity has migrated up the stack to user / app space revealing the need for libcamera framework

# Camera Sensor applications

- ▶ Digital photo and video
- ▶ Home
- ▶ Industrial
- ▶ Intelligent agriculture and farming
- ▶ Medical and life sciences
- ▶ Smart Retail

## Camera Sensor Characteristics

- ▶ Resolution
- ▶ Frame rate
- ▶ Exposure time
- ▶ Gain - analogue, digital
- ▶ Color - monochrome, RGB
- ▶ Shutter - rolling, global
- ▶ Focus lenses and filters - Infrared, manual, auto
- ▶ Communication interface - MIPI CSI-2, I2C, SPI, USB

## V4L2/DVB architecture

- ▶ App communicates with camera using device files such as `/dev/video0` typically using `libv4l`
- ▶ Obtains image and video through camera driver such as `bcm2835_v4l2` that depends on camera sensor I2C driver, `videodev`, and other V4L kernel modules
- ▶ `v4l-utils` package provides commands such as `v4l2-ctl` and `v4l2-dbg`

## v4l2-ctl

```
v4l2-ctl --list-devices
```

```
v4l2-ctl --info --device /dev/video0
```

```
v4l2-ctl --list-formats --device /dev/video0
```

```
v4l2-ctl --set-fmt-video=width=640,height=480,pixelformat='Y10 ' '
```

```
v4l2-ctl --stream-mmap --stream-count=1 --stream-to=output.raw --device /dev/video0
```

Raw Y'UV images can be viewed with an app such as yuview

## libcamera architecture

- ▶ App communicates with camera through user space C++ framework library
- ▶ Raw data is obtained from camera driver such as `bcm2835_unicam`
- ▶ Camera sensor I2C driver such as `imx219` is used to send commands
- ▶ Data is processed using extensible image processing algorithms
- ▶ Supports legacy V4L apps and USB cameras
- ▶ Camera Tuning is supported
- ▶ Android Camera HAL3 implementation is available



# Image processing Algorithms

- ▶ Defective pixel correction (DPC)
- ▶ Spatial Denoise (SDN)
- ▶ Automatic White Balance (AWB)
- ▶ Automatic Exposure Control (AEC)
- ▶ Automatic Gain Control (AGC)
- ▶ Automatic Lens Shading Correction (ALSC)

## Camera Modules from Raspberry Pi

- ▶ Camera Module v1 with OmniVision OV5647
- ▶ Camera Module v2 with Sony IMX219
  - ▶ Filter-less Pi Noir version available
- ▶ HQ Camera with Sony IMX477

## Camera Modules from Arducam

- ▶ Camera modules for Raspberry Pi and other SoCs
- ▶ Monochrome Global Shutter with sensors such as OmniVision OV9281
- ▶ 64MP Autofocus 10x Digital Zoom Camera with Sony IMX686 sensor
- ▶ Stereo/Quad-Scopic Cameras

## libcamera-apps from Raspberry Pi

```
LIBCAMERA_LOG_LEVELS=*:DEBUG libcamera-hello --list-cameras
```

```
libcamera-jpeg -o test.jpg -t 2000 --shutter 20000 --gain 1.5
```

```
libcamera-still -r -o test.jpg
```

```
libcamera-still -o test.jpg --post-process-file drc.json
```

```
libcamera-vid -t 10000 -o test.h264 --save-pts timestamps.txt
```

```
mkvmerge -o test.mkv --timecodes 0:timestamps.txt test.h264
```

```
libcamera-vid -t 10000 --codec mjpeg -o test.mjpeg
```

# libcamera C++ API

- ▶ Open camera
- ▶ Configure streams
- ▶ Start camera
- ▶ Stop camera
- ▶ Close camera

## Open camera

Create instance of `libcamera::CameraManager` to acquire an instance of `libcamera::Camera`

```
camera_manager_ = std::make_unique<CameraManager>();  
camera_manager_->start();  
std::string const &cam_id = camera_manager_->cameras()[0]->id();  
camera_ = camera_manager_->get(cam_id);  
camera_->acquire();
```

## Configure streams

```
StreamRoles stream_roles = { StreamRole::VideoRecording };  
// { StreamRole::StillCapture, StreamRole::Raw };  
// { StreamRole::Viewfinder };  
configuration_ = camera_->generateConfiguration(stream_roles);  
configuration_->at(0).pixelFormat = libcamera::formats::YUV420;  
configuration_->at(0).bufferCount = 2;  
configuration_->at(0).size.width = 640;  
configuration_->at(0).size.height = 480;  
configuration_->validate();  
camera_->configure(configuration_.get());
```

## Start camera

For each frame an application wants to capture it must queue a request for it to the camera

```
Stream *stream = configuration_->at(0).stream();
allocator_ = new FrameBufferAllocator(camera_);
allocator_->allocate(stream);
const std::vector<std::unique_ptr<FrameBuffer>> &buffers =
    allocator_->buffers(stream);
controls_.set(controls::ExposureTime, 15000);
controls_.set(controls::AnalogueGain, 2);
camera_->start(&controls_);
controls_.clear();
camera_->requestCompleted.connect(requestComplete);
for (unsigned int i = 0; i < buffers.size(); ++i) {
    std::unique_ptr<Request> request = camera_->createRequest();
    request->addBuffer(stream, buffers[i].get());
    camera_->queueRequest(request.get());
    requests_.push_back(std::move(request));
}
```



requestComplete

```
if (request->status() == Request::RequestCancelled)
    return;
const libcamera::Request::BufferMap &buffers = request->buffers();
Save(buffers);
std::unique_ptr<Request> requestToQueue = camera_->createRequest();
for (auto bufferPair : buffers) {
    FrameBuffer *buffer = bufferPair.second;
    const Stream *stream = bufferPair.first;
    requestToQueue->addBuffer(stream, buffer);
}
camera_->queueRequest(requestToQueue.get());
requests_.push_back(std::move(requestToQueue));
```

## Save YUV420 when no padding in data

```
assert(configuration_>at(0).size.width == configuration_>at(0).stride);
unsigned int length = 0;
for (auto bufferPair : buffers) {
    FrameBuffer *buffer = bufferPair.second;
    const FrameMetadata &metadata = buffer->metadata();
    std::cout << " seq: " << std::setw(6) << std::setfill('0') << metadata
    for (unsigned i = 0; i < buffer->planes().size(); i++)
    {
        length += buffer->planes()[i].length;
    }
    const std::string& filename = GetTimestamp() + ".yuv";
    std::ofstream out(filename, std::ios::out | std::ios::binary);
    void *memory = mmap(NULL, length, PROT_READ | PROT_WRITE, MAP_SHARED, 1
    out.write((char *)memory, length);
    munmap(memory, length);
    out.close();
}
```

## Encoding image and video

- ▶ libcamera provides raw image frames from the sensor that are processed using image processing and control algorithms based on per sensor tuning file
- ▶ libcamera does not provide additional means to encode or display images or videos
- ▶ libcamera-still uses external libraries to encode images such as libjpeg for jpeg
- ▶ libcamera-vid uses hardware H.264 encoder on Raspberry Pi through /dev/video11, apps may use external libraries such as ffmpeg/libav codec for H.264

Stop camera

```
camera_->stop();  
camera_->requestCompleted.disconnect(requestComplete);  
requests_.clear();
```

## Close camera

- ▶ Configure streams, start and stop camera can repeat multiple times
- ▶ The application is now finished with the camera and the resources the camera uses

```
Stream *stream = configuration_->at(0).stream();  
allocator_->free(stream);  
delete allocator_;  
configuration_.reset();  
camera_->release();  
camera_manager_->stop();
```

## Build and run example

Invoke compiler directly

```
c++ example.cpp -o example `pkg-config --cflags --libs libcamera` -std=c++11
```

Use meson

```
meson build
```

```
cd build
```

```
ninja
```

```
$ LIBCAMERA_LOG_LEVELS=:DEBUG ./example
```

Press enter to exit...

```
seq: 000004
```

```
seq: 000005
```

```
seq: 000006
```

## Explore further

- ▶ [libcamera: The Future of Cameras on Linux is Now](#)
- ▶ [libcamera Documentation](#)
- ▶ [libcamera API](#)
- ▶ [Raspberry Pi Camera Guide](#)
- ▶ [Raspberry Pi Camera Algorithm and Tuning Guide](#)