Generic V4L2 Android camera HAL

Guennadi Liakhovetski

(Contact: linux-media@vger.kernel.org)

August 28, 2012

Everyone is special

- Android performs access to various hardware subsystems using Hardware Abstraction Layer (HAL) libraries. E.g., audio, video.
- These HAL libraries are vendor specific.
- They provide certain services to the OS, use available APIs.
- Example: TI camera HAL provides multiple character devices: for preview, video and still image capture, uses OMAP IP blocks to provide simultaneous preview and video capture.
- An Android camera HAL runs in the context of a media server, it implements a generic Android hardware device with probe() and open() methods. When activated by applications, the media server queries HAL's capabilities, configures the camera, supplies call-backs for managing video buffers and delivering video frames.

Don't reinvent the wheel

- Goal of this project is to develop a generic Android camera HAL, based on the already available basis: the V4L2 API.
- Implement functions, already available in V4L2, add missing ones.

Current state

- It works :-)
- Tested with the standard Android Camera app, with other apps the call order might change.
- Preview, still images and video capture work.
- The principal HAL module is implemented in C, whereas most Android interfaces are in C++. To use them simple thin C++ wrappers have been developed.

The gory detail

- Use an sh7372 ARM Cortex A8 (TM) based mackerel system with an on-board YUV VGA camera.
- Because of the fixed and low resolution, tested only all equal frame formats.
- Flexible design: 3 sets of buffers are allocated, when still images are taken, preview is stopped, for video capture the (larger) video resolution is configured, preview frames are obtained by (down-) scaling capture frames.
- Very low FPS due to data processing and copying in software.

Buffer management

- 2 Sets of buffers: camera HAL uses one OS callback to obtain camera buffers - either mmapped on a file descriptor, or allocated from the system RAM; preview buffers: system RAM buffers, used later to compose the image on the display.
- 2 copy operations: (1) from camera output buffers to window buffers and (2) from window buffers to the resulting image.
- On mackerel YUV camera output buffers first have to be converted to an RGB format with output into surface buffers, surfaces from various apps are then composed by the surface-flinger service to produce a framebuffer image.
- With 2 copy / convert operations the system manages 4.3 fps, without the camera copy-convert operation 13+ fps, a V4L2 test application under Linux on mackerel produces 27 fps.
- 2 optimisation possibilities: (1) use a hardware format converter, (2) use a hardware blending / composing unit.

Hardware acceleration

- A VEU mem2mem driver has been written to convert YUV camera output to a 16-bit RGB format.
- A test program from Sylwester has been fixed and extended.
- To submit video buffers from the camera to the converter MMAP is used with the camera and USERPTR with the mem2mem device.
- No standard way to allocate buffers, suitable for VEU output. “For testing” a hack is being implemented to tell Android to skip allocating buffers in its gralloc module. Instead MMAP shall be used with the VEU output (CAPTURE) queue to allocate contiguous buffers in the driver and submit them to Android for composition.
- In the future Android is going to switch to a new ION memory graphic allocator, which is the reason, why implementing a proper gralloc-based solution ATM isn’t worth the effort.

Future directions

- Android is switching to ION for memory pool management
- Linux introduced DMABUF
- V4L2 has to be made configurable: device nodes etc.
- libv4l porting to Android and use with the HAL for data processing and device configuration, including system-specific MC set up.