Advanced Software Example: Improving A Webcam Driver

Chris Jenkins - Genesi USA, Inc

Terminology

- `ov7690`: “Omnivision 7690”, camera model in use on the Efika MX53-based Drónov Slimbook
- `V4L2`: “Video 4 Linux 2”, video capture / output device API and driver framework for the Linux kernel
- `GStreamer`: open source multimedia framework
- `GStreamer plugin`: code interface between GStreamer and kernel interface (e.g. V4L2) for media device
- `Cheese`: Common web-camera booth software that ships with many Linux distributions.
- OSS: Open Source Software

Developing Solutions using Open Source Software

One of the major advantages of using OSS is having access to a wide variety of freely available software, often trying to achieve a solution to the same or similiar problem. Why reinvent the wheel when someone else has designed a workable solution already? However, that does not mean that a given software solution will always “just work.”

- OSS is provided “as-is”. Some software is *par excellence* such as GCC, others not so much
- If the problem the OSS attempts to solve is not exactly the same, or for the same hardware, same OS, same software versions, etc, as what you are trying to accomplish, you will need to make modifications.
- This may require a small group of skilled software engineers, probably using Agile development. Probably not as much overhead / cost as a larger group using Waterfall method building a solution from scratch

What follows is an overview of an involved problem concerning the Drónov Slimbook camera and the different software surrounding it. This will hopefully give an idea of the kind of work and decisions required when adapting OSS to your particular needs

Drónov Slimbook Camera: OV7690

- Ultra low power and low cost
- programmable controls for through registers written through i^2c.
    - frame rate, AEC/AGC, mirror and flip, scaling and windowing, color saturation, hue, gamma, sharpness (edge enhancement), lens correction, defective pixel canceling, and noise canceling
- support for output formats: RAW RGB, RGB565, CCIR656 and YCbCr422 in VGA, scaling CIF and, sub-sampling QVGA and scaling QCIF

Problems with the Drónov Slimbook Camera

- Incomplete driver for `ov7690`: read / write registers didn’t work, register values out of reset had poor edge definition and high chroma noise. No camera modes developed and mechanism for switching between modes broken
    - We are only supporting capture in raw YUV image format, but that still leaves VGA, QVGA, CIF and QCIF
- The Freescale GStreamer plugin for V4L2 devices (such as our camera) had a few bugs as well, and was not handling different camera capture modes for the camera (when implemented)
- Code in Cheese to get capture dimensions from V4L2 GStreamer plugin had a bug in enumerating capture modes

OV7690 Driver

- Old version can be found here:

  https://github.com/genesi/linux-shortbus/blob/6e1f64babbb4293deb49b5a2d47f0199e9b4c348/drivers/media/video/mxc/capture/ov7690.c
- Things to note:
  - Read / writes to registers did not work (fix taken from Gitorious project for similar hardware)
  - Different camera modes not supported, and mode / fps settings not configured
  - Bug in interface between V4L2 and camera driver causing kernel to think camera supported infinite video standards
  - Poor overall image quality (before / after)
    - High chroma noise
    - Poor edge definition

Discussion: High Chroma Noise

- What causes noise?
  - Random electrical fluctuations in the camera, extreme lighting situations
- What types of noise are there, and how do users react to them?
  - Luminance noise and chroma noise. Of these, luminance gives a “grainy” feel to the image that reminds users of old film, and is reasonably tolerated. Chroma (color) noise, however, is greatly disliked
- How can we deal with chroma noise?
  - Simplest solution is to increase exposure. The more exposed the image is, the more each pixel resembles the signal and not the noise.
- What are the consequences of this solution?
  - Slower frame-rate, motion blur, artificial mage brightness
- Changing the register settings for exposure time, even to the highest setting, did not initially make any difference. Why? Hint: Look at the code again.
  - Time-per-frame first has to be increased through device clock - cannot expose an image for longer than frame

Hands-On: Workflow and the benefits of OSS

How are you going to hand test the various combinations of exposure, frame rate, etc? Hopefully not by changing the driver, recompiling, and booting the Slimbook with a new kernel! Too much turn-around time.

Enter `i2c-tools`, a collection of OSS tools containing “a bus probing tool, a chip dumper, register-level access helpers, EEPROM decoding scripts, and more.” The programs are well documented (MAN pages) and easily scriptable.

Let’s try opening Cheese and changing some of the camera driver settings via command line. Download some helper scripts from this git repo and start changing the register settings through the scripts (This requires the OV7690 Datasheet or activity handout)

GStreamer FSL Plugin

- Old version can be found here:

  https://github.com/genesi/gst-fsl-plugins/blob/master/src/misc/v4l_source/src/mfw_gst_v4lsrc.c
- Things to note:
  - Confused frame-rate and time per frame in a few places.

    Notice first example use isn't wrong, but requires more thought overhead. The second example **is** mistaken.
  - Did not support reading multiple camera modes

    More recent code in GStreamer plugins can read camera mode information from V4L2, but in the interest of time we hard-coded our camera modes.
  - Not setting correct capture capabilities for our camera

Discussion: Data Structures

- FSL plugin has easy to make mistake of mixing up units (Hz vs seconds). But the issue isn't just the mix-up: Even if used correctly, the design of the struct for frame-rate was problematic. Why?
  - GST code asks for both frame-rate and time-per-frame, but there was only one data representation (frames per second), so the programmer always has to be doing unit conversions (though in this case only multiplicative inverse)
- How could we solve this?
  - Data representations for both units of measurement
- What are some drawbacks to this solution
  - Duplicating the same data in different formats means you're more likely to make a mistake (boilerplate)
- Current solution:
  - Uses C union types and magic (knowledge of byte layout)

Cheese

- Old version can be found here:

  https://github.com/genesi/cheese
- Things to note:
  - Detection of camera capture dimensions caused infinite loop
  - Update changelog

    Good form for changes to Debian software in user space. Helps keep track of versioning, etc. (Was also done for Gstreamer plugins.)

Cheese Camera Format Detect

- Let us take another look at the Cheese code responsible for detecting different capture dimensions. The bug is actually quite simple - can you see it just by looking at the code?
- What might a solution look like for this?
- The fix currently in use

Summary

- First and foremost, you cannot do anything with your camera if you cannot read and write registers reliably. If your device is not behaving as you expect, it never hurts to check.
- Implement modes as arrays of register settings; test, tune, and read the camera device specification
- Use appropriate userspace tools to test changes to the kernel
    - This lead to the discovery that the V4L2 interface was claiming to accept an infinite number of video standards
    - Also revealed GStreamer plugins could not read the camera mode capabilities from V4L2
- Download source for modern GStreamer FSL plugins, compare with current GStreamer V4L2 device plugins
    - Make a call: not worth the time to overhaul current FSL plugins with newer code, so hard-code mode capabilities.
- Make sure user programs using device behave as expected (Cheese)
    - Discovered Cheese made assumptions about capture sizes that were incorrect
- Result: a vertically integrated software solution, from the device driver, to kernel subsystem, to core multimedia library, ending at the iconic user programs.

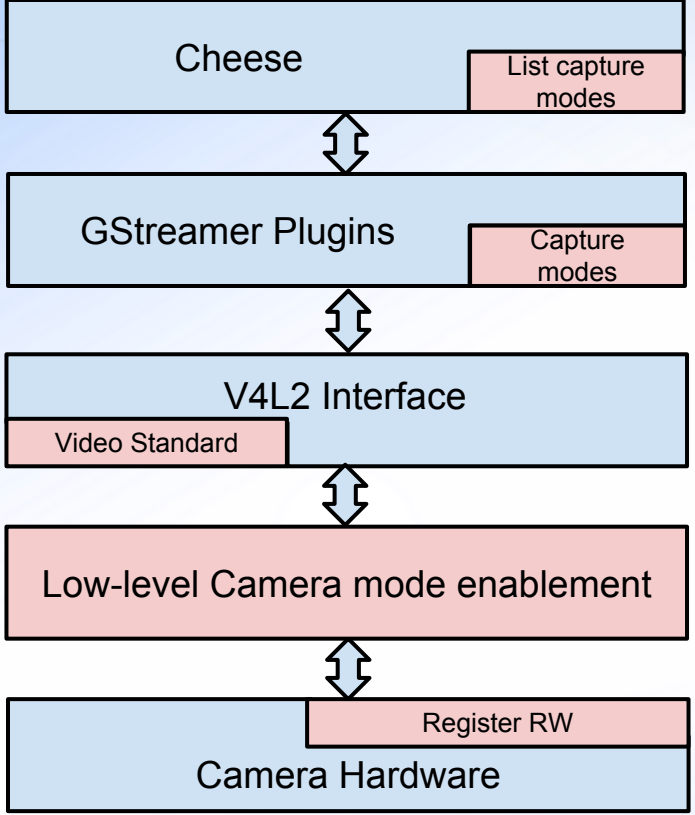

Diagram of software changes discussed

Lessons

- For open source projects, search for solutions that others have developed to the same or a similar problem, and decide whether it makes sense to use their work.
  - Register read / write and initial register settings for modes taken from another project
  - More current GStreamer V4L2 plugins do exist, but implementing this functionality would require major overhaul of FSL code and be very unlikely to “just work”.
  - You have to decide when it is appropriate to use someone else’s code or to write it yourself. Only you can make this judgement, and the extra effort required to reuse code will always vary with the code quality and your project demands.
- Use the right testing tools
  - Your user may only ever use Cheese, but you are likely to miss big errors if you do not use the power tools yourself, `gst-launch` from `gst-tools`, in our case.
- Also, be careful about changing code formatting, especially using an editor that automatically formats for you
  - Style guides are a must. Make them explicit, make sure everyone understands why they are important.

genesi