Bringing display and 3D to the C.H.I.P computer

Maxime Ripard
maxime@bootlin.com
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 **training**, Linux driver development training and Android system development training, with materials freely available under a Creative Commons license.
  - [http://bootlin.com](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 (4.3, soon to be 4.4)
Development effort

- A significant part of the work already done
- But key features for a desktop-like application were missing
  - Audio
  - NAND support
  - Display
- Plus board specific developments
  - Wifi regulators
  - DIP
How to display things in Linux?
Doing display things

- Different solutions, provided by different subsystems:
  - FBDEV: Framebuffer Device
  - DRM/KMS: Direct Rendering Manager / Kernel Mode Setting
  - More exotic ones: V4L2, auxdisplay

- How to choose one: it depends on your needs
  - Each subsystem provides its own set of features
  - Different levels of complexity
  - Different levels of activity
Which one to choose?

- **DRM**
  - Actively maintained
  - Provides fine grained control on the display pipeline
  - Widely used by user-space graphic stacks
  - Provides a full set of advanced features

- **FBDEV**
  - Deprecated?
  - Does not provide all the features found in the modern display controllers (overlays, sprites, hw cursor, ...)
Bringing display and 3D to the C.H.I.P computer

DRM/KMS
DRM stands for Direct Rendering Manager and was introduced to deal with graphic cards embedding GPUs.

KMS stands for Kernel Mode Setting and is a sub-part of the DRM API.

Though rendering and mode setting are now split in two different APIs (accessible through `/dev/dri/renderX` and `/dev/dri/controlDX`).

KMS provide a way to configure the display pipeline of a graphic card (or an embedded system).

KMS is what we’re interested in when looking for an FBDEV alternative.
KMS components

- **Planes**
  - Image source
  - Associated with one (or more!) framebuffers
  - Holds a resized version of that framebuffer

- **CRTC**s
  - Take the planes, and does the composition
  - Contains the display mode and parameters

- **Encoders**
  - Take the raw data from the CRTC and convert it to a particular format

- **Connectors**
  - Outputs the encoded data to an external display
  - Handles hotplug events
  - Reads EDIDs
Allwinner display pipeline

Display Engine → Timing Controller → TV Encoder

Composite → LCD Panel
DRM vs SoC pipeline

Planes
- Display Engine

CRTC
- Timing Controller

Encoder
- TV Encoder

Connectors
- Composite
- LCD Connector
Bringing display and 3D to the C.H.I.P computer

GPU integration
The GPU found in most Allwinner SoCs is the Mali-400 from ARM (with a variable number of cores)

There are two options to support that GPU:

- **Lima**
  - Reversed engineered proof-of-concept
  - Triggered the reverse engineering effort of the GPUs (freedreno, etnaviv, etc.)
  - Development (closed to?) stopped two years ago

- **ARM-Provided support**
  - Featureful
  - Two parts: GPL kernel driver and proprietary OpenGL ES implementation
DRM Stack: GPU

- X11 plugin
- libdrm
- KMS
- DRM driver
- Display Engine
- CMA
- DMA-buf
- RAM
- GPU
- GPU driver
- /dev/mali
- PRIME
- OpenGL ES
Everything is provided by ARM on their website (if you’re lucky)

On the userspace side, you just need to put the library they provided on your system

On the driver side, you need to create a platform glue that will deal with:

- Memory mapping
- Interrupts
- Clocks
- Reset lines
- Power Domains
- Basically everything needed for the GPU to operate properly on your SoC
X11 integration

- We need a DDX (Device Dependent X) driver
- `xf86-video-modesetting` is working on top of KMS and MESA (Gallium3D)
- ARM developed `xf86-video-armsoc` for SoC using a 3rd party GPU (Mali, PowerVR, Vivante, etc.)
- Relies on KMS for the display configuration, driver-specific ioctl for buffer allocations and vendor-provided OpenGL ES implementation
- Just have to write a small glue to use your driver allocator, and give some hints to X about what your hardware support (hw cursor, vblank, etc.)
Questions? Suggestions? Comments?

Maxime Ripard
maxime@bootlin.com

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