Remarks on 2D Hardware Acceleration Support

Paul Kocialkowski
paulk@sys-base.io

[sys-base]

XDC 2025
2025-09-30

2D Hardware Accelerators

Characteristics:

- Implement **2D-only raster** operations
  - Generally: scale, rotate, blit, crop, format, csc and more
- Typically **fixed-function** hardware
  - No shader or instructions in the pipeline
  - Static internal pipeline, bypass and mux
  - Programmed with direct registers or command stream
  - Low power, low latency, high performance
- Used for both (display) **compositing** and **multimedia** pipelines
- **Standalone** (memory to memory) or **dedicated** (front-end) integration

History:

- Very common in microcomputers starting from the late 70s
- Replaced by full GPUs for desktop/gaming starting from the late 90s
- Came back for dedicated use-cases (e.g. embedded, low power) in the 2000s
- Still **relevant, useful and used** today

2D Hardware Accelerators

Implementations:

- **Available designs:** e.g. (Verisilicon) Vivante GC520/GC520L
- **In-house designs:** e.g. Samsung FIMG2D, Rockchip RGA, Allwinner G2D
- Part of a bigger **GPU unit**: e.g. Imagination SGX (PVR2D)

RK3588 RGA3 Unit Block Diagram

Past Support Situation

Decay of 2D support:

- **Fbdev** acceleration via custom ioctls
  - Driver-specific, not generic
- **XFree86 XAA** and **X.org EXA** for DDX
  - Sometimes using direct userspace hardware access
  - Difficult and not very satisfying
- Everything **moved to GL** around 2010
  - DDX became generic with modesetting and Glamor
  - All 2D acceleration was dropped
  - Perhaps relevant for desktop but very bad for embedded
  - Some 2D-accelerated DDX remained (e.g. `xf86-video-armsoc`)
- **Wayland** replaced X11
  - Initial support with GL, sometimes pixman
  - Nothing to support 2D hardware

Current Support Situation

Good things happened too:

- **DRM Render** can support 2D acceleration
  - Driver-specific ioctls, not very elegant for fixed-function
  - Existing drivers: `exynos`, `etnaviv`, (`microchip/gfx2d`)
- **DRM KMS planes**
  - Only for front-ends (not memory to memory)
  - Almost satisfying (flat properties)
- **V4L2 M2M** framework
  - Limited to single in/out buffers and format-based operations
  - Good enough for many multimedia workflows (no blitting)
  - Existing drivers: `s5p-g2d`, `rga`
- **Wayland compositors** can support 2D acceleration
  - Now using KMS planes for front-end compositing (limited)
  - Missing a generic library for general compositing
  - NXP has a Weston renderer using their stack

Why So Difficult?

Explicit objections (from the past):

- There is no defined **standard API**
- 2D is **really hard** to support

Bottomline:

- We are looking like **fools**!
  - 2D acceleration is a 30-year-old feature...
  - Strong demand and valid use-cases
  - Vendors are rolling their own proprietary stacks
- We just need a **reasonable API**, not a standard
- The main difficulty is **internal pipeline variability**:
  - Various internal blocks (features)
  - Data flow configuration (bypass, mux)
  - Order between blocks matters

Proposal: Explicit Topology

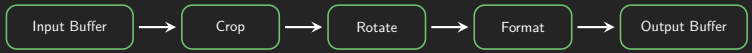
General ideas:

- No **generic flat API** can easily accomodate hardware variability
- Workloads can be described as a **topology of configured linked operations**
- Hardware can be described as a **topology of blocks, possible links and properties**
  - Similar to the Media Controller API for complex pipelines
- **Hardware topology** is configured to match **workload topology**
  - Blocks bypass and muxing when possible
- A **generic library and uAPI** can work without driver-specific bits

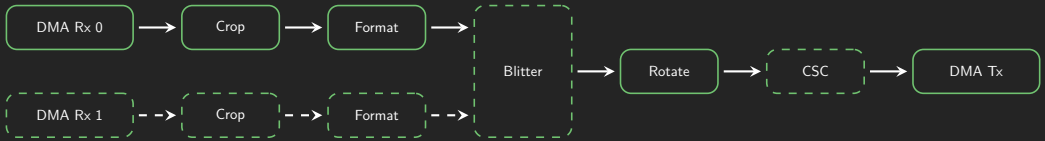
Components:

- Kernel-side: DRM G2D core, drivers and uAPI
- Userspace: libg2d, minimal API

Proposal: Explicit Topology Example



Example requested workload topology



Example corresponding hardware topology

Proposal: DRM G2D

General:

- **Generic uAPI and core** for fixed-function hardware
- Userspace retreives **topology from driver**
- Userspace submits **jobs (state)** to a queue, can **validate** first
- **State** is a list of **properties** for each block
  - Base function properties (generic)
  - GEM memory handles, fences, etc for DMA
  - Links between blocks, bypass
- **Sync** with fences or explicit ioctl

Driver role:

- Register **static hardware blocks** description
  - One for each base function and DMA rx/tx
- **Validate** requested state
- **Configure each block** as requested in state
- Pretty dumb and simple

Proposal: libg2d

General:

- **Generic API** for applications, topology-based
  - Add configured function blocks and link them
  - Standalone (not derived from bigger API) and simple
- Various possible **backends**:
  - Generic DRM G2D
  - Specific DRM Render
  - Generic V4L2 M2M
  - Generic OpenGL/Vulkan
- **Checking and validation**, may fail due to hardware

Topology solving:

- **Accomodate** user-provided topology using hardware topology
- **Generic solver** is doable (but not easy)
- **Simplified static** approaches can exist

Discussion

Takeaway:

- Proper 2D hardware acceleration support is **possible**
- This is a **rough proposal outline**, still lots of details to discuss
- Please **reach out** if interested

Thanks for listening!