



Bringing Linux DRM Display Panel support in the modern age

XDC 2026

Neil Armstrong - Rui Silva

Problem



Displays are not described.

The DRM Framework's Blind Spot

Physical displays lack proper representation in the DRM (Direct Rendering Manager) framework. Our current model focuses entirely on the *transmission*, not the *destination*.

What we DO describe:

- Connector capabilities.
- EDID/DisplayID blob
- The link to the panel (I2C, DP AUX, DSI commands).

What we MISS:

- The actual physical properties, timings, and complex state machine of the panel itself.

Displays are not described.

Panels Are Not Bridges

To make embedded panels work today, we model them as DRM "bridges" attaching to a connector.

The architectural mismatch: A bridge is designed to convert or pass signals along, not act as the final physical endpoint.

Why this is a problem: By treating the panel merely as a bridge, we abstract away modern display complexities (like advanced power sequencing and physical hardware quirks) behind an inaccurate software model.

Missing display control.

Who Controls the Panel?

- **The Disconnect:** We don't directly control a display; we orchestrate its dependencies.
- **What we actually touch:**
 1. **Video Data:** Driving the transmission link.
 2. **Sideband Comms:** Sending commands via I2C, DP AUX, or DSI.
 3. **Support ICs:** Commanding discrete backlight controllers and regulators.
- **The Architectural Flaw:** These controls do not live on the display device itself. The panel is a passive sink in our framework, while the actual control mechanisms are scattered across completely different kernel subsystems (Regulator, Backlight, I2C/SPI buses).
- **The Result:** Panel drivers become complex state machines just to synchronize external chips and buses, rather than modeling the display hardware itself.

Missing display characteristics.

Userspace is Flying Blind

- **The Current Hacks:** To get basic display characteristics, userspace relies on fragile methods:
 - Parsing EDID (often missing on embedded panels).
 - Matching hardcoded panel names and basic modes.
 - Manually parsing DisplayID or sending undocumented commands over DP AUX.
- **The Missing Modern Features:** Despite these workarounds, the kernel UAPI fails to share critical modern display metrics:
 - Colorimetry.
 - Backlight mapping curves.
 - HDR characteristics and physical limits.
- **The Core Problem:** The kernel initializes the hardware but forces userspace to reverse-engineer its capabilities just to render a proper image.

Vendors are absent

Nowhere to be seen

Display & Panel Vendors are completely absent in Linux

- they don't share any documentation
- we don't get full display specifications
 - actual required timings (!)
 - vendor protocols
 - physical characteristics
 - DDIC documentation...

This adds to the fact that display protocols are not public and hard to implement (HDMI, DP, DSI, ...)

Embedded Displays

Outdated API

Embedded display panels are described as panel with a very old non-atomic API

Lacks all the modern-ish features:

- Atomic state in order to get the mode
- Standby and advanced power states
- Advanced color management
- Dynamic rate switching
- Command mode self-refresh

Unable to match vendor kernels

Welcome to the Android closed-open-source

Commercial Android phones have super powerful displays supporting awesome features like:

- self refresh
- dynamic rate switching,
- advanced low power mode with low latency wakeup...

With the lack of documentation on the protocols, DDICs & Display engines we just can't match the features

Solutions

linaro linaro linaro

Get proper documentation

The Impossible task

The Goal: Transition from reverse-engineering to official hardware documentation.

The Hurdle: Panel initialization sequences and physical parameters are often treated as strict trade secrets, gated behind restrictive NDAs.

The Outreach: Initiate direct dialogue with major panel vendors (e.g., Samsung, BOE, LG).

The Facilitators: Leverage trusted umbrella entities (Linux Foundation, X.Org Foundation) as legal intermediaries to negotiate terms that satisfy both vendor IP concerns and GPL requirements.

Represent the display

Every type of display

Let's represent the display:

- External or Internal
- Classic or Specific (rounded, transparent, XR, ...)
- Provide add all the characteristics we can probe from the kernel
- Add direct API to control to the display:
 - Backlight management should be a display control
 - Firmware update control should have a dedicated channel
 - Any controllable setting modern display exposes...
- Associated the controls (touch, buttons, privacy, ...)
 - We should stop trying to figure out which device is associated with witch display

Modernize the Panel API

Enter the Atomic Mode Setting world

- Expose the DDIC as a Bridge (well, it's a bridge to the panel !)
- Expose the physical panel as a device
- Add brightness control with the real backlight mapping curve
 - As a panel property
- Allow supporting mode DDIC features
 - Firmware update ?
 - Self-refresh when possible
 - Dynamic rate switching if Display Engine supports it
 - Advanced Colorimetry, HDR....



linaro.org

Thank you

Time for a short Q&A, we can discuss in the hallway for more questions !