



Bringing Linux DRM Display Panel support in the modern age

LPC Kernel Summit 2026

Neil Armstrong

The Problem



The Core Problem: Displays Are Not Bridges

- The Mismatch

To make embedded panels work today, we model them as DRM "bridges" attaching to a connector. A bridge converts or passes signals; it is not the final physical endpoint.

- The Blind Spot

Our current model focuses entirely on the transmission, not the destination. We describe connector capabilities and the link (I2C, DSI), but we completely miss the physical properties, timings, and complex state machine of the panel itself.

- The Result

We abstract away modern display complexities (power sequencing, hardware quirks) behind an inaccurate software model.

Control & Subsystem Fragmentation

- The Disconnect:
We don't directly control a display; we orchestrate its dependencies.
- What we actually touch:
Video data, sideband comms (I2C, DP AUX), and support ICs (Regulator, Backlight subsystems).
- The Architectural Flaw:
The panel is a passive sink in our framework. Panel drivers have become a set of undocumented register writes to blindly initialize a mode, without taking in account any characteristics of the DDIC or the display hardware itself.

Userspace is Flying Blind

- The Current Hacks:
Userspace relies on fragile methods to get display characteristics (parsing missing EDIDs, matching hardcoded panel names, or sending undocumented DP AUX commands).
- The kernel UAPI fails to share:
 - True display Colorimetry and HDR characteristics (physical limits).
 - Backlight mapping curves.
- The Core Problem:
The kernel initializes the hardware but forces userspace to reverse-engineer its capabilities just to render a proper image.
Not using a compositor makes it impossible to properly display on some hardware.

The Expanding Definition of a "Display"

- The API isn't just failing mobile panels; it's failing new form factors.
- XR Displays:
High-resolution, ultra-low latency, complex custom timings, and lens distortion parameters.
- DP Tiled Screens:
Single logical displays driven by multiple physical links that need perfect synchronization.
- Complex Topologies:
Display walls and multi-screen devices where userspace needs to physically identify where connectors/displays are located on the machine (spatial mapping).

Missing Modern Features in the API

- Embedded panels still rely on a very old, non-atomic API.
- What we need for modern UX:
 - Atomic state support to fetch and transition modes cleanly.
 - Advanced standby and low-latency wakeup power states.
 - Universal Self-Refresh: Not just for eDP and command-mode DSI, but standardizing self-refresh across all kinds of displays.
 - Partial Content Redraw: Support for damage tracking/PSR2 to only redraw parts of the screen that changed, saving massive amounts of power.
 - Dynamic rate switching.

Bridging the Vendor Gap

- The Android Disconnect:
Commercial phones have amazing features (advanced low power, self-refresh), but lack of documentation on DDICs and closed-source vendor trees means mainline Linux can't match them.
- The Maintenance Burden:
Upstreaming panel drivers is painfully slow even if time-to-review is short and merge time is fast. Panel init sequences are hidden behind NDAs.
- To help OEMs and vendors upstream faster without exposing IP in C code, we need new approaches.
 - A BPF panel driver patchset was recently submitted. Could this be the bridge that allows vendors to supply complex init sequences safely and quickly?

The Target Architecture

- First-Class Display Entities:
Expose the DDIC as a bridge, but expose the physical panel as a real device.
Same for DisplayPort/HDMI/VGA monitors.
- Unified Display Controls:
Backlight management, firmware updates, and touch/privacy controls must be explicitly associated with the display entity. Stop guessing which input device belongs to which screen.
- Advances properties
Brightness control with real mapping curves as panel properties.
Advanced colorimetry properties, physical position, XR characteristics, ...

The Discussion



Remodeling the Subsystem

- Should we add a DISPLAY DRM object or a DISPLAY blob property to the connector with all properties encoded in a well-known binary format ?
- Shall we start separating now the DDIC into a bridge from the physical Panel in the DRM tree before properly defining the Panel as a Display object ?
- Should backlight, touch, and physical buttons be folded under a unified "Display Device" representation in sysfs/DRM, or linked via device-tree/fwnode properties?
- How do we export spatial/physical location data for display walls and tiled DP displays to userspace?

Modern UAPI & Power

- How do we design a generic UAPI for Partial Content Redraw (PSR2/damage tracking) and universal Self-Refresh that works across DSI, DP, and internal panels?
- Moving to an Atomic Panel API: What are the immediate blockers? Shall we try to rework the entire API instead of hacking the actual one ? We could add helpers to ease transition to the new API.

The BPF Proposal

- BPF for Panel Drivers: Is this acceptable to the DRM maintainers?
- Won't it add maintenance burden for limited advantage ?
 - Won't it distract use from having documented C code instead of register tables in BFP ?
- Most important, will it simplify panel initialization upstreaming from vendors ?
- Does moving initialization sequences to BPF actually solve the time-to-distribution time issue for vendors, or does it just move C driver built with the kernel to userspace blobs?

Vendor & Protocol spec

- How can we leverage foundations (X.Org, LF) to negotiate standard documentation practices with panel vendors (Samsung, BOE, LG)?
- [X.Org](#) can already provide some standard protocol specifications to their members, but what about the MIPI protocols, is there a way to gain access to those ?



linaro.org

Thank you