THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Embedded Linux Conference

When 10,000 Screens Go Dark

Engineering Resilient Linux Drivers for Manufacturing Reality

Subhajit Ghosh

Founder, Tweaklogic

Embedded Linux Professional
Linux Kernel Contributor
Edge AI & Driver Development

Contents

- Background
- Production lines stalled
- Root cause: Display Driver IC substitution
- Why IQC did not catch it
- How LCD vendor selection and bring-up works
- Managing hardware variants in manufacturing
- Approach 1: Separate panel drivers per variant
- Approach 2: Runtime display driver IC detection
- How the solution works
- Outcome
- Possible improvements

Background

Function

- Linux and low-level software developer in Devices team comprising of RF, Electronics, Mechanical, Production and Hardware engineers

Manufacturing

- B2B device manufacturing runs are triggered based on customer orders
- Components are procured and stocked based on future demand projections
- Manufacturing runs planned in parallel with order finalization to reduce lead time

Setup

- A new LCD vendor was selected for a fresh product line
- First several production runs went smoothly for over a year

Production lines stalled

What happened?

- None of the LCDs on one device variant initialized after firmware injection at the manufacturing facility

What was found?

- A new batch of LCDs with approximately the same part number had been installed in all affected devices
- LCD vendor had sent an email without a formal ECN, stating internal components would be slightly varied
- Vendor assured electrical and mechanical design remained unchanged
- Vendor validated new panels using their existing test box and declared compatibility

Root cause: Display Driver IC substitution

Cause:

- LCD vendor changed the Display Driver IC from HX8394F to ILI9881D
- Reason: no supply of variant HX8394F and no alternative source available to them

Why it broke everything:

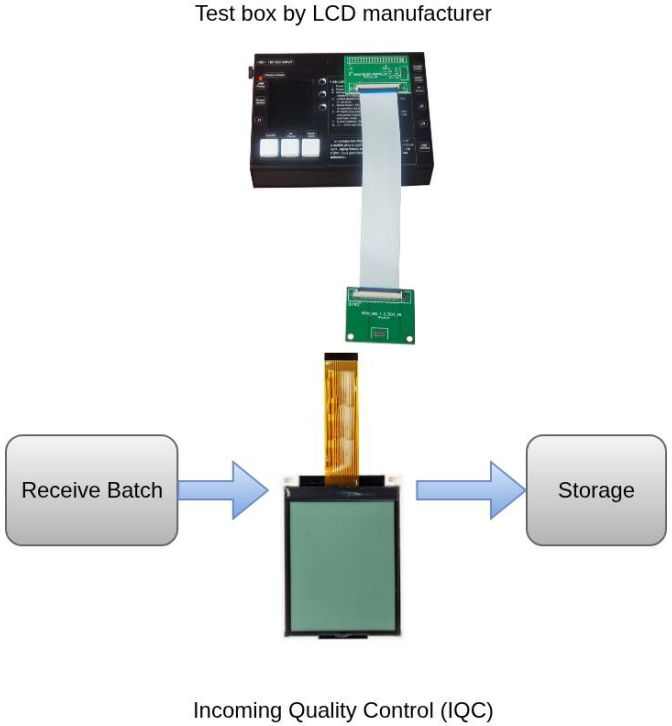
- Display Driver IC initialization sequences differ between HX8394F and ILI9881D
- Linux driver had a hardcoded init sequence written for HX8394F only
- ILI9881D received wrong commands at boot and failed to initialize the panel

No ECN = No Warning

- An Engineering Change Notice (ECN) was never formally issued
- Only an informal email notification was sent, which did not trigger any driver review

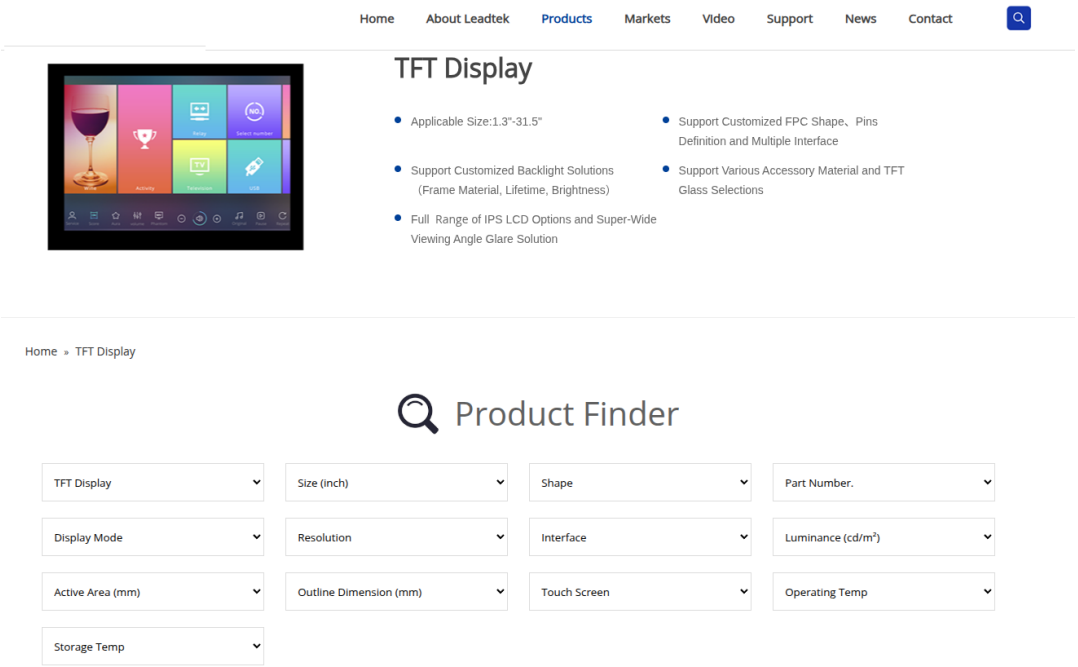
Why IQC did not catch it

- The LCD vendor and PCBA facility are in the same geographical location
- Vendor had silently updated their LCD test box to support the new Display Driver IC
- New panels passed IQC at the factory without raising any flags
- IQC validated the panel electrically and mechanically using the vendor's updated test box
- The test box passed panels that the device firmware could not drive



How LCD vendor selection and bring-up works

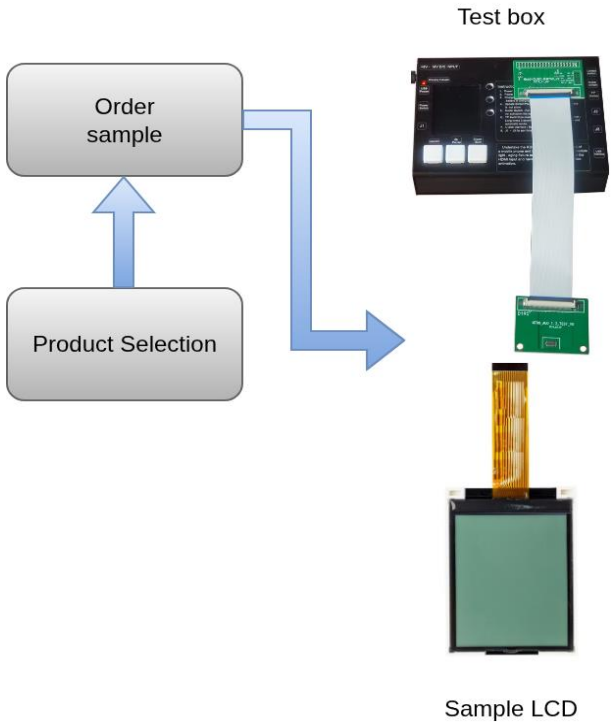
- You don’t get to choose the display driver IC



How LCD vendor selection and bring-up works

Vendor selection

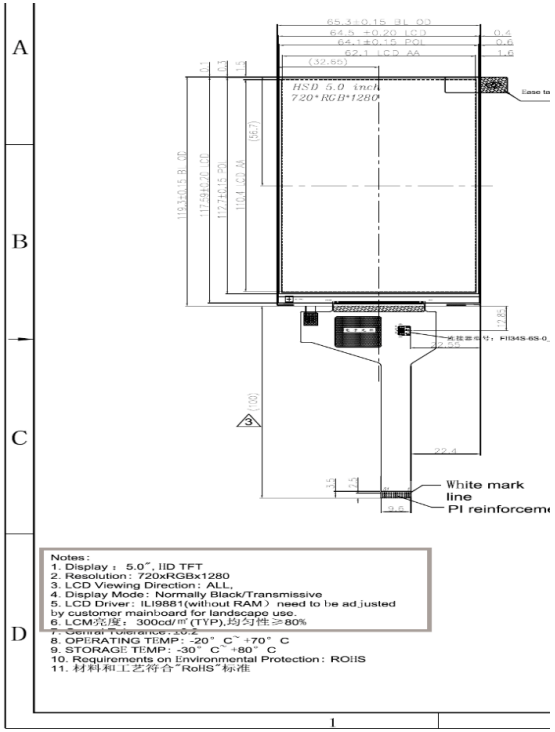
- Most LCD vendors do not offer a choice of Display Driver IC - you select based on interface, resolution, refresh rate, color depth, and lane count
- Once a panel model is selected, samples are ordered and the vendor ships a test box for validation



How LCD vendor selection and bring-up works

Electrical and mechanical fit

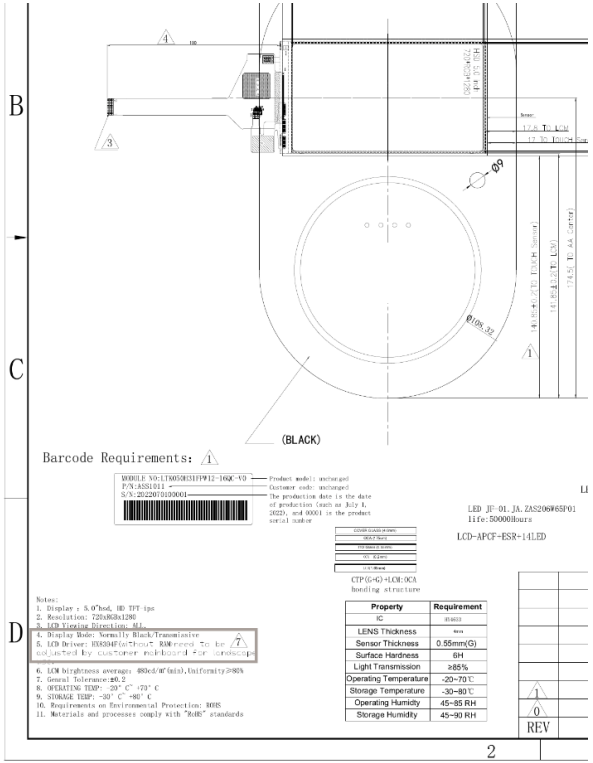
- Choice of Flexible Printed Circuit (FPC) connector, termination, length, pin-count and pitch
- Placement of Touch-screen IC, On or Off FPC.
- Backlight supply through FPC or separate cables



How LCD vendor selection and bring-up works

Product integration

- LCD vendor in this case had a broad product range and offered additional capabilities such as:
  - Bonding the panel assembly to the device glass sub-assembly
  - Supporting both plastic and metal back panels. Metal interferes with NFC designs
  - Quick turnaround times for samples and production



How LCD vendor selection and bring-up works

Software bring-up

- If a Linux kernel driver already exists for the Display Driver IC - bring-up is straightforward
- If not - roll up your sleeves: write a new MIPI DSI panel driver from scratch using vendor Application Notes and Technical Reference Manuals
- Vendor-specific initialization sequences are written as MIPI DSI command writes to Display Driver IC registers
- Software engineers may or may not get a say in panel selection - hardware teams often finalize the choice before firmware engineers are involved

Managing hardware variants in manufacturing

EOL component handling

- End-of-life and mid-lifecycle part substitutions are common in electronics - NAND flash, DDR chips, and display driver ICs are frequent examples

Existing approach in this product range

- A U-Boot menu system loaded different Device Tree Overlays at boot time
- Allowed a single firmware release to support multiple device variants
- Operator selects the appropriate overlay for the hardware present

Two approaches considered for this issue

- Approach 1: Write separate MIPI DSI panel drivers for each Display Driver IC variant
- Approach 2: Detect the Display Driver IC at runtime and load the appropriate init sequence within a single driver

Approach 1: Separate panel drivers per variant

Cleanest way

- Write two separate MIPI DSI panel drivers with corresponding Device Tree bindings and overlays - one per Display Driver IC variant
- Extend the existing U-Boot menu with additional entries to select the correct overlay at boot

This approach requires

- Updated production test plan documents
- Formal ECN issued to the factory
- A new firmware release with both drivers
- Considerably more engineering effort across software, manufacturing, and documentation

Verdict

- Clean and maintainable long-term but heavy overhead given the urgency of stalled production lines
- More work for me!

Approach 2: Runtime display driver IC detection

Key insight

- Even with the new Display Driver IC installed, MIPI DCS commands could still be sent and valid responses received
- A Manufacturer ID query over MIPI DSI could identify the Display Driver IC at runtime

DCS

- Display Command Set is a set of commands for the MIPI DSI interface
- Display driver IC manufacturers have implemented the MIPI standard and vendor-specific register interfaces

HX8394-F

720RGBx1280dots, TFT Mobile Single Chip Driver

Himax

Temporary DATA SHEET V01

5.18.51 Read ID1 (DAh)

DAH	RDID1 (Read ID1)									
	DCX	D7	D6	D5	D4	D3	D2	D1	D0	HEX
Command	0	1	1	0	1	1	0	1	0	DA
1st parameter	1	module's manufacturer[7:0]								xx
Description	This read byte identifies the LCD module's manufacturer. It is specified by display supplier and for xx is defined as xxHEX.									
Restriction	-									

ILITEK

I Love Innovation

a-Si TFT LCD Single Chip Driver
720(RGB) x 1520 Resolution and 16.7M-color

ILI9881D-03

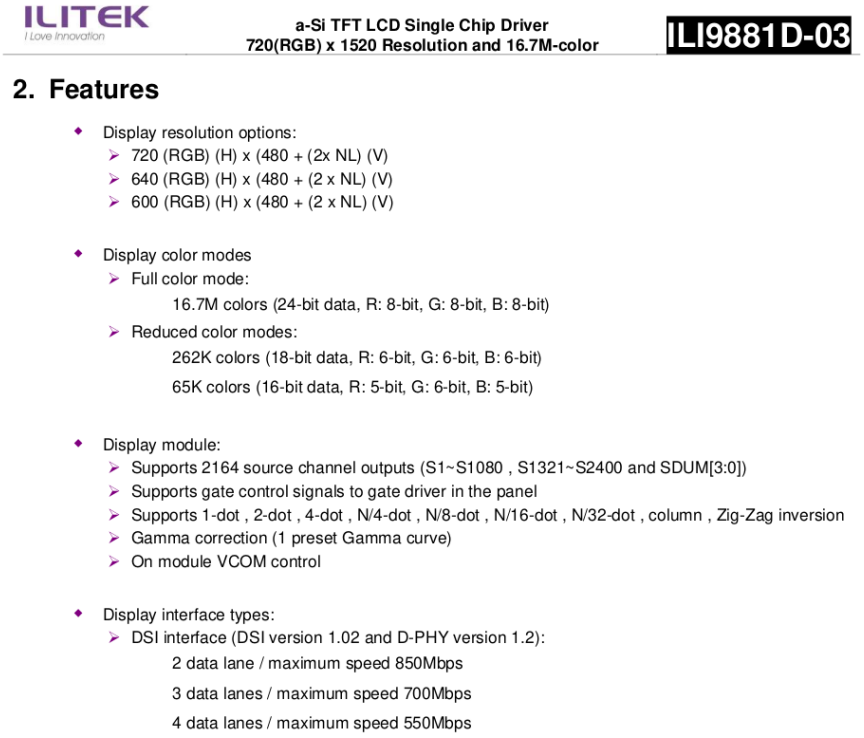
5.3.56. **Read ID1 (DAh)**

Command Page			Page 0								
Address	Parameter	W/R	D7	D6	D5	D4	D3	D2	D1	D0	Default
DAh	1st	R	ID1[7:0]								00h
Description			Dah: RDID1 (Read ID1). This read byte identifies the display module's manufacturer. The ID1[7:0] is programmed by the OTP function.								

Approach 2: Runtime display driver IC detection

Why this works

- Display Driver ICs are highly configurable, supporting multiple resolutions, lane counts, and hundreds of register-level settings
- LCD manufacturers use the same Display Driver IC across multiple panel form factors by loading different initialization configurations
- All configurations are delivered via MIPI DSI command writes to registers in the vendor Application Notes or Technical Reference Manual



ILITEK
I Love Innovation

a-Si TFT LCD Single Chip Driver
720(RGB) x 1520 Resolution and 16.7M-color

ILI9881D-03

2. Features

- Display resolution options:
  - 720 (RGB) (H) x (480 + (2x NL) (V)
  - 640 (RGB) (H) x (480 + (2 x NL) (V)
  - 600 (RGB) (H) x (480 + (2 x NL) (V)
- Display color modes
  - Full color mode:
    16.7M colors (24-bit data, R: 8-bit, G: 8-bit, B: 8-bit)
  - Reduced color modes:
    262K colors (18-bit data, R: 6-bit, G: 6-bit, B: 6-bit)
    65K colors (16-bit data, R: 5-bit, G: 6-bit, B: 5-bit)
- Display module:
  - Supports 2164 source channel outputs (S1~S1080 , S1321~S2400 and SDUM[3:0])
  - Supports gate control signals to gate driver in the panel
  - Supports 1-dot , 2-dot , 4-dot , N/4-dot , N/8-dot , N/16-dot , N/32-dot , column , Zig-Zag inversion
  - Gamma correction (1 preset Gamma curve)
  - On module VCOM control
- Display interface types:
  - DSI interface (DSI version 1.02 and D-PHY version 1.2):
    2 data lane / maximum speed 850Mbps
    3 data lanes / maximum speed 700Mbps
    4 data lanes / maximum speed 550Mbps

How the solution works

Implementation steps

- Issue a Chip ID / Manufacturer ID query via MIPI DCS read
- Based on the returned ID, select and load the appropriate vendor-specific initialization sequence for that Display Driver IC
- Both Display Driver IC variants HX8394F and ILI9881D are handled transparently within the same firmware release

```
static int ltk05031_is_hx8394f(struct ltk05031 *ctx)
{
    u8 id[3] = {0};

    mipi_dsi_dcs_read(ctx->dsi, 0xDA, &id[0], 1);
    mipi_dsi_dcs_read(ctx->dsi, 0xDB, &id[1], 1);
    mipi_dsi_dcs_read(ctx->dsi, 0xDC, &id[2], 1);

    if ((id[0] == 0x83) && (id[1] == 0x94) && id[2] == 0xF)
        return 1;

    return 0;
}
```

```
static int ltk05031_panel_detect_configure(struct ltk05031 *ctx)
{
    ltk05031_hard_reset(ctx);

    if (ltk05031_is_ili9881d(ctx)) {
        dev_info(&ctx->dsi->dev, "Chip:ILI9881D\n");
        ctx->panel_desc = &ltk050h3123w_data;
        ltk05031_hard_reset(ctx);
    } else if (ltk05031_is_hx8394f(ctx)) {
        dev_info(&ctx->dsi->dev, "Chip:HX8394F\n");
        ctx->panel_desc = &ltk050h31fpw12_data;
        ltk05031_hard_reset(ctx);
    } else {
```

How the solution works

Implementation steps

- Chip specific init sequence and data are mapped at panel prepare stage
- Not the cleanest implementation on the planet but saves a lot of effort
- Since the LCD vendor promised electrical and mechanical compatibility, same regulator and reset pin mappings & sequence can be used

```
static const struct ltk05031_desc ltk050h3123w_data = {
    .mode = &ltk050h3123w_mode,
    .init = ltk050h3123w_init_sequence,
};
```

```
static const struct ltk05031_desc ltk050h31fpw12_data = {
    .mode = &ltk050h31fpw12_mode,
    .init = ltk050h31fpw12_init_sequence,
};
```

```
static int ltk05031_prepare(struct drm_panel *panel)
{
    struct ltk05031 *ctx = panel_to_ltk05031(panel);
    int ret;

    /* Power the panel */
    ret = regulator_enable(ctx->power);
    if (ret)
        return ret;
    msleep(5);

    ret = ltk05031_panel_detect_configure(ctx);
    if (ret)
```

How the solution works

What did not change

- No changes to the U-Boot menu or Device Tree Overlays
- No additional factory process steps or production test plan updates
- No formal ECN required - single unified firmware release deployed
- Production lines were unblocked without requiring a hardware split or multiple firmware images
- Review process and implementation confined to a single driver source
- It can address mixed fleet of devices in the field without worrying about compatibility issues

Outcome

Quick turnaround

- A single unified firmware release shipped that transparently supports both LCD variants
- No changes required to the manufacturing process, factory test plans, or U-Boot configuration
- Production lines unblocked rapidly without a formal hardware split

Broader takeaways

- Supply chain substitutions are a manufacturing reality - firmware must be resilient to component-level changes
- IQC processes must be aligned with production firmware, not just vendor test boxes
- Component changes without a formal ECN are a serious process gap that needs to be closed with vendors
- MIPI DCS read capability is a powerful tool for runtime hardware identification in Linux panel drivers

Possible improvements

Separate display driver IC configuration from driver code

- Like how Wi-Fi dongles load firmware blobs from /lib/firmware
- However, /lib/firmware is not suitable as an early splash screen may be required before the root filesystem is mounted
- Better approach is to store configuration in Device Tree, making it accessible to both U-Boot and the Linux kernel during early boot

```
/*
 * Init sequence was supplied by the panel vendor without much
 * documentation.
 */
dsi_dcs_write_seq(dsi, 0xB9, 0xFF, 0x83, 0x94);
dsi_dcs_write_seq(dsi, 0xB1, 0x50, 0x15, 0x75, 0x09, 0x32, 0x44, 0x71, 0x31, 0x55, 0x2F);
dsi_dcs_write_seq(dsi, 0xBA, 0x61, 0x03, 0x68, 0x6B, 0xB2, 0xC0);
```

```
static const struct ili9881d_instr ili9881d_init[] = {

    ILI9881D_SWITCH_PAGE_INSTR(3),
    ILI9881D_COMMAND_INSTR(0x01,0x00),
    ILI9881D_COMMAND_INSTR(0x02,0x00),
    ILI9881D_COMMAND_INSTR(0x03,0x73),
```

Possible improvements

Write panel driver against the display driver IC chip

- Rather than writing a driver tied to a specific panel model, write it against the Display Driver IC chip itself
- Expose all possible resolutions, lane counts, and configuration options as optional Device Tree properties
- Publish proper Device Tree bindings for each supported configuration
- Makes the driver reusable across any panel that uses the same Display Driver IC, regardless of the panel vendor

Thank You!
Questions?

Subhajit Ghosh
subhajit.ghosh@tweaklogic.com
https://www.linkedin.com/in/subhajit-ghosh-26389a45