Testing and remote access to embedded system
DPI/LVDS display output

Marek Vasut

June 28th, 2023
Table of content

- Motivation and why own hardware
- Display busses and capture options
- Approaches to the problem at hand
- Hardware build and testing
- LVDS encore
- Wrap up
Marek Vasut

- U-Boot bootloader custodian
- Linux kernel developer
- OE contributor
- FPGA hobbyist
- Consultant
Motivation

- SoM vendor development kit panel (re)testing
  - Too many panel options for each development kit
  - Testing them all means constant unplugging and replugging
  - Eventually the connector fails …

- Remote access
  - Sometimes it is not viable to ship hardware
  - Hardware in limited quantity or early prototype stage …
  - Sometimes it is convenient to put the hardware in rack and use remote access from anywhere

- CI testing
  - Automated SoM IO test in production is a must
  - Automated test of all available panels on each update is great
  - Automated testing improves confidence in release quality

Need  Device which is plugged in once and can do all of the above.
Available options

- **Grab fbdev content and stream it over e.g. ethernet**
  - **cons f1** Does not work (well) with modern DRM subsystem
    (fbdev is deprecated for over a decade)
  - **cons f2** A fully assembled frame buffer may not even exist in DRAM
  - **cons f3** Does not provide any information about the situation past the
    video buffer in DRAM (CRTC/BRIDGE(s)/CONNECTOR(s)),
    i.e. the pixels which are sent to the panel itself

- **Weston RDP backend**
  - **cons w1** Overhead of assembling and payloading buffers to RDP stream
  - **cons w2** Same as f3 above, no information past DRAM buffer

- **DRM subsystem CRTC CRC support**
  - **CRTC on some SoCs is capable of returning CRC of currently
    scanned out frame**
  - **Used for functional safety reasons (e.g. in automotive)**
  - **Used by [igt-gpu-tools] for CI testing**
  - **cons c1** No full image available for inspection/remote access
  - **cons c2** Similar to f3 above, no information past CRTC output

**Need** Capture the whole display output of the device.
Embedded systems display busses

**DPI**  Display Parallel Interface
- Likely oldest and simplest of still used interfaces
- Clock line, 1..24+ data lines, HS/VS/DE sync signals
- Wild source to sink data line mapping
- Uses LVTTTL 3V3 signalling

**FPD**  Flat Panel Display Link
- Uses LVDS – Low-Voltage Differential Signaling
- 1 differential clock lane + 3/4 differential data lanes
- Often incorrectly called LVDS
- FPD-Link is ≈ serialization of DPI
- Fewer available bus formats (1x 18bit and 2x 24bit)

**DSI**  MIPI Display Serial Interface
- Entirely different from the above
- MIPI Alliance standard
- Differential signaling, multiple PHYs, HS/LP mode
- Packet based protocol with back channel
DPI bus

- Clock, data, HS, VS, DE signals
- Closely matches panel behavior
- Maximum clock rate limited by too many data lines
- Often found in older panels
- Signal polarity and sampling edge is important
- Different modes – HS/VS or DE
- Widely varying pixel formats

DPI Host

```
---------. .---------.
R[7:0] |>=============|       |
G[7:0] |>=============|       |
B[7:0] |>=============|       |
PCLK  |>-------------| Panel |
     |>-------------| or    |
     |>-------------| Bridge |
     |>-------------|       |
--------' '---------'
```
DPI timing and synchronization

Linux 6.4-rc6
[Documentation/devicetree/bindings/display/panel/panel-timing.yaml]
DPI capture

- What should the device capture?
  - Active area only – fewer data, but incomplete
  - Everything including margins – great for CI test

- DPI panel maximum resolution $\approx 1024 \times 600$ 24bpp 60FPS

- Sampling other interesting signals
  - Use signals above the 24bpp
  - Capture 32bpp instead
  - Interesting signals include HS/VS/DE and even PWM...

- Data rate at $1024 \times 600$ 32bpp 60 FPS including margins
  - Let's assume CDTech S070PWS19HP-FC21 panel
  - Total width $W(\text{pixels}) =$
    \[
    HSA + HBP + HACT + HFP = 20 + 140 + 1024 + 160 = 1344
    \]
  - Total height $H(\text{lines}) =$
    \[
    VSA + VBP + VACT + VFP = 3 + 20 + 600 + 12 = 635
    \]
  - Total data rate
    \[
    D = W \times H \times (32\, \text{bpp}/8) \times 60\, \text{FPS} \approx 204\, \text{MiB/s}
    \]

- Data rate at $1920 \times 1080$ 32bpp 60 FPS $\approx 500\, \text{MiB/s}$

Need high bandwidth interface
static const struct drm_display_mode cdtech_s070pws19hp_fc21_mode = {
    .clock = 51200,
    .hdisplay = 1024,
    .hsync_start = 1024 + 160,
    .hsync_end = 1024 + 160 + 20,
    .htotal = 1024 + 160 + 20 + 140,
    .vdisplay = 600,
    .vsync_start = 600 + 12,
    .vsync_end = 600 + 12 + 3,
    .vtotal = 600 + 12 + 3 + 20,
    .flags = DRM_MODE_FLAG_NHSYNC | DRM_MODE_FLAG_NVSYNC,
};
High bandwidth interfaces

What is available on most PCs and can receive high bandwidth continuous stream of data?

Ethernet Gigabit

- **pro** Almost every PC has gigabit ethernet plug
- **con** Insufficient bandwidth of $\approx 125$ MiB/s

Ethernet Multiple bonded gigabit ethernets

- **con** Too complex

Ethernet 10G ethernet

- **con** Seldom available on contemporary PCs
- **con** Complex and expensive on FPGA TX side

USB 2.0

- **pro** Almost every PC has USB 2.0 plug
- **con** Insufficient bandwidth of $\approx 60$ MiB/s
  - ▶ Compression might help, but not useful for testing and CI

USB 3.0

- **pro** Almost every PC has USB 3.0 plug
- **pro** Enough bandwidth on root port of $\approx 625$ MiB/s

Need to use USB 3.0 and avoid any downstream hubs
32bit input to USB 3.0 bridge chips

Existing 32bit-input-to-USB3.0 bridge chips:

**FTDI** FT602Q
- 32bit RX FIFO to USB 3.0 UVC (USB Video Class) device
- **pro** No need for firmware
- **pro** USB UVC device, behaves like a webcam
- **con** FT602Q is 100MHz or 66MHz clock source
- **con** Need asynchronous FIFO between DPI⇔FT602Q

**Cypress** FX3
- 32bit general purpose interface, CPU, DMA, USB 3.0 device
- **pro** Flexible due to CPU and 32bit up to 100 MHz GP interface
- **pro** GP interface does support external clock input
- **pro** Familiar ARM926EJS ARMv5 core
- **pro** Documentation for the chip peripherals is good
- **pro** Firmware examples contain CPI camera to UVC video demo
- **con** CPI to UVC demo must be adapted and pixel format tweaked
- Not all of vendor firmware SDK runs on Linux
- Vendor firmware SDK is dubiously licensed outdated **blobware**
Multiple approaches

- DPI → FPGA → bridge as UVC → Host PC
  - Maximum compatibility
  - Native USB UVC support in most OSes (not enough)
  - FPGA adds to price and complexity
  - FX3 as UVC requires blobware firmware
  - FX3 input state machine design tool does not work on Linux

- DPI → bridge as FIFO → Host PC
  - Maximum simplicity
  - Requires custom host software (easy)
    - Read data from bridge
    - Display / write to file / pass to e.g. gstreamer
Failed UVC blobwork

Multiple problems:

1. UVC USB Video Class does not support varying resolution
2. Both FT602Q and FX3 in UVC mode expect CPI sensor input
UVC pixel format inflexibility

- USB UVC is USB-IF (Implementers Forum) standard
- Stream width, height, pixel format, bitrate, ... all encoded in static USB descriptors
- Not all pixel formats supported by various OSes are part of the USB-IF standard
- Some pixel formats are non-standard extensions, usually poorly documented
- BGRX8888 extension pixel format now in Linux 6.1.y media: uvcvideo: Add GUID for BGRA/X 8:8:8:8
- Cypress firmware can be patched to produce this format
- As far as I can tell, patch cannot be distributed due to SDK licensing
UVC resolution inflexibility

- UVC USB Video Class does not support changing resolution during streaming
- The DPI input resolution may change at runtime or even fluctuate (during debugging)
- Linux kernel `uvcvideo` driver expects continuous stream
- Linux kernel `uvcvideo` driver checks line width
- Frames with short lines dropped without module parameter: `uvcvideo nodrop=1`
- Both `gstreamer` and `ffmpeg` check V4L2 frame size
- Short frames are dropped unless either is patched

😢 This is not really seamless experience
CPI timing and synchronization

- CPI uses two sync signals to denote active area with valid pixels

**LV** Line Valid – Indicates active columns \( \approx HFP + HSA + HBP \)

**FV** Frame Valid – Indicates valid rows \( \approx VFP + VSA + VBP \)

- Example: MT9P006 Pixel Array Structure + LV and FV

```
+---------+-------------------------------------+---------+
|         | dark rows                           |         |
|         | v                                  |         |
/ +---------+-------------------------------------+---------+
| #        | ^                                  | #       |
| #        | #                                  | # dark  |
| columns | #                                  | # dark  |
| hactive | #                                  | hactive |
|<------->#<-------+--------------------------->#<------->|
|         | v                                  |         |
|         | #                                  |         |
|         | v                                  |         |
\ +---------+-------------------------------------+---------+
|         | dark rows                           |         |
|         | v                                  |         |
```

```
CPI vs. DPI timing

Need adapter which converts DPI sync signals to CPI sync signals
FPGA asynchronous FIFO

- Crossing clock domains with an Asynchronous FIFO
  The ZipCPU by Gisselquist Technology
- A FIFO used to move wider data from one clock domain to another
- Width and depth often configurable
FPGA asynchronous FIFO implementation

- Assume $f_{in} < f_{out}$
- Assume FIFO width is 32 bits and depth is slightly more than 1 full line worth of pixels
- FIFO fill level $\approx LV$
- When FIFO contains 1 line worth of input data, it can be flushed into the output bridge
- Padding to the width configured in the UVC descriptors is necessary
- Line counting to emulate FV is necessary
- The FV can be a short pulse (not a full line) for FX3 to recognize it
- If FIFO output does not produce data for too long $\Rightarrow$ this is loss of signal
- FIFO must be carefully placed in FPGA and timing constraints are a much
- Altera Cyclone IV/E maxed out at $f_{out} \approx 65$ MHz
Failed UVC blobwork is failed

Summary:
- UVC was not a win due to necessary workarounds
- Firmware patching was extremely problematic
- DPI to CPI adapter using FPGA is convoluted
Success with bridge as FIFO

- FTDI FT601Q is 32bit FIFO to USB 3.0, but still clock source
- Cypress FX3 is capable of being a clock sink
- Capture raw byte stream using FX3, process on Host PC
- The FX3 firmware and tooling is still a problem
- The vendor tooling is a problem
The sigrok project aims at creating a portable, cross-platform, Free/Libre/Open-Source signal analysis software suite that supports various device types (e.g. logic analyzers, oscilloscopes, and many more).

fx2lafw
fx2lafw is an open-source firmware for Cypress FX2 chips which makes them usable as simple logic analyzer and/or oscilloscope hardware.

If only there was similar firmware for Cypress FX3.
fx3lafw

- fx3lafw [link] by Marcus Comstedt
  This is an open source firmware for using a Cypress FX3 USB controller as a logic analyzer with sigrok. It does not rely on any libraries or tools provided by Cypress under license.
- Suggested by t4nk at sigrok channel during unrelated discussion – THANK YOU!
- No need to use the Cypress blobware anymore
- Compatible with fx2lafw with extension to support faster sampling
- Uses clock generated by FX3 to oversample the 32bit bus
- sigrok support is implemented by a few easy patches to libsigrok [link], currently outdated, but rebase is easy
What is missing

- Switch the fx31afw to use external clock from DPI
- Capture data using sigrok and pipe them into e.g. gstreamer
- Replace sigrok with something more lightweight
- Build more permanent hardware setup
Patching fx3lafw

- Clock direction bit must be flipped
- SYNC/SPEED bits must be enabled
  Needed for higher speed capture and capture synchronous to input PCLK
- DLL must be disabled else slight data loss happens
  Resulting image crawls slowly to the left and wraps around
- Input PCLK invert configuration (for selection of PCLK sampling edge) is a good tunable to have, implemented using libusb control endpoint, like other fx2lafw configuration messages
- Use libusb fxload example tool to load firmware into FX3
  (or run sigrok-cli twice, the first time will fail to re-enumerate the FX3 as it changed bus from USB 2.0 to USB 3.0 and libsigrok cannot handle that yet)
- Patched fx3lafw [link]
Build and use fx3lafw

```bash
# Udev rule (use 'udevdadm control -R' to reload the rules once added)
$ cat /etc/udev/rules.d/92-cypress.rules
SUBSYSTEM=="usb", ATTRS{idVendor}=="04b4", ATTRS{idProduct}=="00f3", MODE="0666"

fx3lafw $ git clean -fqdx
fx3lafw $ make -j$(nproc)
# For sigrok purposes
fx3lafw $ cp fx3lafw-cypress-fx3.fw /usr/share/sigrok-firmware/
# For libusb/fxload purposes
fx3lafw $ cp -Lv fx3lafw-cypress-fx3.fw fx3lafw-cypress-fx3.img
fx3lafw $ /libusb/examples/fxload -t fx3 -i fx3lafw-cypress-fx3.img
```
sigrok continuous capture

- The sigrok-cli tool supports triggering capture from command line
- Captured data can be stored into a file in different formats:
  - -O binary or -O csv ...
  - Capture into file is useful for detailed analysis
  - The file can be a named pipe (FIFO) for realtime visualization
  - FIFO needs a consumer too
- sigrok supports software triggers -t channel, useful to trigger on edge of sync signal(s)
- Gerhard Sittig of sigrok fame suggests --continuous option

```bash
$ mkfifo /tmp/fifo
$ sigrok-cli -D -d fx2lafw --continuous --channels D0,D1,D2,D3,D4,D5,D6,D7,D8,D9,D10,D11,D12,D13,D14,D15,D16, D17,D18,D19,D20,D21,D22,D23,D24,D25,D26,D27,D28,D29,D30,D31 -t D24=r --config samplerate=192m -l 5 -o /tmp/fifo -O binary
```
Gstreamer FIFO consumer

- The last step is to create FIFO consumer that can visualize the data
- Gstreamer is a good candidate with gst-launch-1.0 tool
- Easily assemble a pipeline
- It is necessary to know the input stream resolution, otherwise the visualization will be distorted
- Lets assume CDTech S070PWS19HP-FC21 panel again 1024x600 active area, 1344x635 including margins

```bash
$ W=1344 H=635 S=$((4*$W*$H)) ; gst-launch-1.0 filesrc location=/tmp/fifo blocksize=${S} !
    video/x-raw,width=${W},height=${H},framerate=60/1,format=RGBx !
    queue !
    videoconvert !
    autovideosink
```
Replacing the RX with light weight tool

- A more light-weight tool called fx3stream is available at [link]
- Based on libusb with optional xlib and gstreamer dependencies
- The tool is currently rudimentary
- Provides three sink options – FIFO, Gstreamer, X11 window
- Gstreamer sink is an appsrc which push feeds gstreamer pipeline
- Provides frame rate counter (using gstreamer fpsdisplaysink)
- Provides sync signals visualization
- Compile, upload fx3lafw to FX3 using fxload, run the tool

Usage: ./stream-gst <width> <height> <pclkpol> <vsdetpol> <colorsync>
width ....... width of input frame, including HSA HBP HACT HFP
height ...... height of input frame, including VSA VBP VACT VFP
pclkpol ..... 0 - PCLK sampled on RISING edge
            1 - PCLK sampled on FALLING edge
vsdetpol .... 0 - VSYNC lock on RISING edge
              1 - VSYNC lock on FALLING edge
colorsync ... 0 - do not color sync signals in frame
              1 - color active HIGH sync signals
              2 - color active LOW sync signals
GSTFPS=1 environment variable enables frame rate counter overlay
Demo – Linux is booting

- Chefree CH101OLHLWH-002 1280x800 panel connected via DPI-to-LVDS bridge. The capture is attached to the DPI, i.e. before the DPI-to-LVDS bridge.

- Host: $ ./stream-gst 1440 823 0 0 0

```
[ 1.570111] inx-drm display-subsystem: bound inx-ipps3-crtc.3 (ops 0xc00b7f44)
[ 1.574491] inx-drm display-subsystem: bound inx-ipps3-crtc.6 (ops 0xc00b7f44)
[ 1.584801] inx-drm display-subsystem: bound inx-ipps3-crtc.7 (ops 0xc00b7f44)
[ 1.952193] inx-drm display-subsystem: bound disp0 (ops 0xc8d59e0)
[ 2.099921] [drm] Initialized inx-drm 1.0.0 20120907 for display-subsystem on minor 0
```

```bash
$ ./stream-gst 1440 823 0 0 0
```

Host:

```
[ 1.570111] inx-drm display-subsystem: bound inx-ipps3-crtc.3 (ops 0xc00b7f44)
[ 1.574491] inx-drm display-subsystem: bound inx-ipps3-crtc.6 (ops 0xc00b7f44)
[ 1.584801] inx-drm display-subsystem: bound inx-ipps3-crtc.7 (ops 0xc00b7f44)
```

```bash
$ ./stream-gst 1440 823 0 0 0
```

Host:

```
[ 1.570111] inx-drm display-subsystem: bound inx-ipps3-crtc.3 (ops 0xc00b7f44)
[ 1.574491] inx-drm display-subsystem: bound inx-ipps3-crtc.6 (ops 0xc00b7f44)
```

```bash
$ ./stream-gst 1440 823 0 0 0
```
Demo – Linux, FPS overlay

▶ Host: $ GSTFPS=1 ./stream-gst 1440 823 0 0 0
Demo – Linux, Sync signals visualization and FPS

- Great for CI and debugging purposes.
- Host: $ GSTFPS=1 ./stream-gst 1440 823 0 0 2

rendered: 9937, dropped: 0, current: 60.04, average: 59.95
Demo – Weston, with FPS, with FPS

▶ Device: gst-launch-1.0 videotestsrc!
video/x-raw,width=800,height=600 ! queue !
fpsdisplaysink video-sink=autovideosink

▶ Host: $ GSTFPS=1 ./stream-gst 1440 823 0 0 2
Loss of signal handling

- The FX3 expects continuous PCLK input
- Loss of signal triggers GPIF PIB interrupt and stop of capture
- Patch out that interrupt generation
- The FX3 also contains clock loss detection interrupt
- Test the clock loss interrupt generation, it does work
- With patched out GPIF PIB stop interrupt, clock loss interrupt is not needed
- FX3 is not sending any data on clock loss, and recovers when clock are back
Hardware

- The Cypress CYUSB3KIT-003 is convenient
- It provides two 2x20 2.54mm plus spaced 41.5mm apart
- It is possible to use easy to get 2.54mm M-x cables
- With fly wiring, higher clock frequencies are a problem
- Making permanent hardware connection is a matter of trivial adapter PCB
- Trivial adapter PCB can be designed in KiCad and manufactured at PCB house
- PCB houses these days can do a lot for you, even populate the PCB ...
- Make sure you check your newly populated PCB for shorts ...
KiCad – Adapter board schematic

Make sure you run ERC
KiCad – Adapter board PCB

Make sure you run DRC
KiCad – Entire assembly

Stream from the device looks as expected, no surprises there.
LVDS bus

- Basically a serialization of DPI bus
- ANSI/TIA/EIA-644-A
- Pixel format limited to three options:
  - jeida-18 (SPWG, LDI, VESA)
  - jeida-24 (DSIM, LDI)
  - vesa-24 (VESA)
- Differential signalling, 1 clock lane, 3 or 4 data lanes
- High resolution panels often use dual-link LVDS

---

LVDS Host

<table>
<thead>
<tr>
<th>CLK P/N</th>
<th>&gt;=============</th>
<th>Panel</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0 P/N</td>
<td>&gt;=============</td>
<td>or</td>
</tr>
<tr>
<td>D1 P/N</td>
<td>&gt;=============</td>
<td>Bridge</td>
</tr>
<tr>
<td>D2 P/N</td>
<td>&gt;=============</td>
<td></td>
</tr>
<tr>
<td>D3 P/N</td>
<td>&gt;=============</td>
<td></td>
</tr>
</tbody>
</table>
### LVDS bus

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>DATA0</td>
<td>G0</td>
<td>R5</td>
<td>R4</td>
<td>R3</td>
<td>R2</td>
<td>R1</td>
<td>R0</td>
<td></td>
</tr>
<tr>
<td>DATA1</td>
<td>B1</td>
<td>B0</td>
<td>G5</td>
<td>G4</td>
<td>G3</td>
<td>G2</td>
<td>G1</td>
<td></td>
</tr>
<tr>
<td>DATA2</td>
<td>DE</td>
<td>VS</td>
<td>HS</td>
<td>B5</td>
<td>B4</td>
<td>B3</td>
<td>B2</td>
<td></td>
</tr>
<tr>
<td>DATA3</td>
<td>00</td>
<td>B7</td>
<td>B6</td>
<td>G7</td>
<td>G6</td>
<td>R7</td>
<td>R6</td>
<td></td>
</tr>
</tbody>
</table>

**Figure:** LVDS JEIDA-18 3-lane LVDS bus

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>DATA0</td>
<td>G2</td>
<td>R7</td>
<td>R6</td>
<td>R5</td>
<td>R4</td>
<td>R3</td>
<td>R2</td>
<td></td>
</tr>
<tr>
<td>DATA1</td>
<td>B3</td>
<td>B2</td>
<td>G7</td>
<td>G6</td>
<td>G5</td>
<td>G4</td>
<td>G3</td>
<td></td>
</tr>
<tr>
<td>DATA2</td>
<td>DE</td>
<td>VS</td>
<td>HS</td>
<td>B7</td>
<td>B6</td>
<td>B5</td>
<td>B4</td>
<td></td>
</tr>
<tr>
<td>DATA3</td>
<td>00</td>
<td>B1</td>
<td>B0</td>
<td>G1</td>
<td>G0</td>
<td>R1</td>
<td>R0</td>
<td></td>
</tr>
</tbody>
</table>

**Figure:** LVDS JEIDA-24 4-lane LVDS bus

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>DATA0</td>
<td>G0</td>
<td>R5</td>
<td>R4</td>
<td>R3</td>
<td>R2</td>
<td>R1</td>
<td>R0</td>
<td></td>
</tr>
<tr>
<td>DATA1</td>
<td>B1</td>
<td>B0</td>
<td>G5</td>
<td>G4</td>
<td>G3</td>
<td>G2</td>
<td>G1</td>
<td></td>
</tr>
<tr>
<td>DATA2</td>
<td>DE</td>
<td>VS</td>
<td>HS</td>
<td>B5</td>
<td>B4</td>
<td>B3</td>
<td>B2</td>
<td></td>
</tr>
<tr>
<td>DATA3</td>
<td>00</td>
<td>B7</td>
<td>B6</td>
<td>G7</td>
<td>G6</td>
<td>R7</td>
<td>R6</td>
<td></td>
</tr>
</tbody>
</table>

**Figure:** LVDS VESA-24 4-lane LVDS bus
LVDS to DPI

- LVDS to DPI deserializer chips do exist
- For example TI DS90CF384, OnSemi FIN3386, Thine THC63LVDF84B ...
- Use one of the chips and convert LVDS problem to already solved DPI problem
- Use capacitors close to the chip supplies
- Carefully route LVDS differential lines
- LVDS lines require 100Ω termination
Dual-link LVDS

- Uses 8 differential pairs, transmits 2 pixels at a time
- Used to drive 1920x1080 FullHD panels where single-link LVDS does not suffice
- Pixels always sent as odd-even pixel pair
- Likely can be captured using TI DS90CF388 de-serializer and two FX3
- Pixel reassembly on Host PC, possibly using Gstreamer ORC SIMD
Demo – Linux is booting

- Lets assume EDT ETML0700Y5DHA panel
- Host: $ ./stream-gst 1344 635 0 1 0
Host: $ GSTFPS=1 ./stream-gst 1344 635 0 1 0
Demo – Linux, Sync signals visualization and FPS

▶ Great for CI and debugging purposes
▶ Host: $ GSTFPS=1 ./stream-gst 1344 635 0 1 2

rendered: 52440, dropped: 0, current: 59.92, average: 60.00
Demo – Weston, with FPS, with FPS

- Host: $ GSTFPS=1 ./stream-gst 1344 635 0 1 2
- The image stretch left and right is CRTC leaving the data lines in pre-sync state during sync period instead of setting them to zero
KiCad – Adapter board schematic

Make sure you run ERC

It is a really good idea to learn KiCad key bindings (it is faster)
KiCad – Schematic symbol design

LVDS de-serializer schematic symbols may not be in the KiCad library

Derive symbol from existing high quality symbols in the library
KiCad – Adapter board PCB

Make sure you run DRC
Soldering TSSOP56 package at home takes practice and patience
Same for 0402 passive components like resistors, don’t inhale them

Get a good flux, soldering iron, fume extractor, maybe a loupe …
KiCad – Populated board
KiCad – Entire assembly

Stream from the device looks as expected, no surprises there.
Next steps – MIPI DSI

- Packet based
- Multiple different PHY options (C-PHY/D-PHY/M-PHY)
- Likely target D-PHY as it is most common in embedded
- Would require FPGA PHY ≈ HS/LP RX, byte aligner, depacketizer
- HS/LP input into FPGA can be implemented using a few termination resistors and suitable FPGA with the right IO voltage on the right banks
- Multiple FPGA CSI-2 D-PHY RX exists already:
  - CircuitValley [link]
  - gatecat [link]
- Extracted packets can be fed into FX3 and captured on PC
- sigrok protocol decoder support might be useful here too
Wrap up

- Hardware is obtainable – FX3 kit and adapter board
- Software is available
- Build is easy
Thank you for your attention

Questions?

Marek Vasut <marek.vasut+eoss23@mailbox.org>