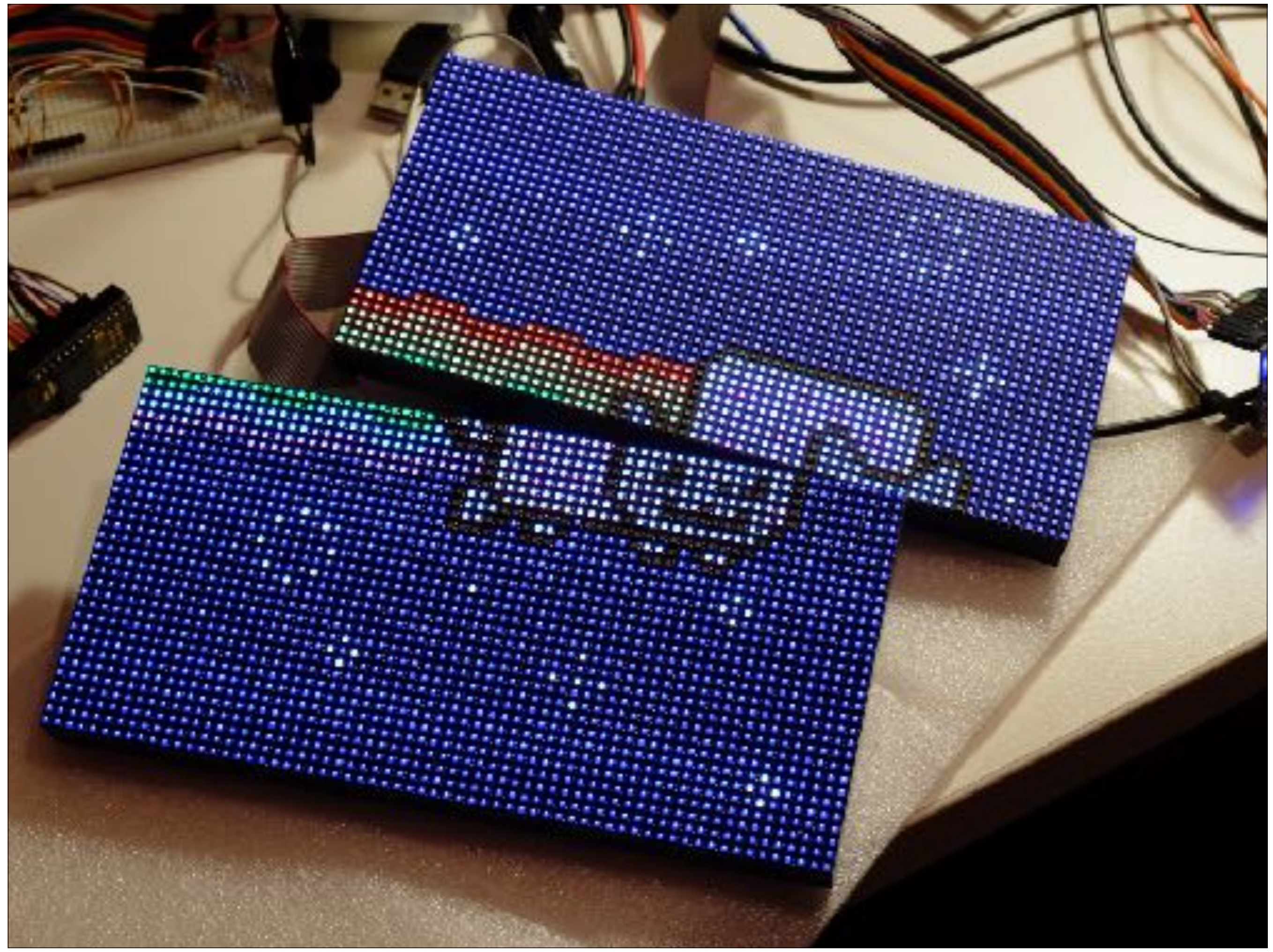
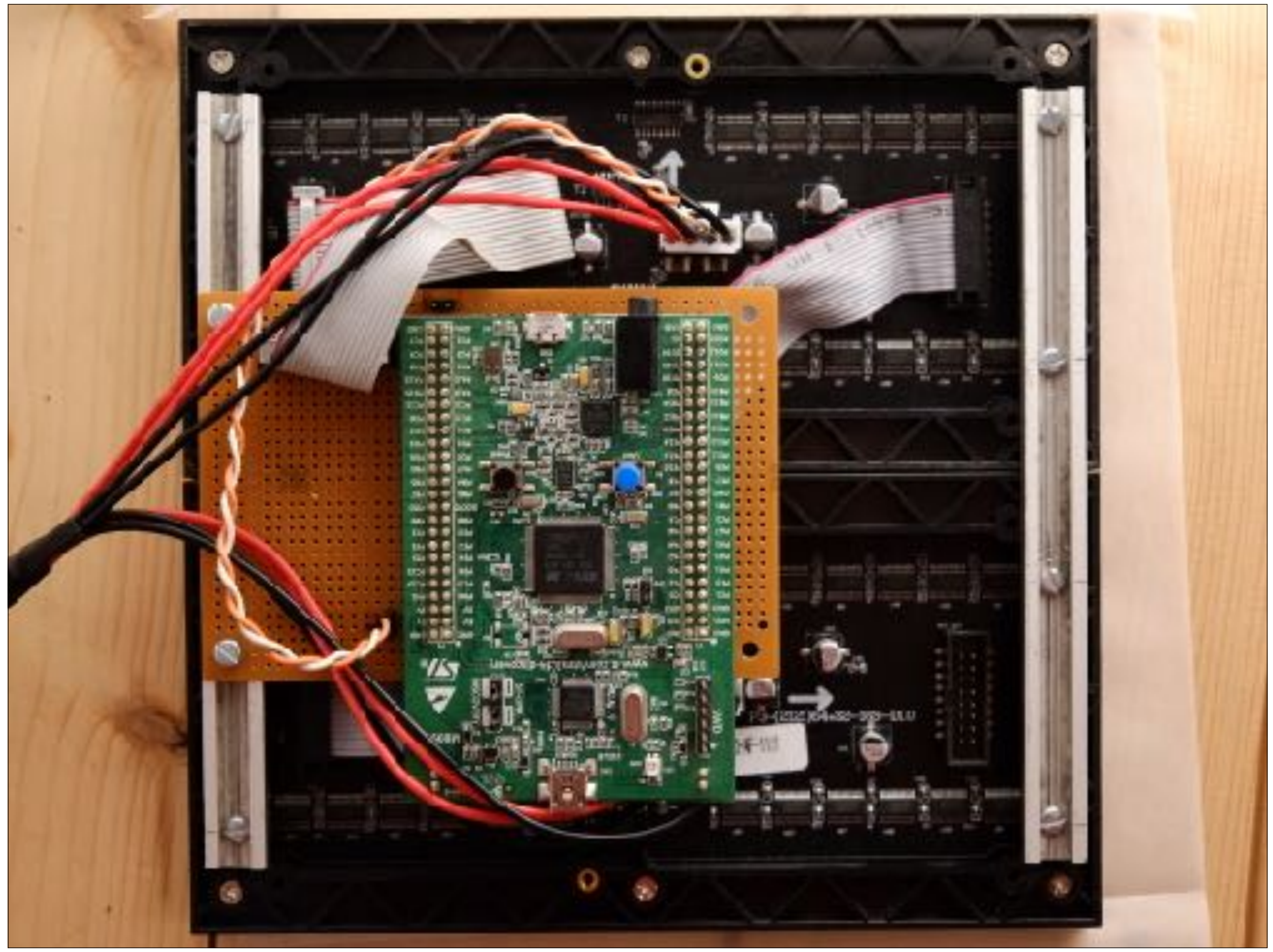
If you can read this, the projector setup is borked ;)

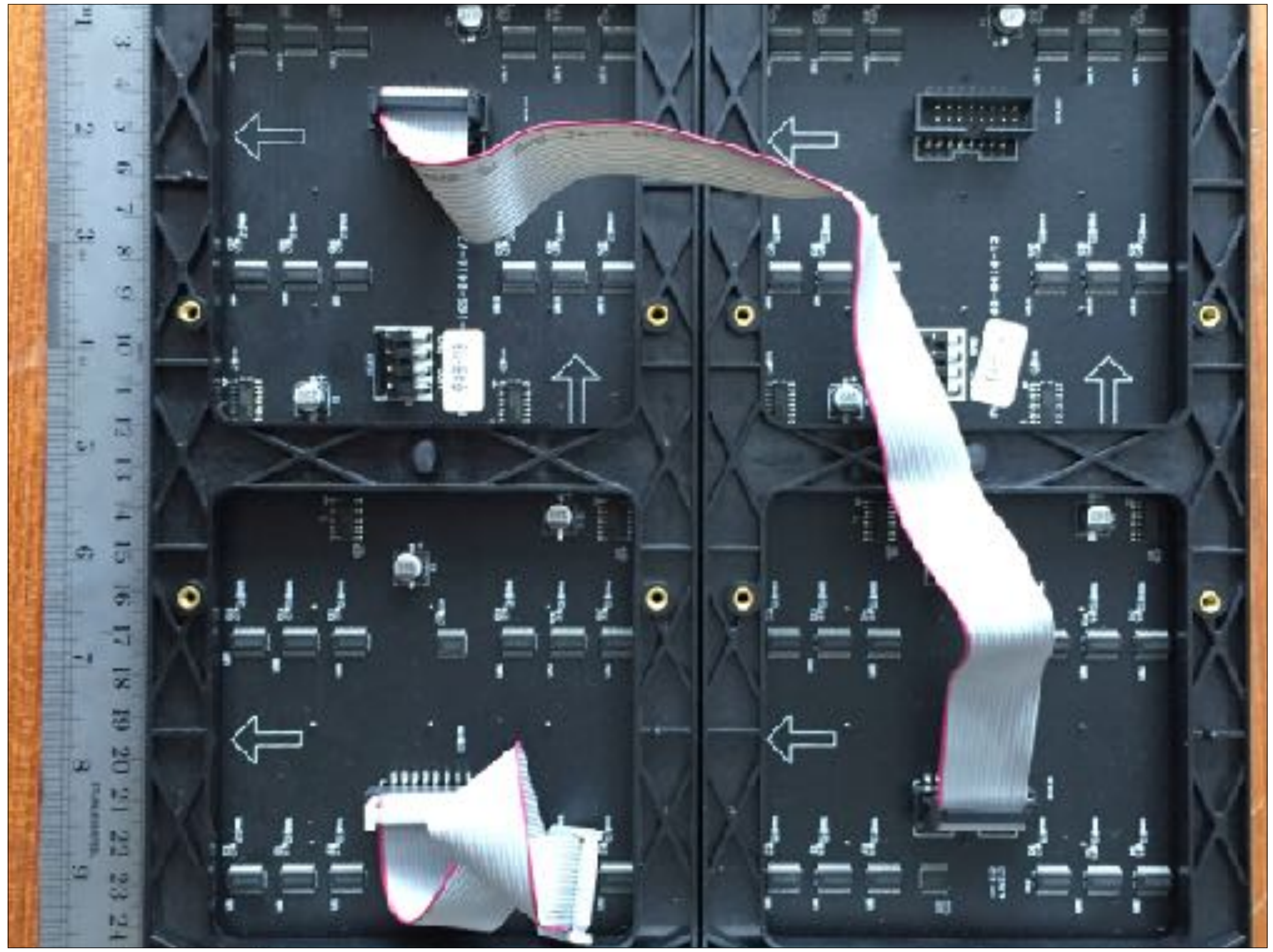
It all started with … Shenzhen
LED panels are cool — I find them abnormally fascinating :D **RGB**
Lots of LEDs on a PCB, in modules with easy mounting/wiring
Your driver circuit needs to scan video through these manually
Manual BCM/PWM!
High FPS crucial



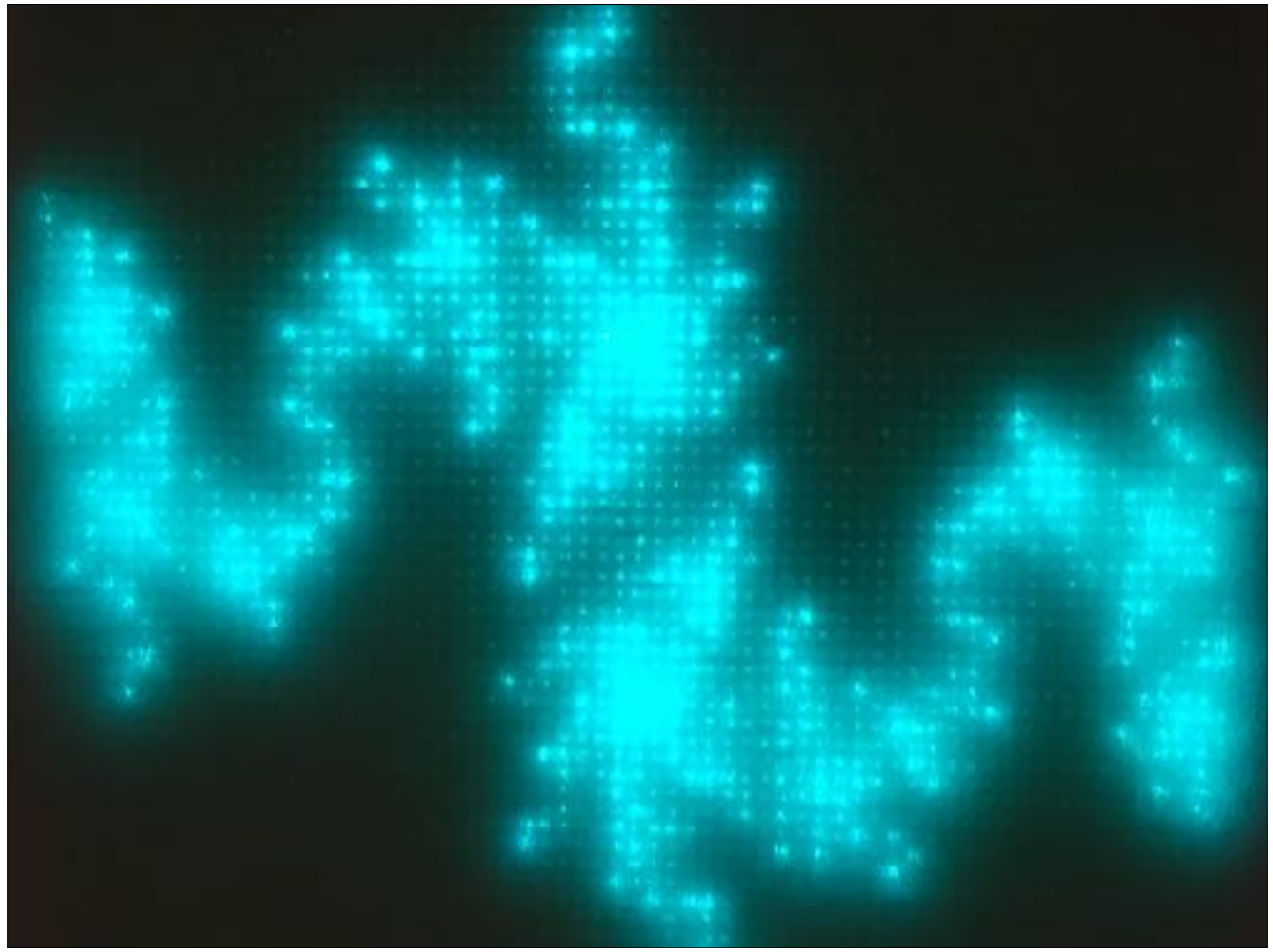
I got two 64x32, stuck together to make 64x64 — 4K display!
My first test, purely **bitbanged** (haha) in software using an mbed — **EXAMPLE**
Bitbanging is a fine way to try POC, test out an algorithm, etc.



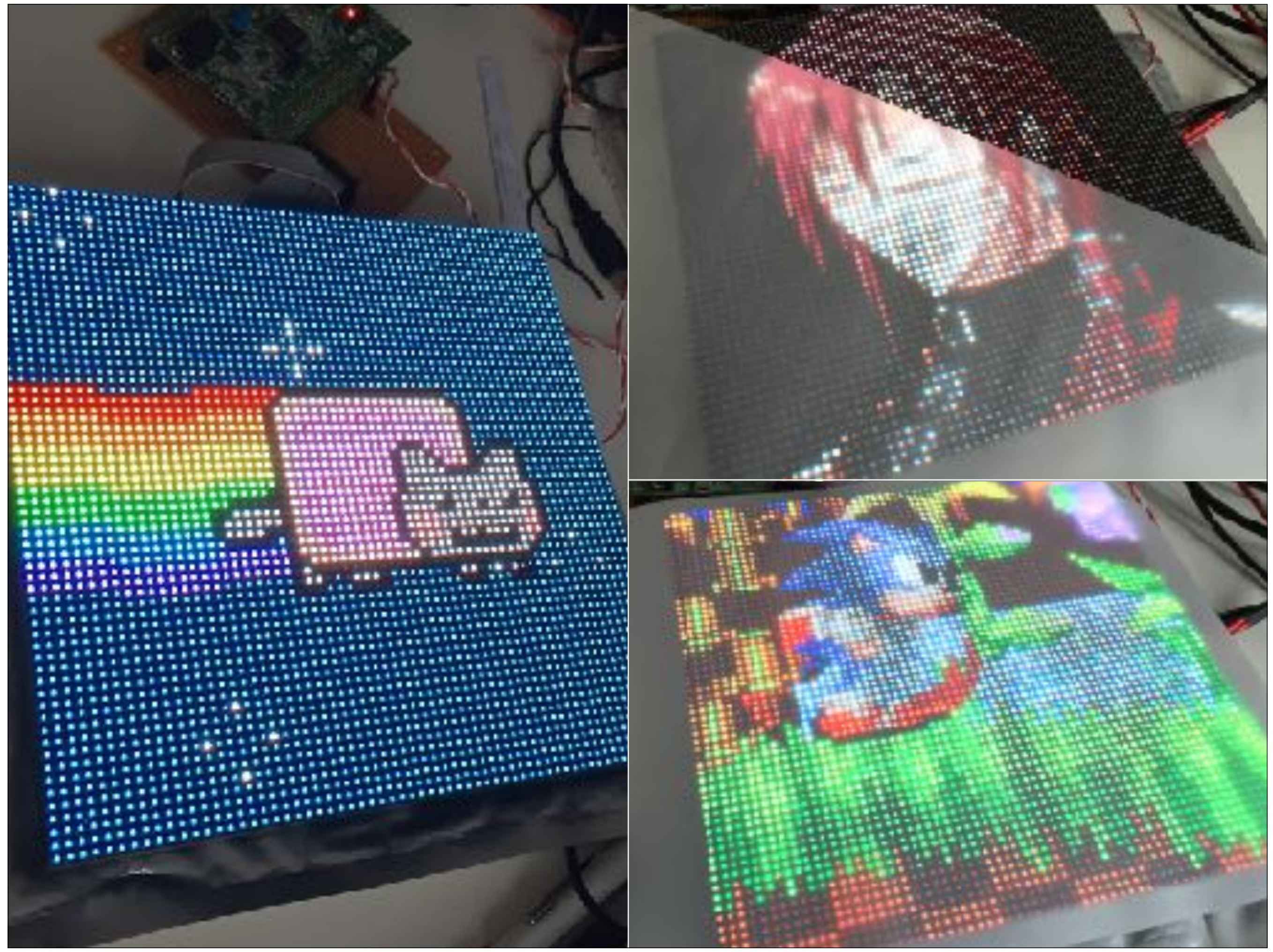
First proper go — driven from **STM32**
Big shift register
Clock data in for a whole row, then latch — example later
Full-colour RGB — you have to do the PWM/brightness control by hand



Daisychain serial data out of one, into next



Got the panels running from my STM32, playing back **animations**
Demo effects **plasma, blobs, fractals**



Also **MAME** captures & GIFs.

Timing is key — **flicker** very noticeable! Will write about using **DMA controller**. Very **predictable timing.**

Pleased with this **~170Hz 33BPP** / 11BPC.

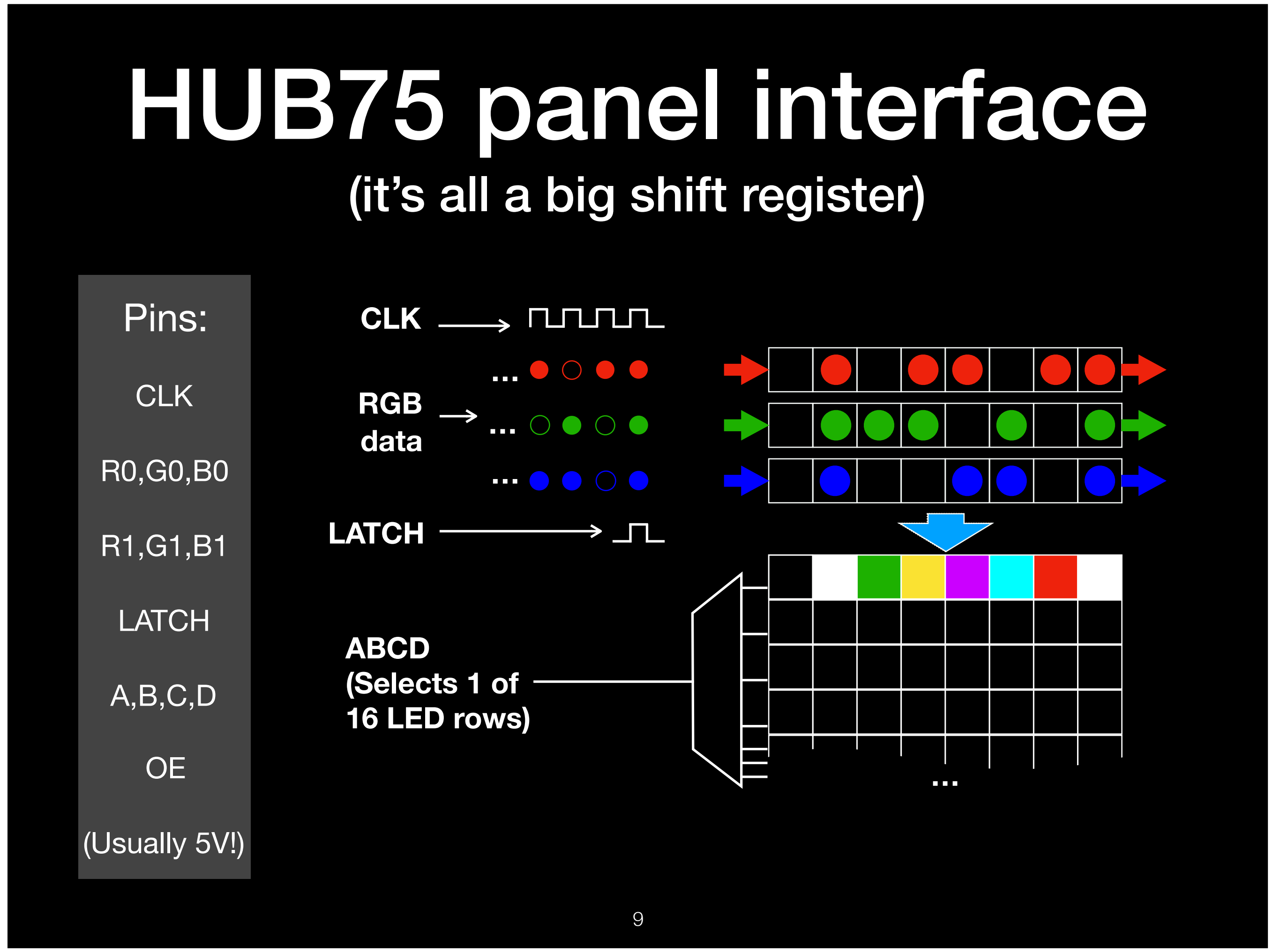
WAIT — but WHY? For **lulz**

BUT… wanted **network**, stream **video from phone**. WHY? Haha for lulz

LED panels from a Raspberry Pi

- Great library for driving LED panels from Raspberry Pi:
  - https://github.com/hzeller/rpi-rgb-led-matrix
- But, doesn't use DMA — ***it bitbangs, software loop***
- "The system needs constant CPU … roughly 30-40% of one core."
- To avoid flicker: "If you have a loaded system … you can ***reserve one core*** just for the refresh of the display" 😫

Okay, so I wanted **networking** — **RPI zzz** but Linux is just too convenient to ignore. But **Pi Zero** — **€5**-10!
Existing libs. Adafruit.
Crux of my argument: Bitbanging — great for simple tasks. **Rubbish for realtime**.
Dedicate a 64-bit Cortex A53? **Eww**. PiZero only has one core. **PROBLEM**



What do I need? HUB75 is a common interface. 5V. Some variation:
R,G,B data bit + **clock** into 3 shift regs, for R/G/B along a row
+**Latch** = energise data onto a row
row selected by 4 bits A,B,C,D
Data needs to go in about **25-30MHz**
As **fast** as possible — higher refresh rate, higher colour depth

Display Parallel Interface (DPI)

- Once upon a time, I was attaching an LCD to a RPi using DPI
- Parallel interface designed to drive TFT LCDs from BCM2835 — alternative to HDMI
- 24-bit pixel output (+ pixel clock, + sync bits) high-speed digital output, 3.3V CMOS

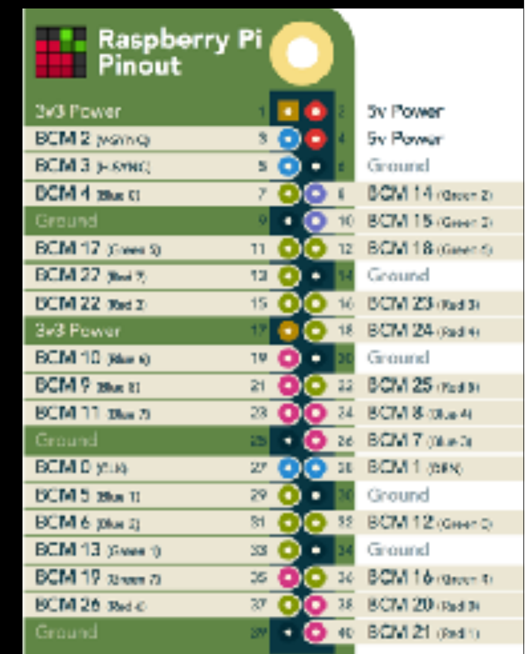


Image from pinouts.xyz

Interlude: I was using DPI for another project (to drive an LCD as intended)
Very **high speed** pixel output — up to >100MHz
Digital: takes a pixel, sends it out

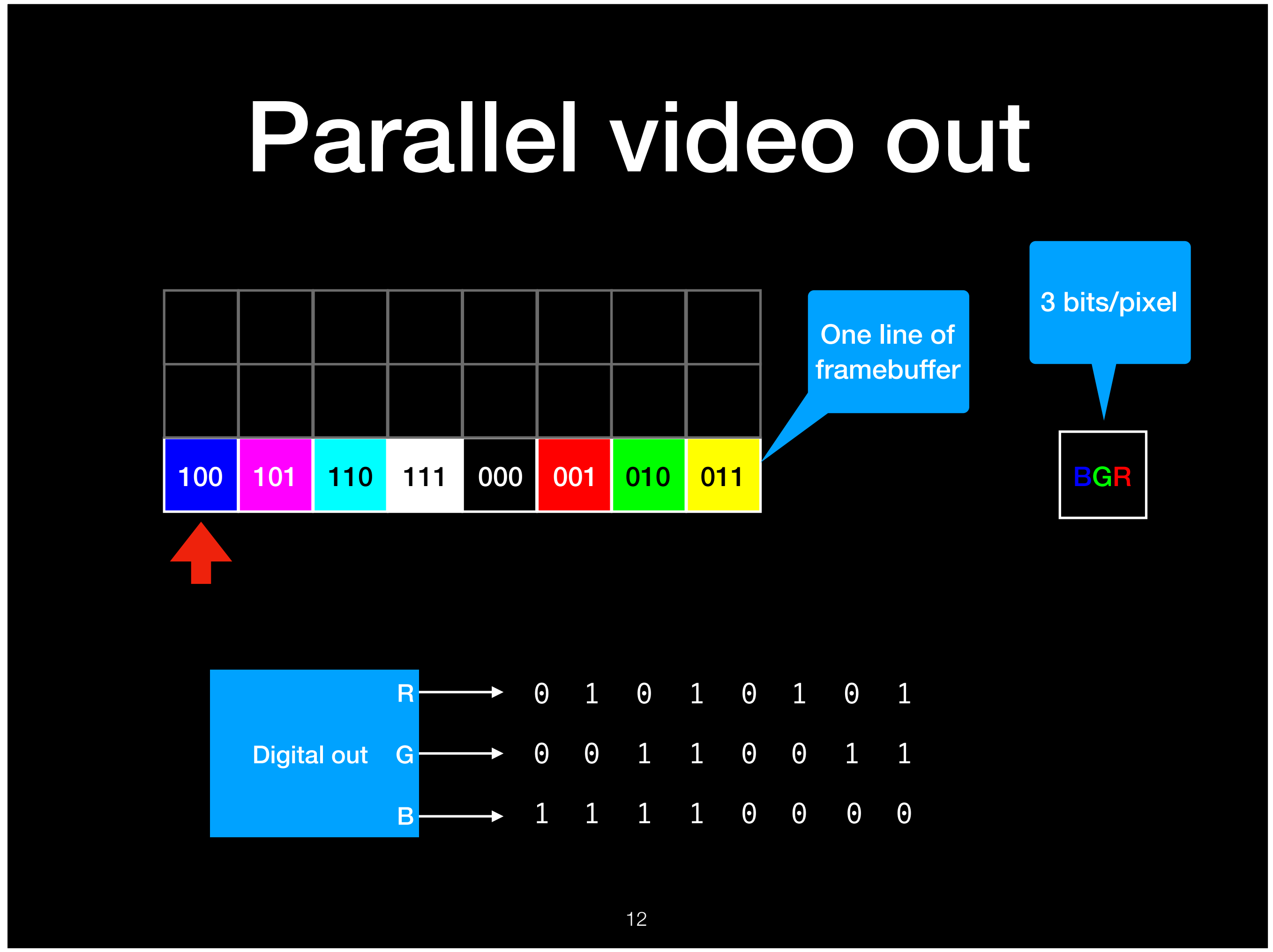
I haz an idea!

Put *pixel patterns* in the video framebuffer that send digital *data patterns* to DPI output pins!

No CPU overhead to display it

Guaranteed not to hiccup

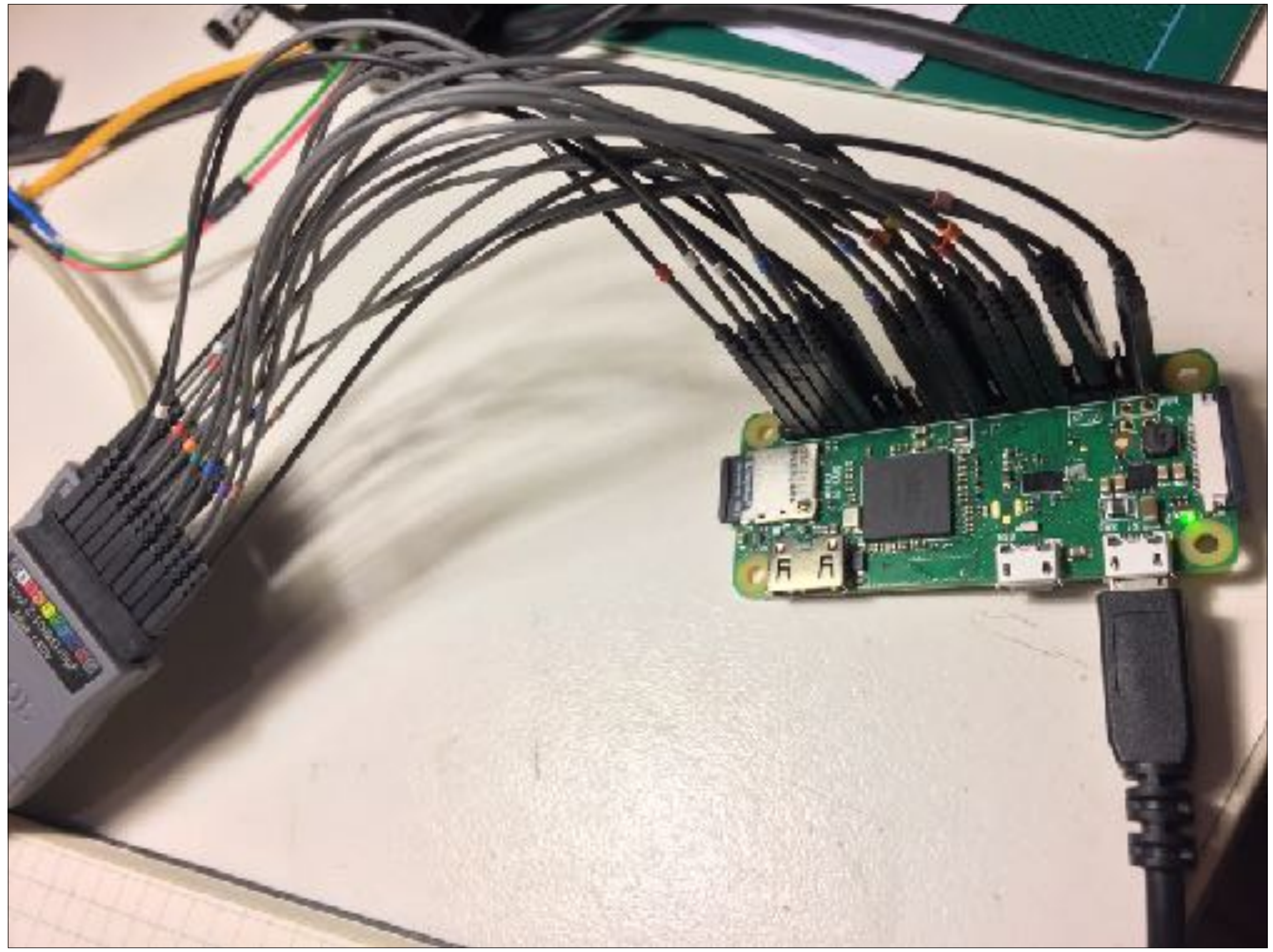
Patterns generate signals — **signals like HUB75**!



Taking an example, say we have 3 bits/pixel R,G,B framebuffer
Colour bits output with regular timing
Pixel bits — as clock, or data, anything
HSYNC at end of line can **latch** a block — a bit like the row **latch** in HUB75

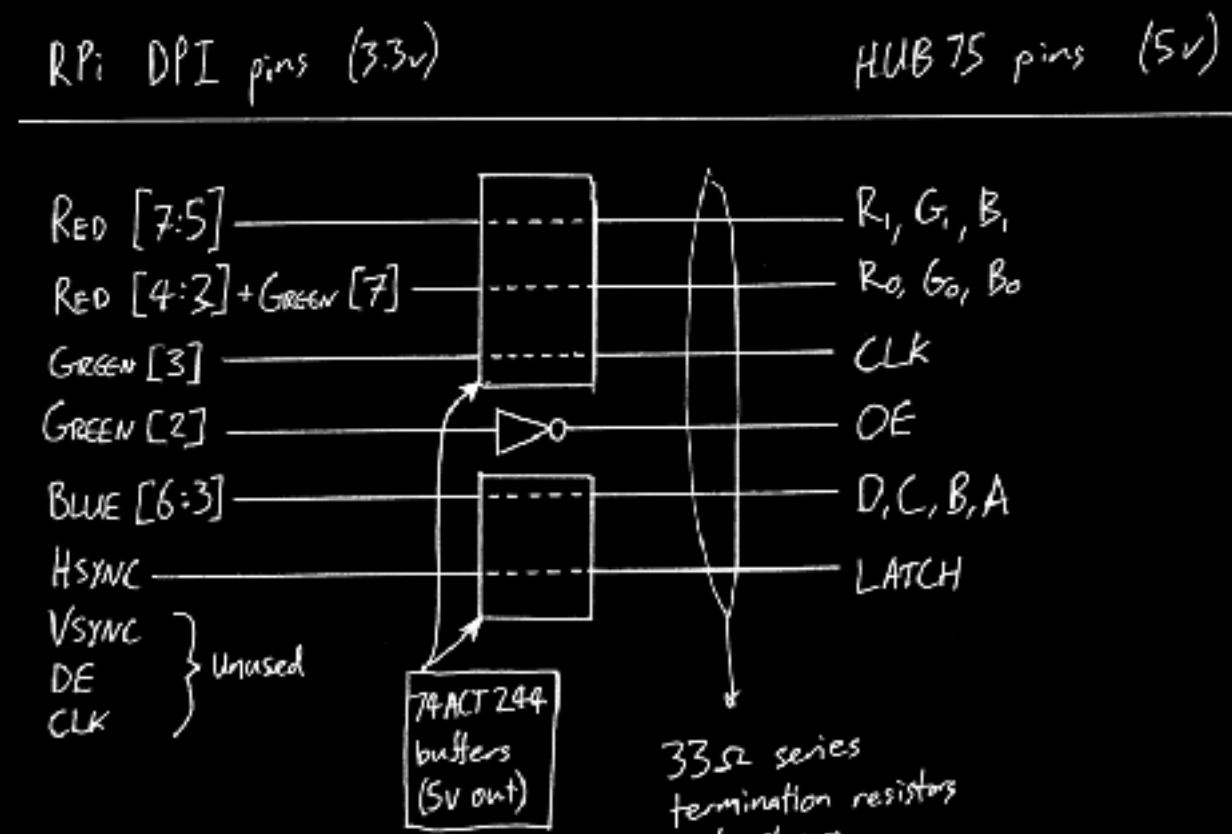
Misusing video outputs

- You may have seen people using VGA for analog out:
  - Tempest for Eliza: AM radio transmitter
  - Fabrice Bellard's DVB-T transmitter
  - osmo-fl2k: Using FL2000 USB dongle as SDR transmitter
- Haven't found any projects using *digital* video out for other things



LA experiments.
DPI pretty flexible - program resolution, sync width etc., variable bit depth
Just dump stuff into **/dev/fb0 framebuffer device**
Test program to try different FB resolutions/**POC**

Wiring HUB75 to DPI

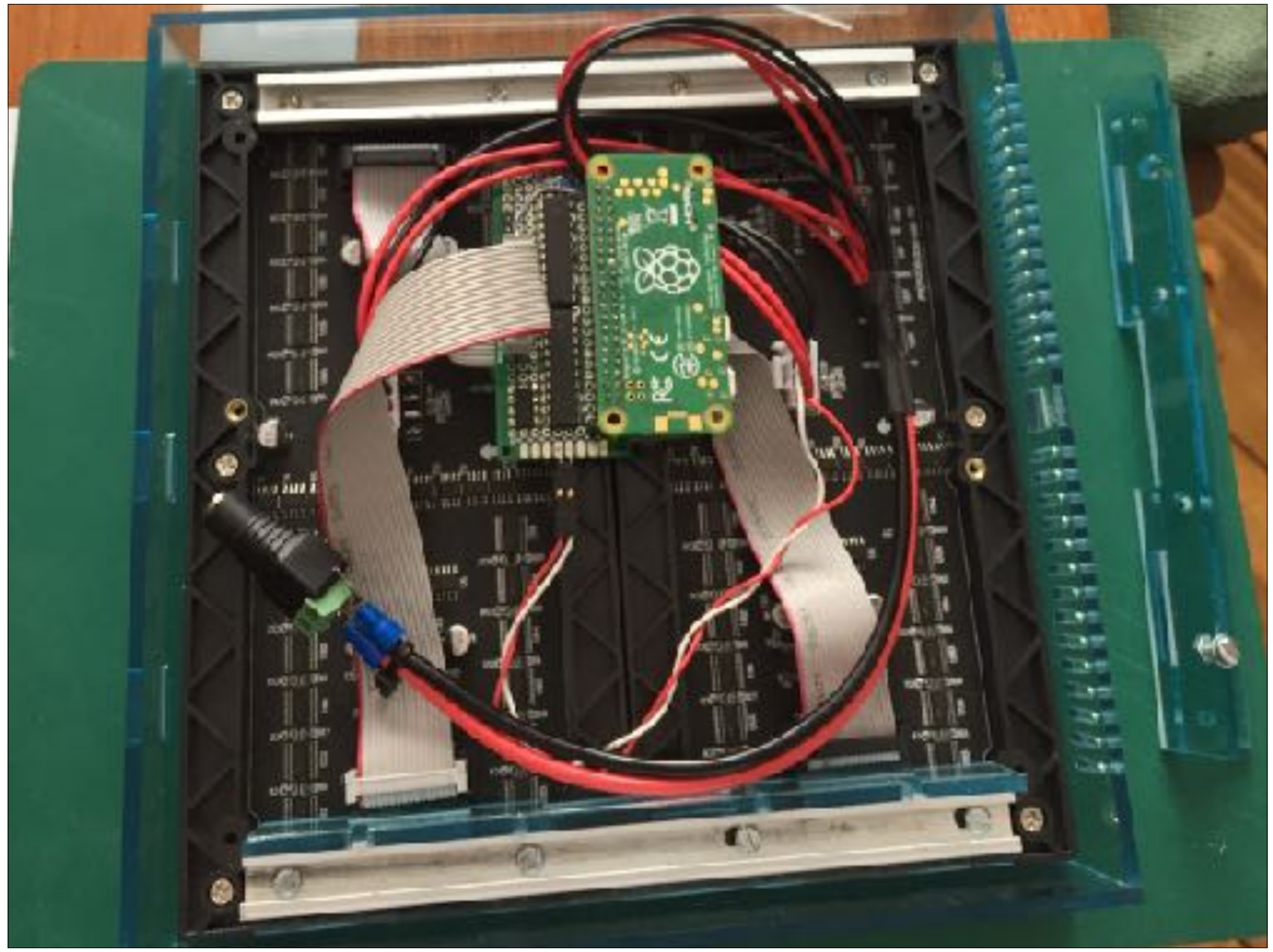


Budget runs out for fancy diagrams. Prototype:

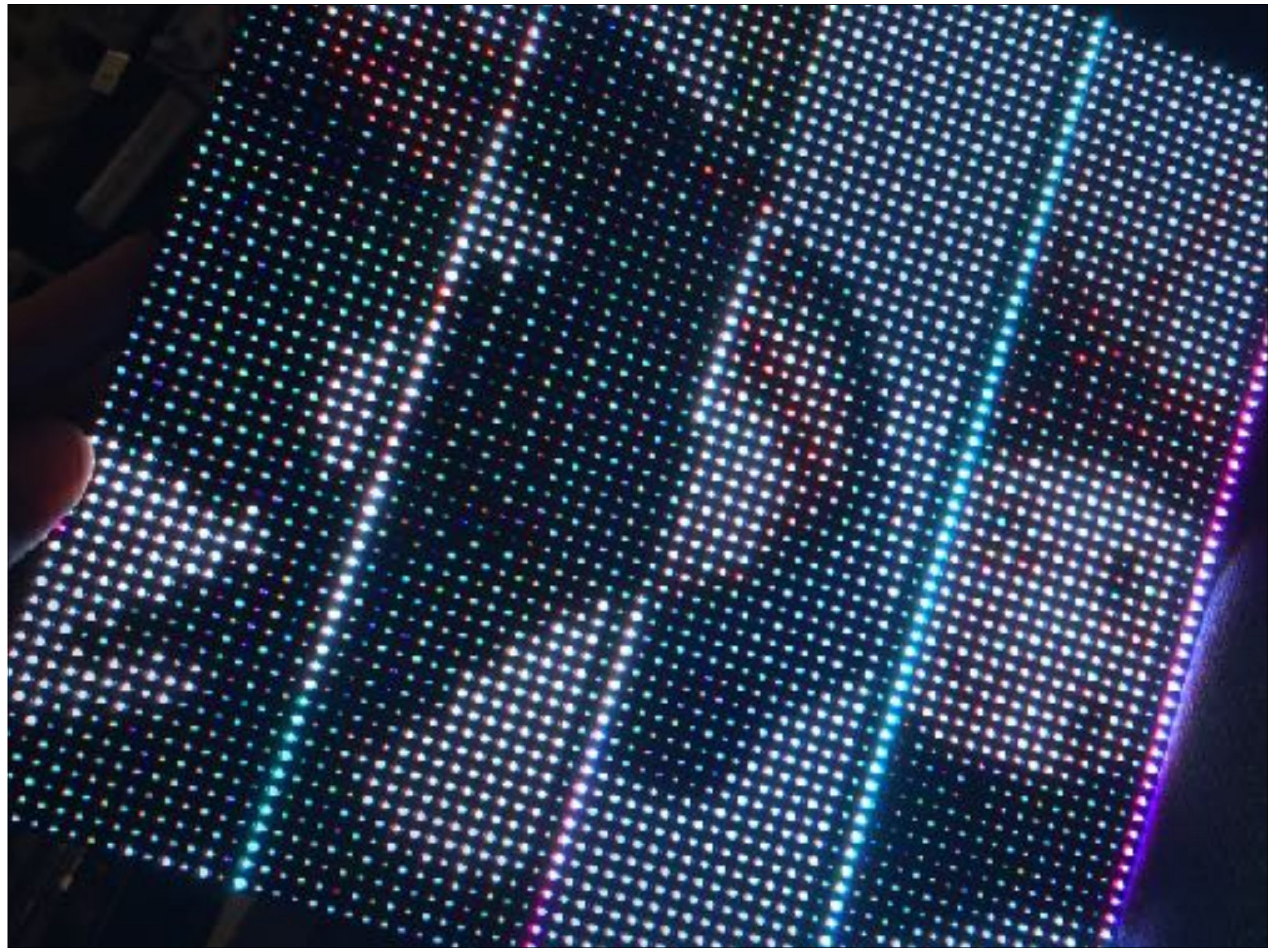
- 4 bits -> ABCD row select 1/16, 6 bits for RGB0/RGB1, 1 bit for CLK
- 1 bit for OE — modulates brightness of current row
- HSYNC -> LATCH

Termination very important — wires long, fast signals

So I solder that up.



Little bit of protoboard — 74’245 to drive 5V
Extremely cheap! Interface costs less than 1 beer
Can see the acrylic case I started to build for it…
Glueing acrylic and trying to get it NEAT, OMG painful



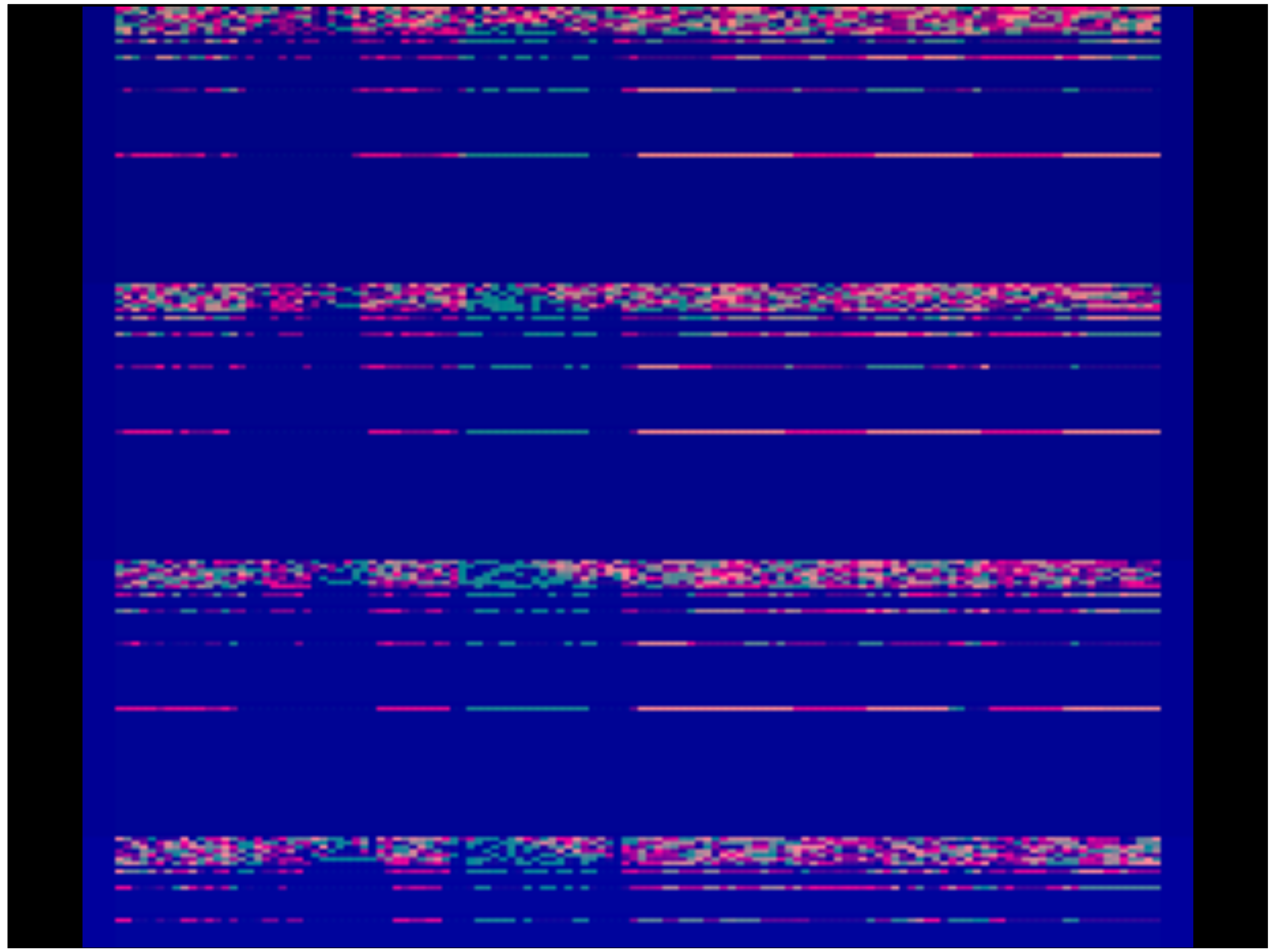
ANNND…   OK, more debug needed. But **concept good**:

Clocks one row worth of data every video line; HSYNC at end of line then latches that to drive LEDs.

For a given **row, spends 68 lines** driving different intensity levels of same pixels — BCM

Then, select **new row** — same again

Then, after **16 rows, frame done**.



What the Pi’s framebuffer actually looks like (e.g. if you connected HDMI)
First **four of 16** rows
128 clocks horizontally, for each driving RGB*2 into LED rows
Look at the **gap** — this is the timing for the intensity levels — higher bits in colour/intensity are left on for a longer time
BLUE background - dark to light (**ABCD row sel**)

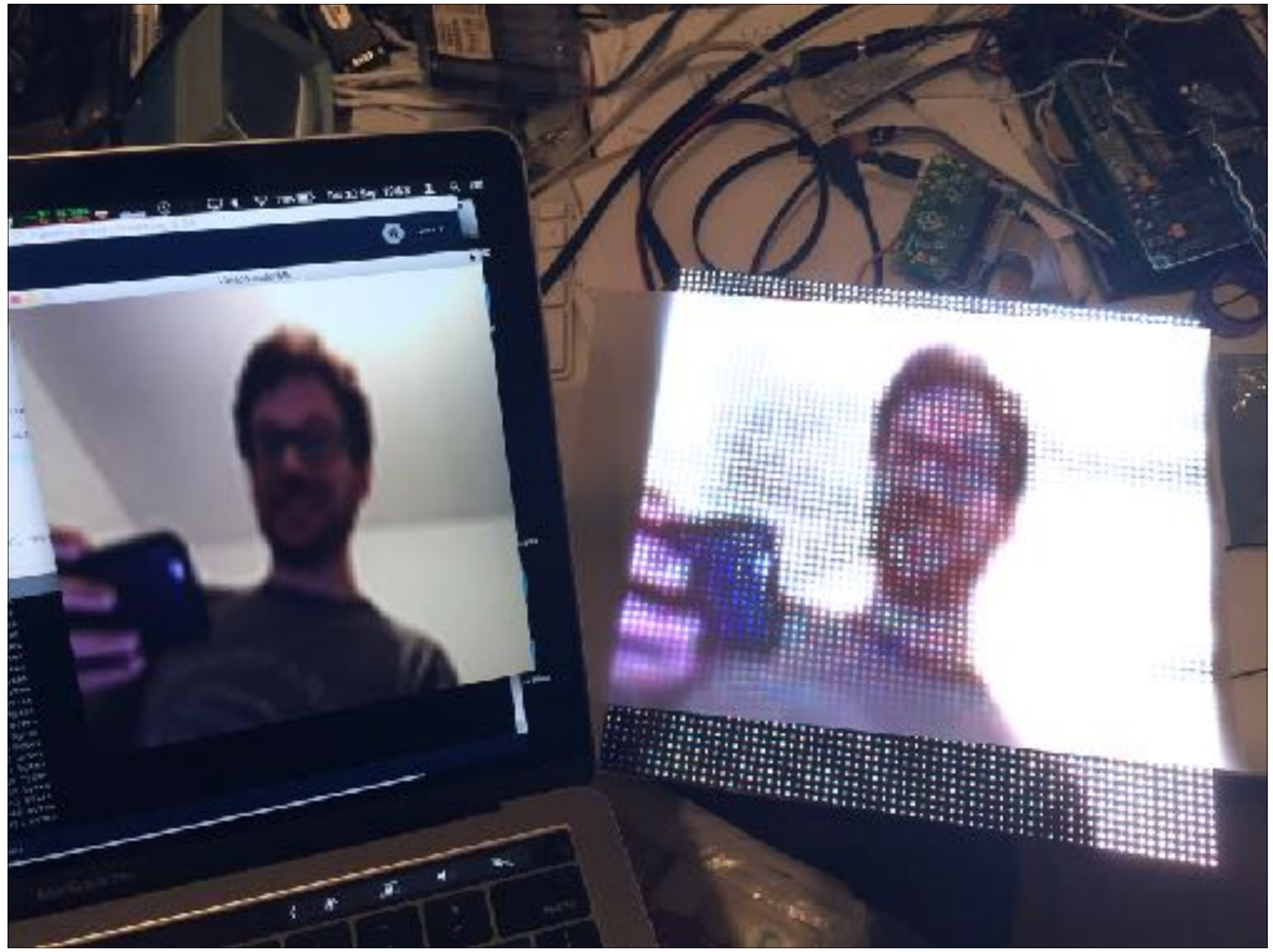
Works really well — **guaranteed no flickering**

Simple **Linux library** that takes 64x64 32BPP RGB and does **massive bit-shifting**, writing to /dev/fb0 to send data out line by line.

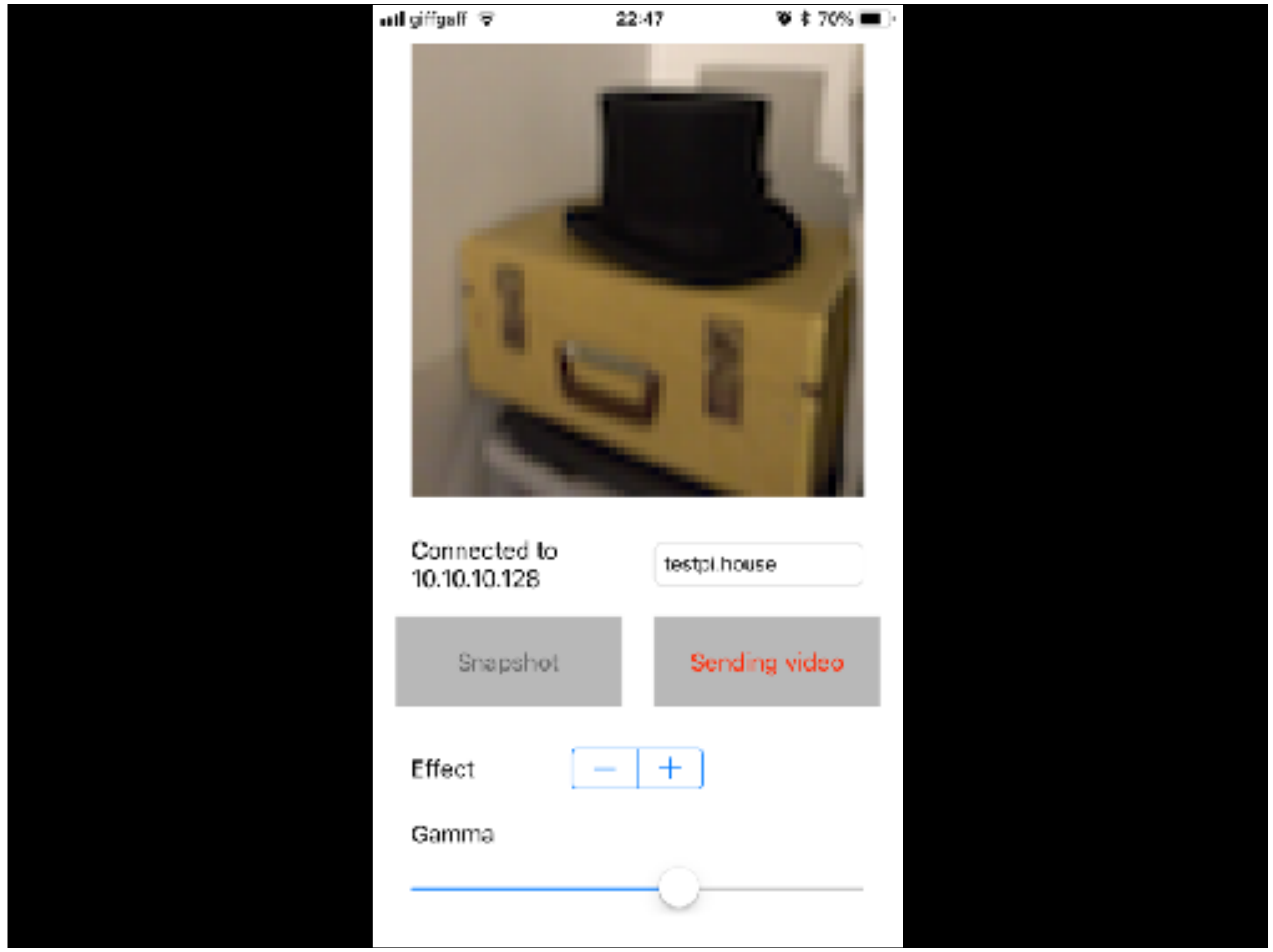
App simply sees a **flat RGB** frame-buffer

Plasma, fractal animations

The **realtime Julia set fractal** was pretty smooth on the STM32, but super smooth on 1GHz CPU
I haven’t hacked **DOOM** to use it yet :)
177Hz, 11BPC/33BPP — **entirely flicker free** no matter how heavily loaded network/CPU is!
0% CPU overhead



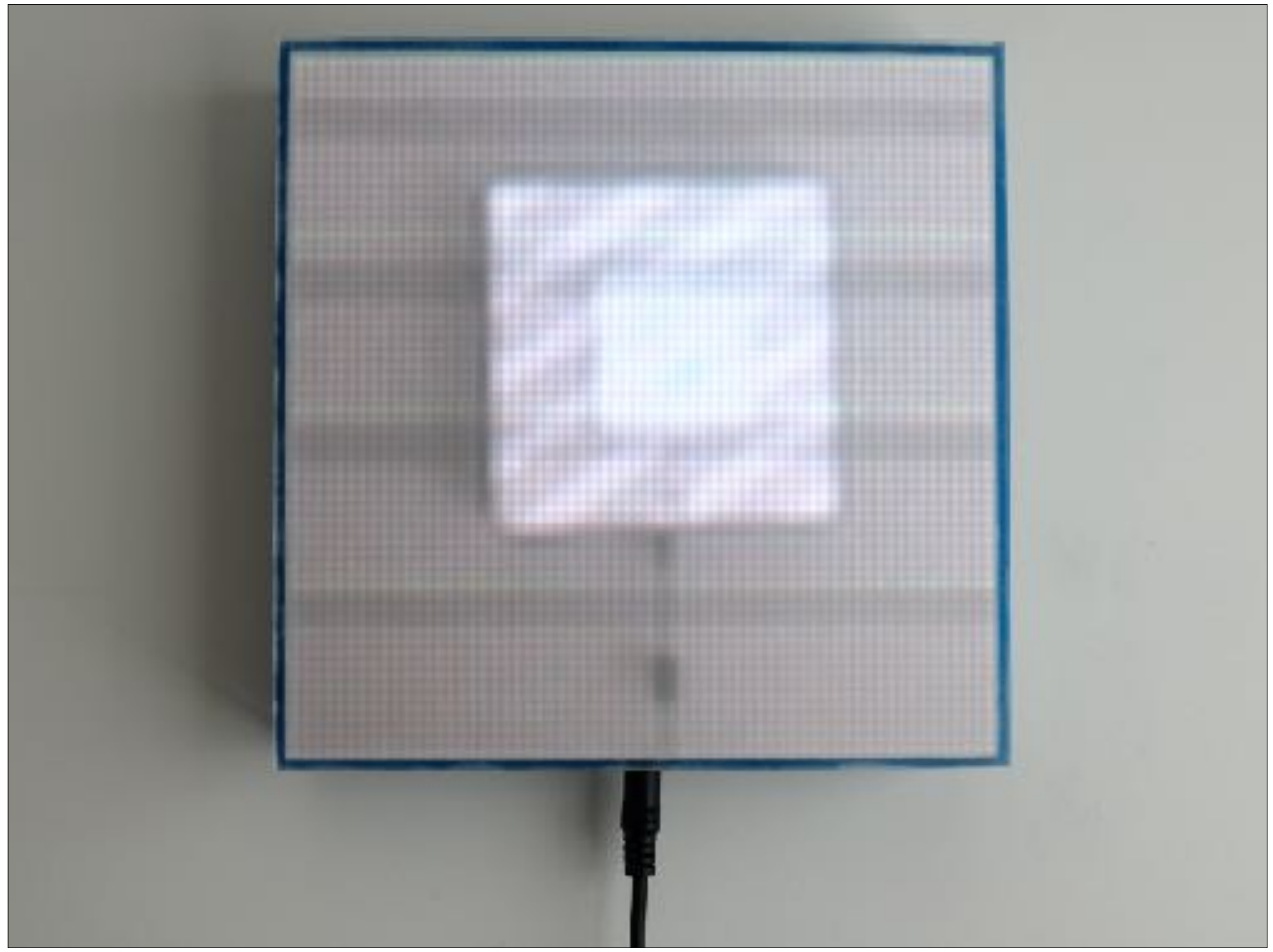
I got my **video streaming** wish…



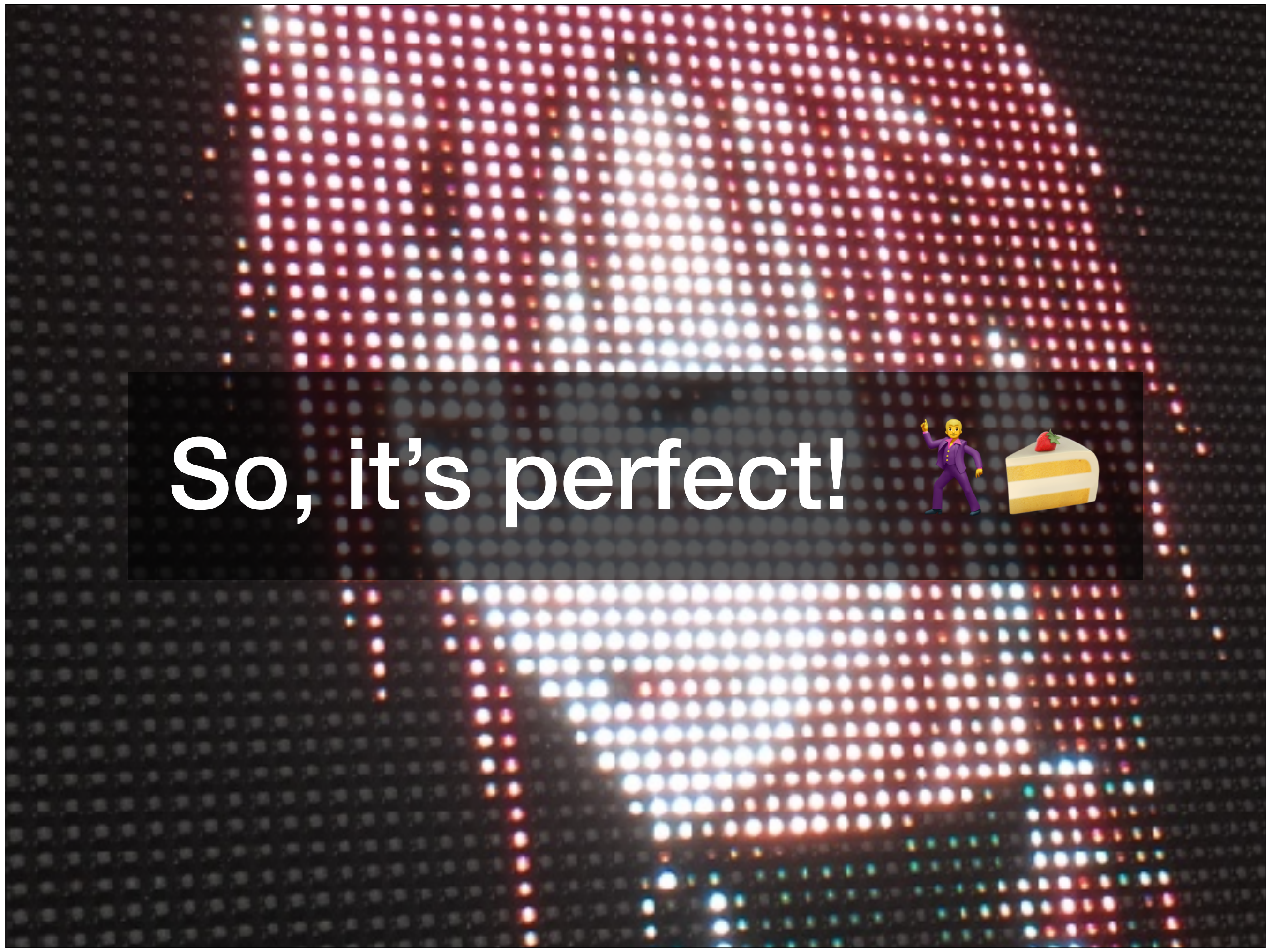
…wrote a little **app for my phone** to send image snapshots or **stream video**

Pointless fun thing for the wall

Also resolution so low that software **MPEG2 player** is only a few % CPU ;-)



Recursive…
On the software side, I mentioned **0% CPU** to hold the image flicker free
To **change the image** there's the **massive bit-shifting exercise** - low but not free, **1.5ms**: at 60changes/sec it's 9% CPU
-> Not optimised — expect can improve


So, it’s perfect! 🕺🍰

Not quite — some quirks

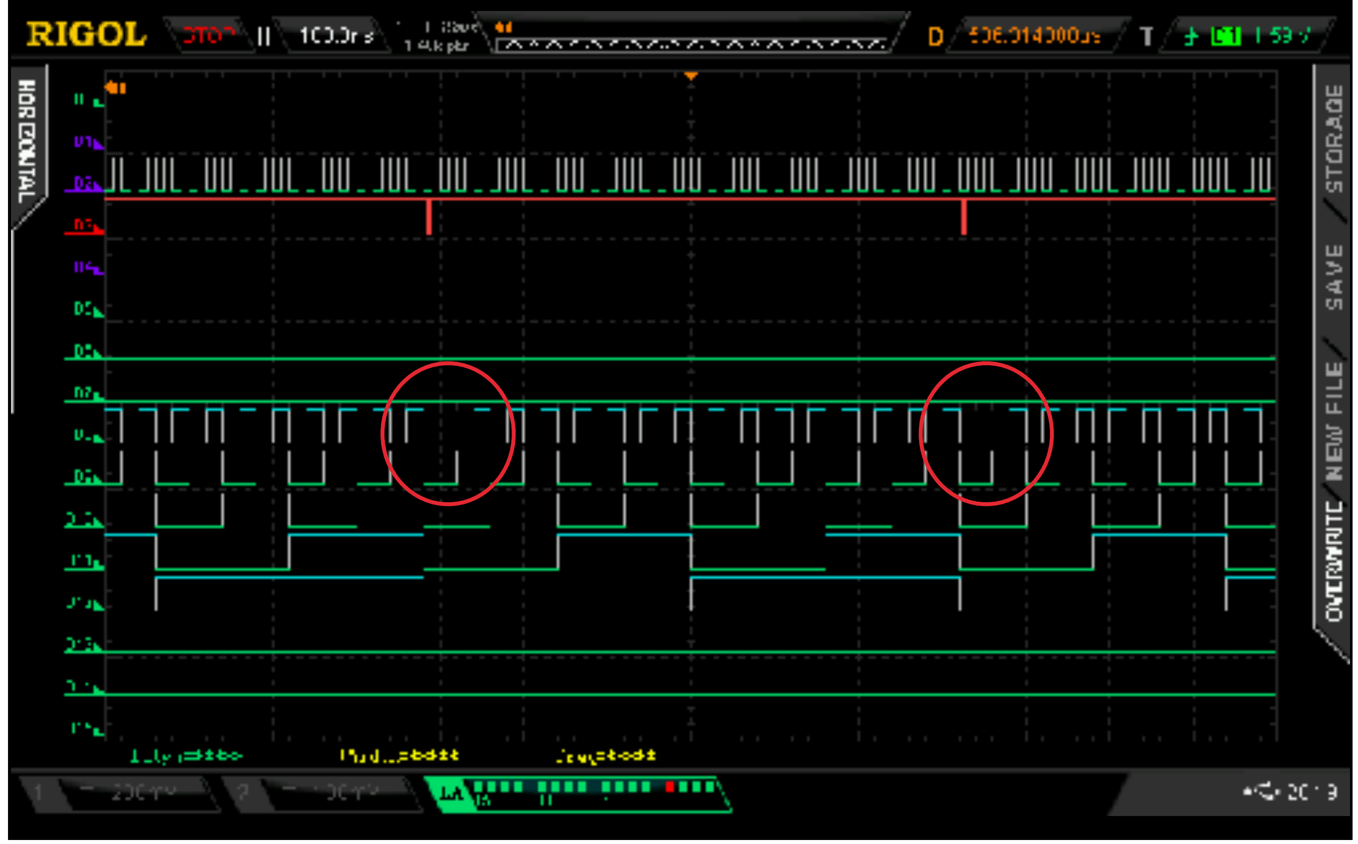
- 24 bit DPI uses *all* GPIOs
  - DeviceTree configures per-pin multiplexing — you don't *have* to use them all
- I used a 16BPP screen mode — I only needed a few bits output
  - But where's my LSB of Blue?

Use custom DT to use UART — max 22 bits

In 16BPP noticed **odd/intermittent disappearance** of b0 in SOME situations

Can re-create by drawing a gradient ramp in blue (R=G=0)

Lost blue bit



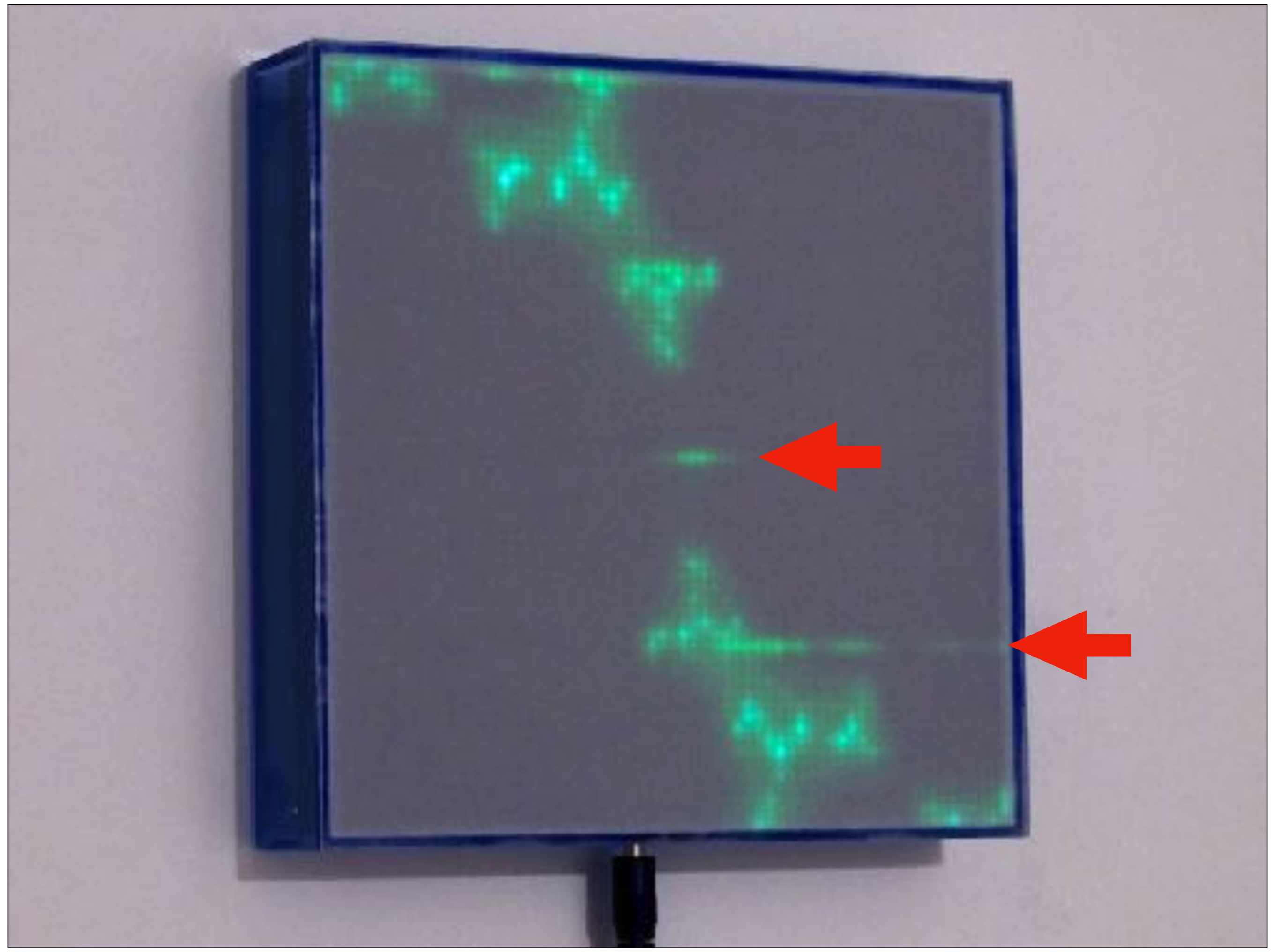
LA plot showing **intensity ramp in blue**
Binary counting up 5 bits of blue
See the gaps?
B=0b00000 and 0b00001 are both output as zero

Lost blue bit

- My theory: display controller doing post-framebuffer dithering/colour correction in 16BPP modes
- This does not occur in 32BPP modes — I recommend just using 32BPP!

Just use 32BPP mode.

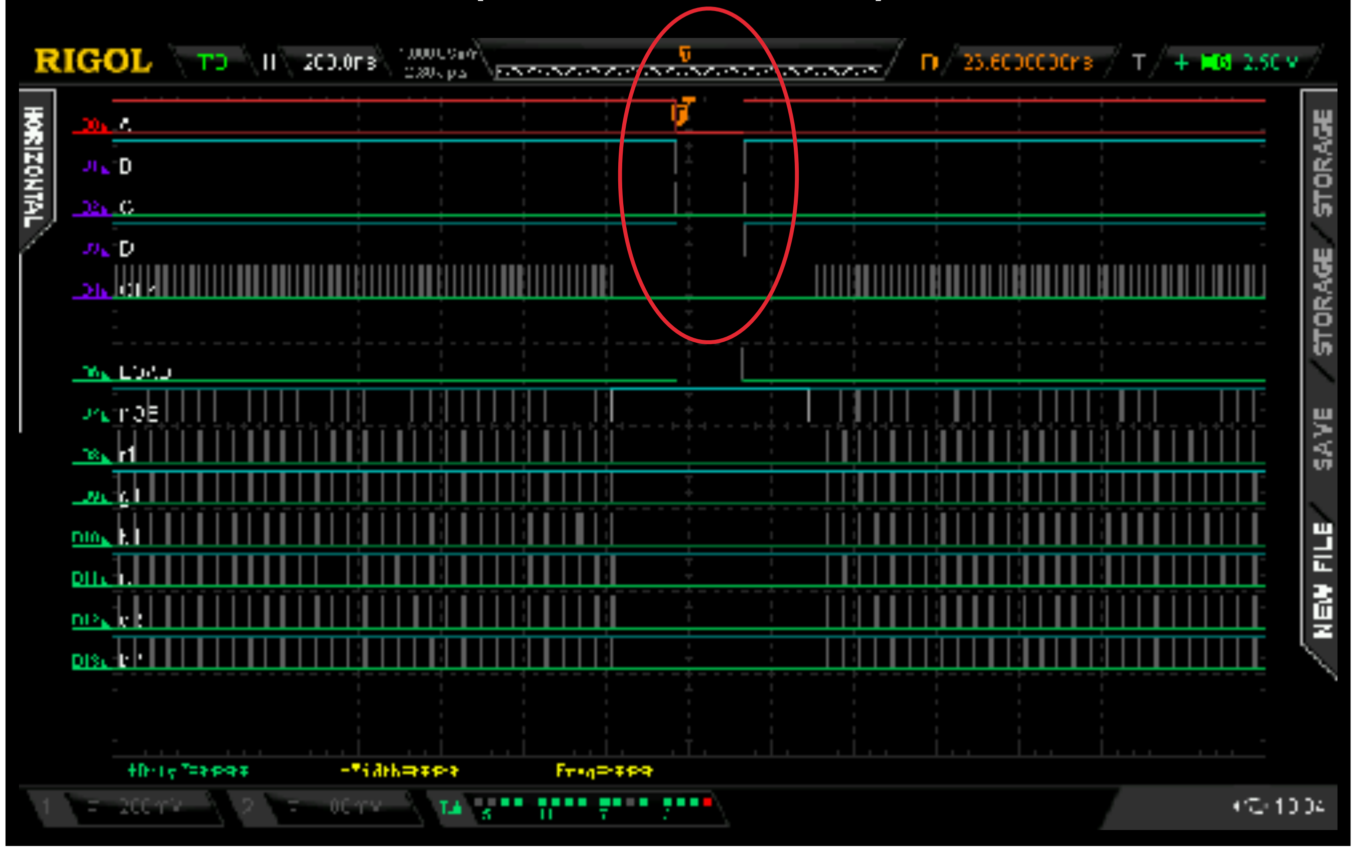
OK anything else?



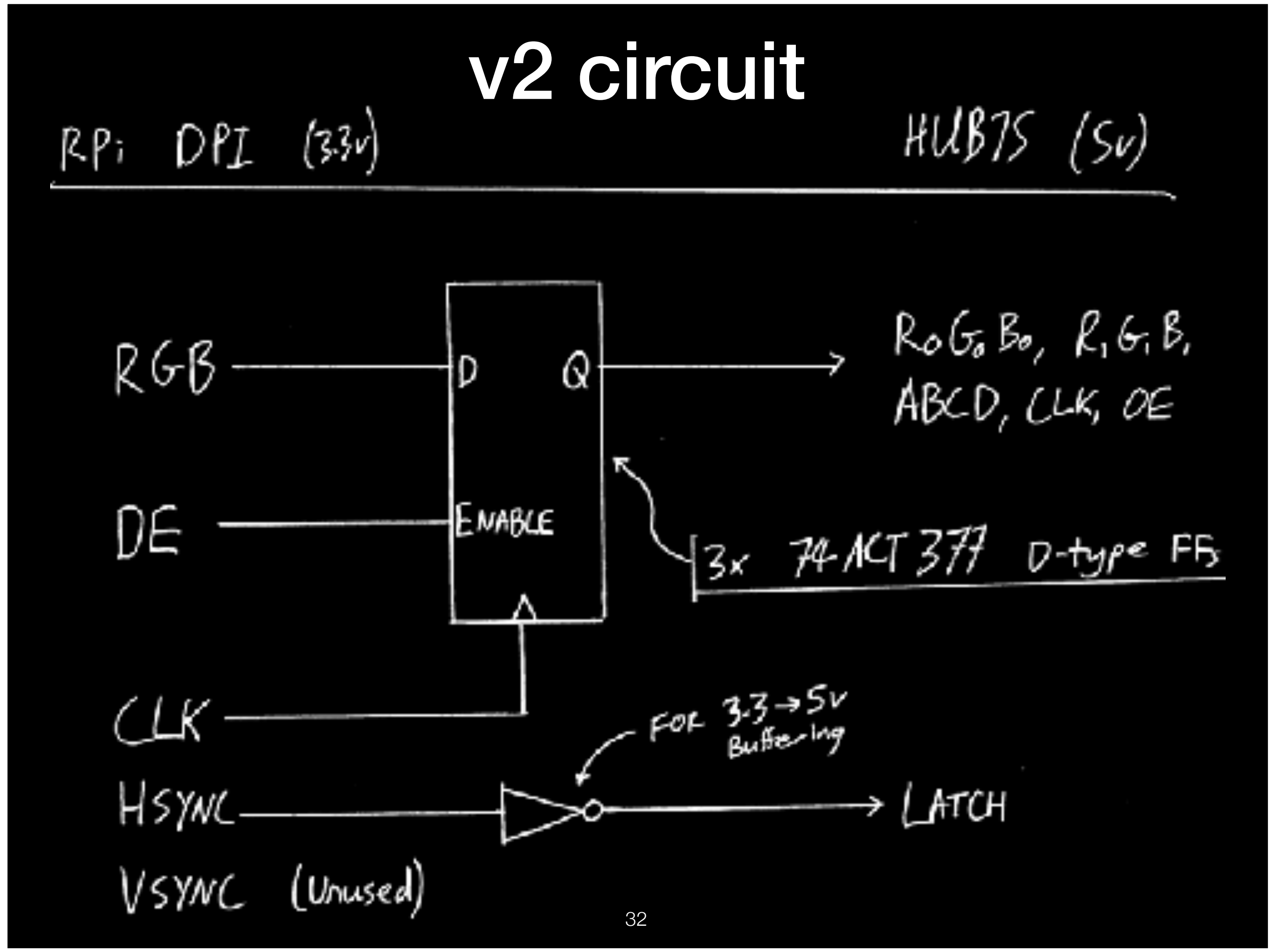
Another quirk: These lines aren't supposed to have that **shadowing** of other stuff lower down the image

These lines correspond to row 1 out of 16 for each segment of the panel (4)

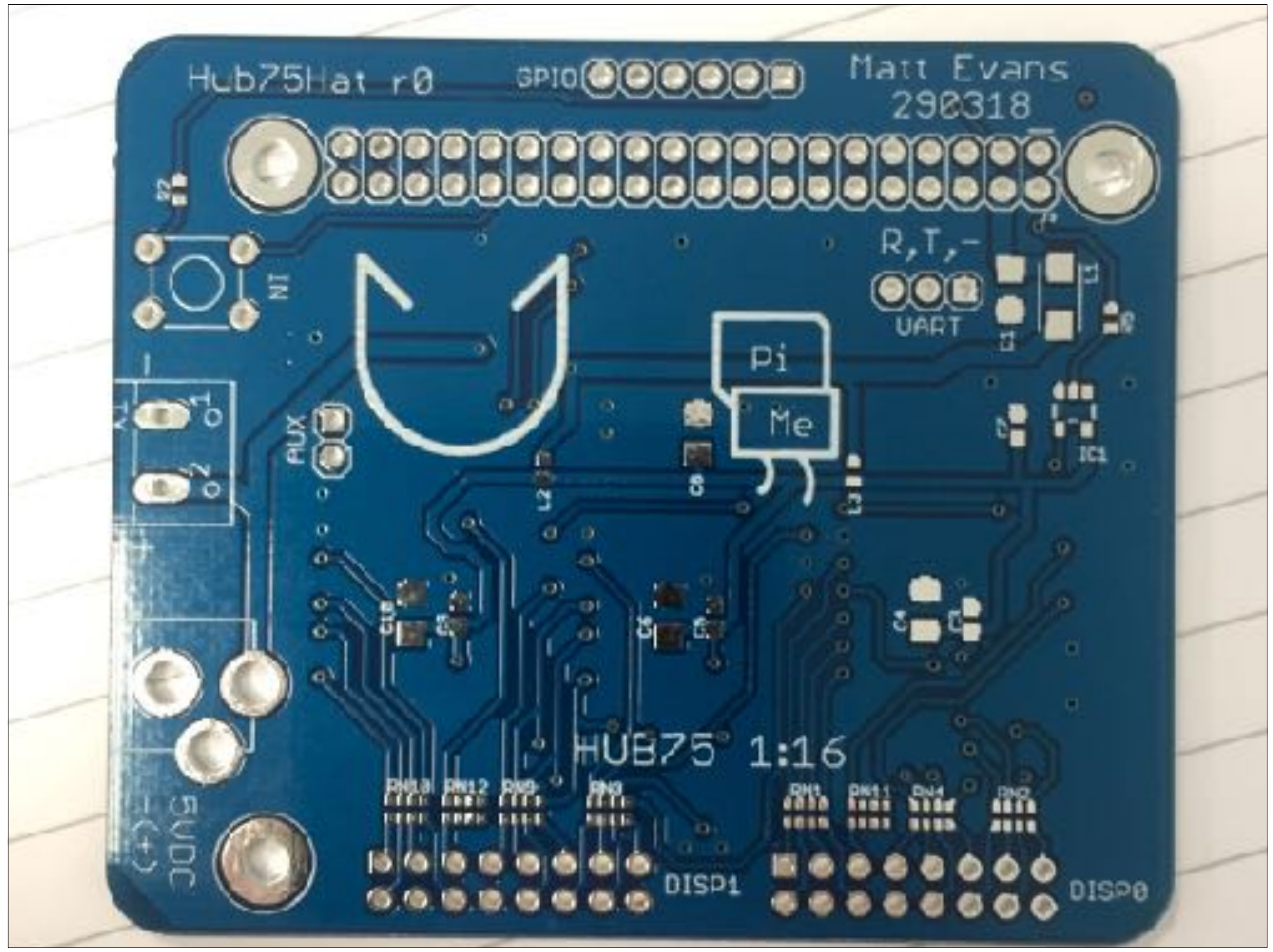
Signals not (all) held across Hsync
(but some are…)



LA plot: You can see the **ABCD/row** lines all going to **0 across HSYNC** (LOAD). (ALSO **B0** loss)
That means that **first row is getting selected across HSYNC** — displaying whatever was latched
Remember OE? Improved this by **de-asserting OE** before HSYNC
Wait time isn't long enough — more padding bad



Decided to take a **different approach**
Added **flip-flips** — latch output pixel
Hold it across HSYNC (so current row held) — **predictable**
Buffer to 5V. Still **cheap** interface — <1 beer



WORKS GREAT! No shadowing.

Enabling DPI

- OK Matt, DPI sounds *amazing*, how do I use it?
- https://www.raspberrypi.org/documentation/hardware/raspberrypi/dpi/README.md
- Enabled through /boot/config.txt
- Then, just write the display framebuffer as usual

Using **SDL** or similar makes it easier to gain **control of the framebuffer**

- Disables the **cursor** and **debug messages** trampling on your image

```
dtoverlay=dpi18
enable_dpi_lcd=1
display_default_lcd=1
framebuffer_depth=16
# HS/VS phase (0) polarity (0) control (7), nCLK/DE (2),
# RGB order(1), 565 mode 3 (3):
dpi_output_format=0x007213
# Custom timings
dpi_group=2
dpi_mode=87
# hdmi_timings=<h_active_pixels> <h_sync_polarity> <h_front_porch>
#  <h_sync_pulse> <h_back_porch>
#  <v_active_lines> <v_sync_polarity> <v_front_porch> <v_sync_pulse>
#  <v_back_porch> <v_sync_offset_a> <v_sync_offset_b>
#  <pixel_rep> <frame_rate> <interlaced> <pixel_freq> <aspect_ratio>
hdmi_timings=272 0 0 8 0  1103 0 1 1 100 0 0  0 170 0 60000000 1
```


Example working config

If the Pi doesn't like your configuration, it will silently:

— choose one it prefers

— **fail to boot**

— boot but keep **DPI disabled**

BCM2835 DPI capabilities

Pixel clock	min 32MHz, 105MHz tested OK Maximum ~150MHz?
X pixels	min 8 max 1920
Y pixels	min 8 max 1280
Sync widths	min 1 pixel for HS min 1 line for VS
Misc	Didn't check max sync widths or lowest frame rate (theoretically 14Hz)

Remember don't NEED PCLK — can clock external stuff from a pixel colour bit
My scope isn't high enough BW to see how good o/p @140MHz is, but it enabled
FB dimensions; max/min/alignment — **fit your data** around this
SYNC— :(**Can't get completely unbroken** stream of data out

Many other computers support similar LCD video output

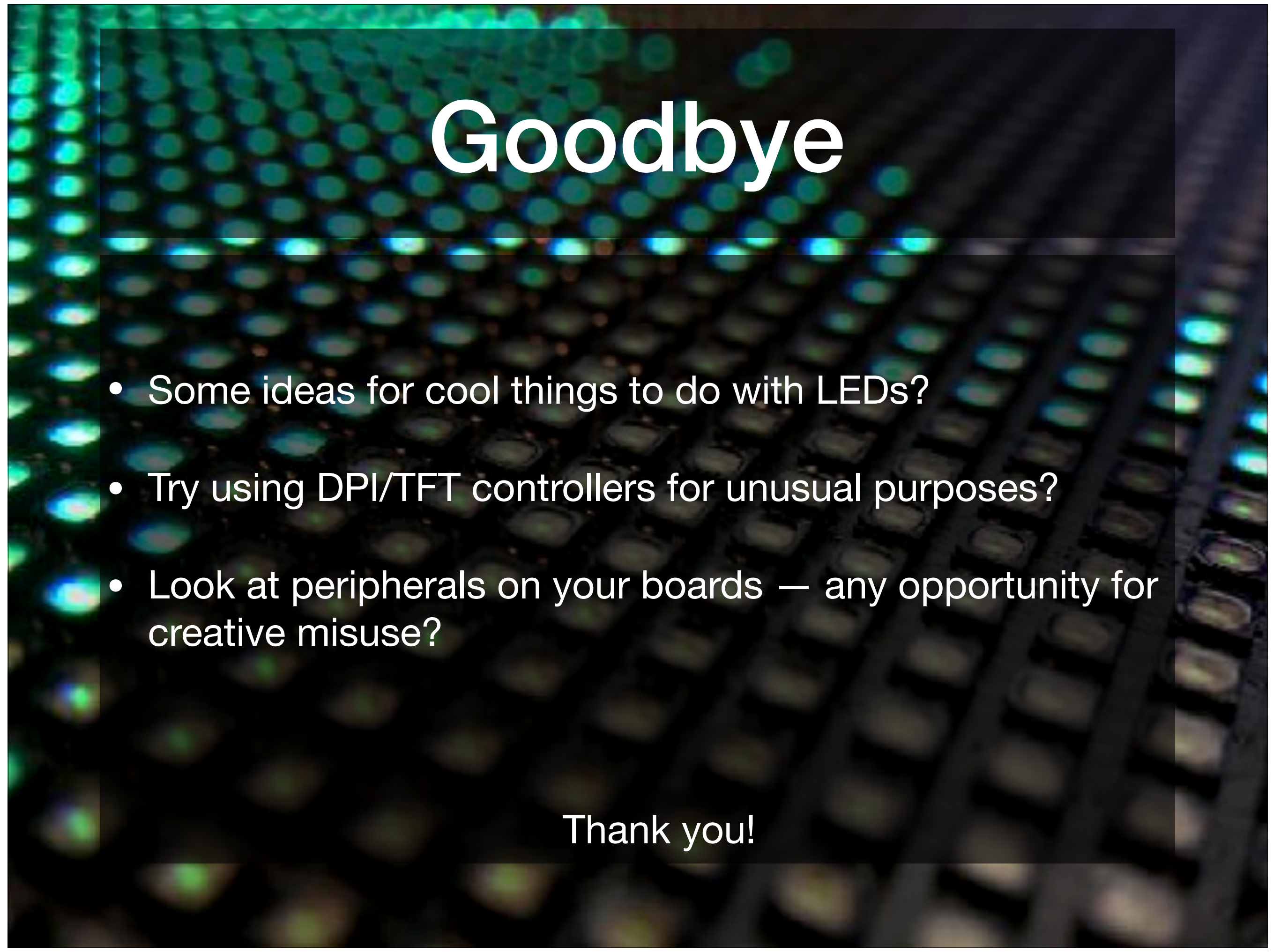
- Common for SBCs to have parallel output from LCD controller!
  - Beagleboard, various cheap Allwinner/sunxi boards
- I like this technique because:
  - Often faster than GPIO
  - Realtime, zero CPU overhead
  - Much easier to get started/debug than using DMA controllers
  - Can do this from **userspace**, or even python
- Not as nice as a Beaglebone PRU ;-) (But $$$/complicated!)

I used a Pi here but this **technique** applies to many other machines.

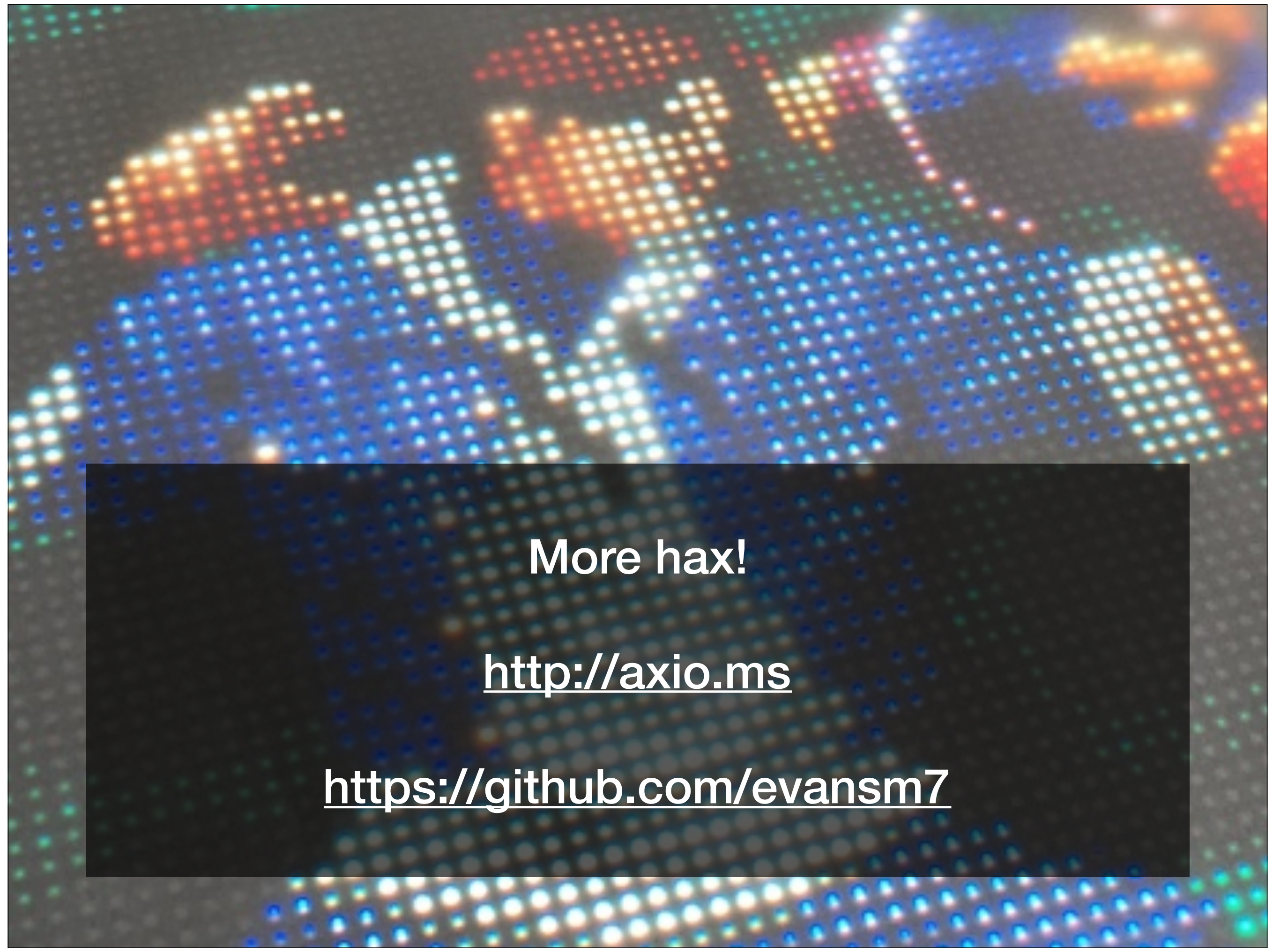
Aside from VGA and LCDs and LEDs, what is it good for?

- Supply data to FPGA/CPLD — pattern/signal generator?
- Motors — 24 servos!
  - Steppers?
- Drive 20 SPI LCDs at once
- Or 24 strings of WS2812s — 24x1024 at >30fps!
  - Only 1.2kW & about €2000 🧐

Wanted to do **sig gen** — but **hard** w/o **unbroken stream** of data (VS/HS gaps) — FPGA **retime**
Stepper motors might work if they can deal with the VS/HS gaps?
Have been playing with some **very cheap 2" TFT** modules (0.5 beer) — SPI stream, could drive **20 of these** in parallel from one Pi. 20 tiny displays, cool!
… **all from userspace**, all 0% CPU overhead



OK, tour has come to an end. Cheers, seeya. :D


More hax!
http://axio.ms
https://github.com/evansm7