
Okay, restarting April 8th, 2025.

Going to try to vibecode it.

Reference docs

Redone schematics

https://github.com/mishimasensei/macse30mlb?tab=readme-ov-file

----

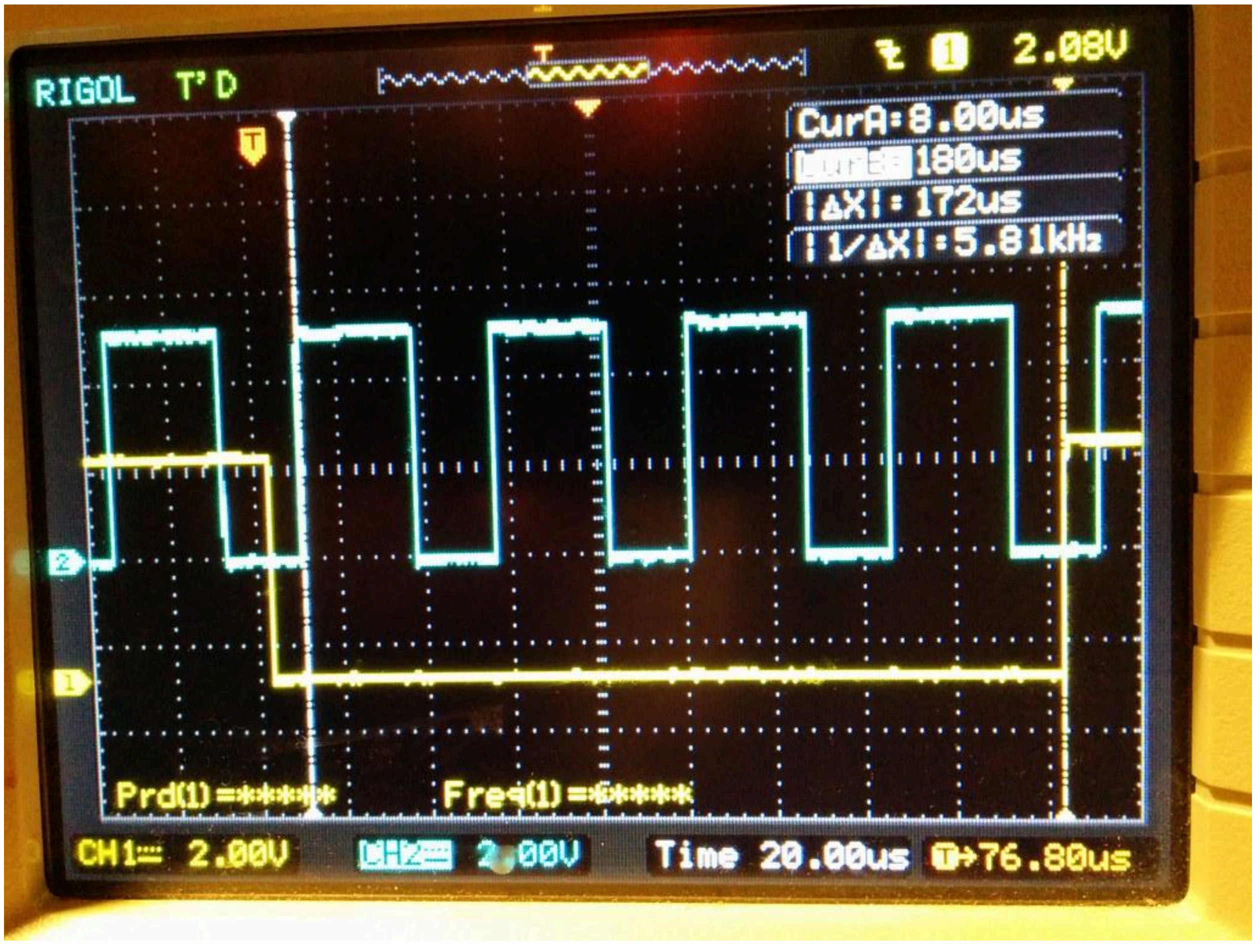
Timing notes from here: https://trmm.net/Mac-SE_video/#macintosh-se-video-interface

Pin	Color	Function
1	Black	GND
2	Black	GND
3	Black	GND
4	Black	GND
5	Black	GND
6	Blue	-5V DC
7	Green	-12V DC

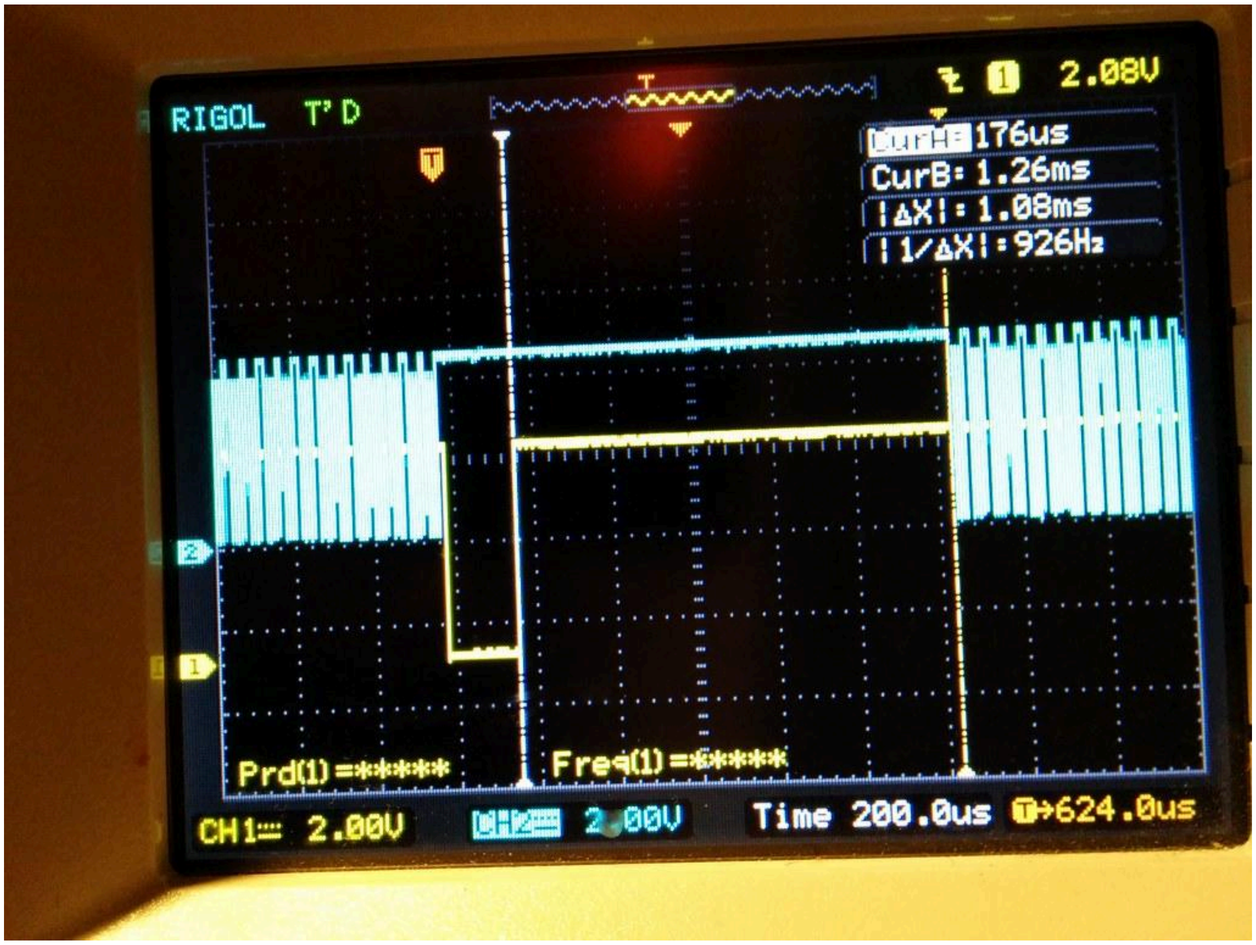
8	Black	GND
9	White	VIDEO (starts 11.2 usec after falling edge of HSYNC, lasts about 33 usec, first data line is 1.26ms after falling edge of VSYNC)
10	Purple	HSYNC (square wave, falling edge triggered, 45 usec per line)
11	Gray	VSYNC (square wave, falling edge triggered, 60.10 Hz / 16.64 ms per frame, 180 usec low)
12	Orange	+5V DC
13	Orange	+5V DC
14	Yellow	+12V DC

Since the pins immediately go through NAND gates (74LS78), it is not necessary to level shift from the 3V output from the BBB to the 5V used by the Mac. The powersupply for the monitor produces a clean 5V that can run the BBB once things are setup. Only need three pins from the PRU (video, hsync and vsync), leaving plenty for an eventual ADB port.

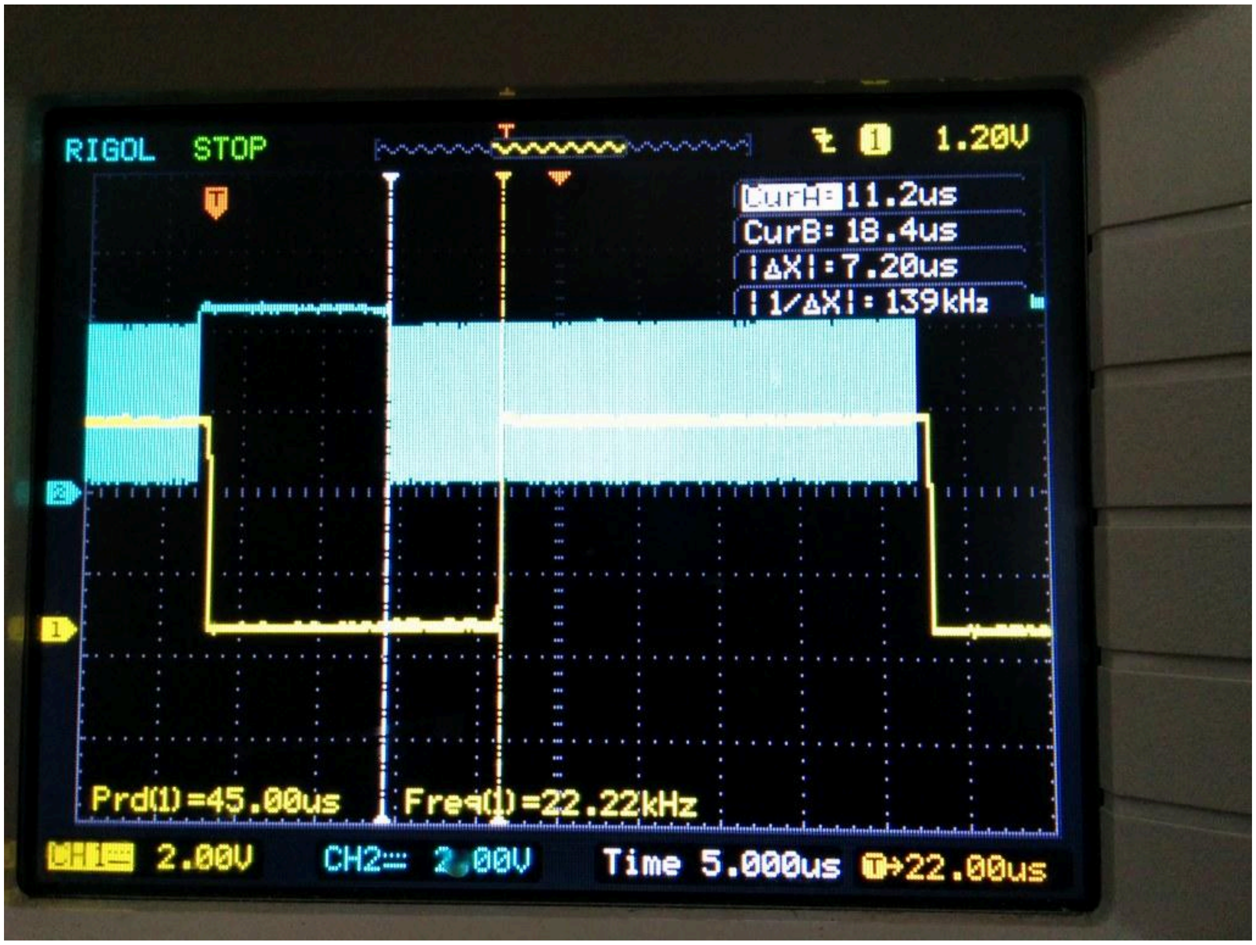
Timing



Falling edge of VSYNC triggers vertical blanking and start of a new frame. VSYNC falling edge to rising edge: 180 usec. HSYNC pulses continue during vertical blanking, offset 8 usec from the falling edge of VSYNC. HSYNC pulse falling edge to falling-edge is about 45 usec.



Falling edge triggers vertical blanking and start of a new frame. VSYNC falling edge to rising edge: 180 usec. VIDEO pulses begin after 1.26 ms. HSYNC pulses continue during this time, too.



HSYNC is falling edge triggered, 45 usec between edges. Rising edge occurs 18.4 usec after falling edge. Video data begins 11.2 usec after falling edge. Roughly 16 MHz dotclock (512 pulses in 32 usec). The "hsync backporch" is negative, which doesn't seem possible with BeagleBone's LVDS framer.

Mac Classic CRT timing notes from here

https://nerdhut.de/2016/06/26/macintosh-classic-crt-1/

Introduction

This article will explain how the timing of the Macintosh Classic CRT works and how I tried (and failed) to interface it with the Raspberry Pi, and how I successfully interfaced it with the BeagleBone Black's PRU.

How does the CRT display an Image?

I will not explain how the CRT works in theory, other people have done that before. But I want to write about the signals needed to compose an image on the CRT-display.

The screen basically needs three signals: HSYNC, VSYNC and DATA (It might remind you of VGA). HSYNC stands for horizontal synchronization, VSYNC is for vertical sync, DATA sends the video pixels to the display. Since the Macintosh Classic display is monochrome (black/white), there is only one DATA line. If it had more colors, there could be more separate lines (like one for red, one for blue and one line for green).

HSYNC triggers a new line on the display. At every falling edge, the CRT's circuit positions the screen's electron beam to the beginning of the next line. The signal has a frequency of roughly 22.25kHz and it looks like this:

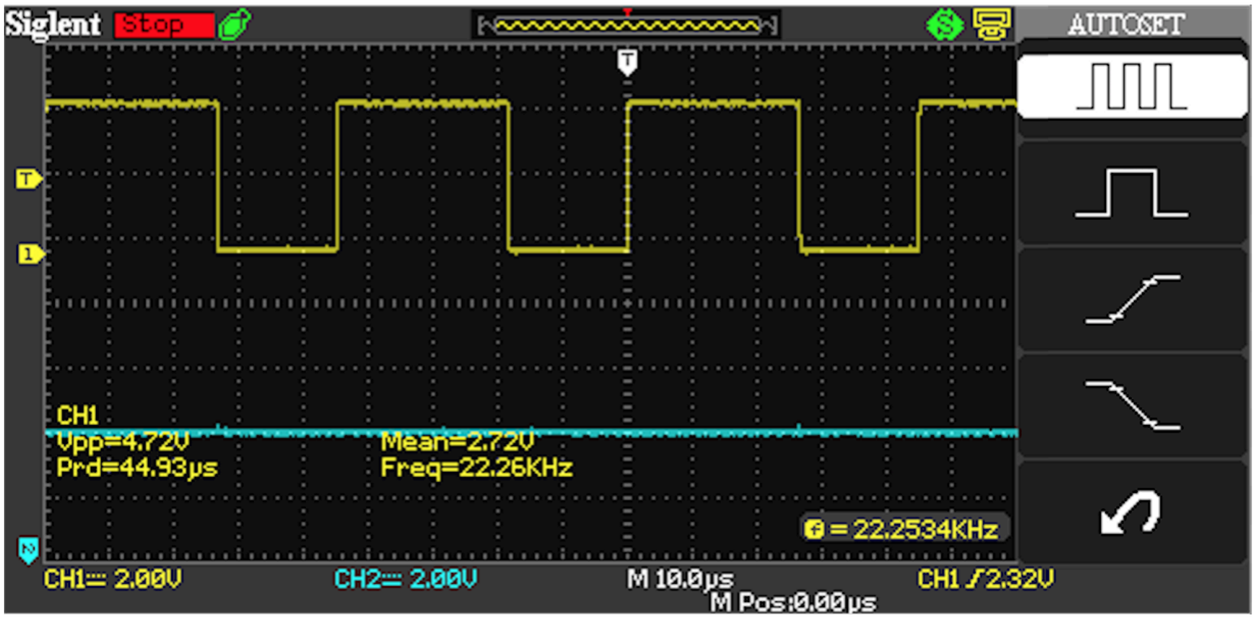


Fig. 1: The HSYNC signal for the CRT

Each time the HSYNC signal gets low, the screen gets ready to display a new line of data. You can see that the period is about 45µs for one complete frame. But you can also see, that the signal is HIGH 59% of the time and LOW for 41%.

The VSYNC signal triggers a new frame on the display. It has a frequency of about 60.15Hz, which is also referred to the refresh rate of the screen. This signal was a bit harder to create because it had to be synchronized with the HSYNC signal. In my application, the VSYNC signal occurs after 342 HSYNC edges. The HSYNC pulses continue during the VSYNC, but the video data doesn't:

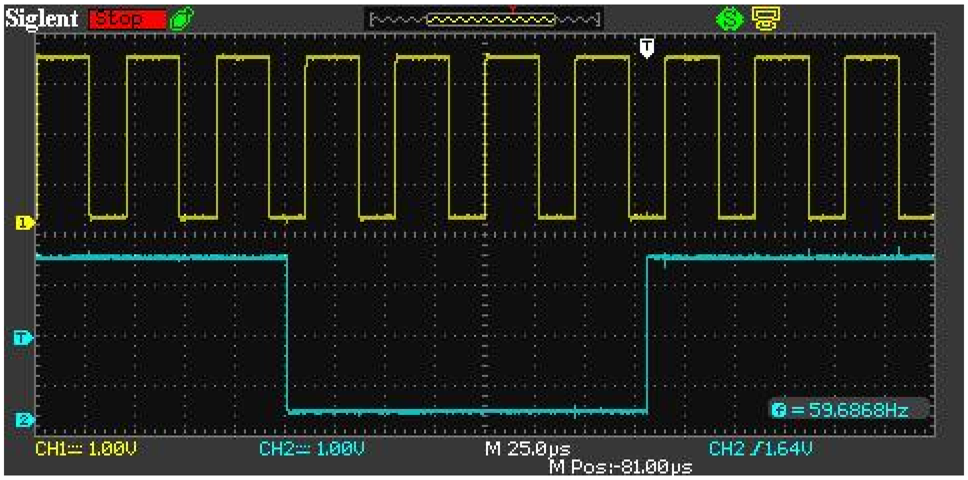


Fig. 2: HSYNC pulses (ch1, yellow) and VSYNC pulse (ch2, cyan)

You can clearly see that the HSYNC pulses continue while the VSYNC line get's low. VSYNC is also triggered on the falling edge.

The last signal is the DATA line. This was the trickiest bit and it was not possible to produce it with the Raspberry Pi. The problem is, that this signal is time critical. It always has to take exactly the same time to send this data to the CRT (and this is not possible to make with the raspberry pi by just writing code. However, you can construct slower frequencies exactly with the Pi).

The data is sent by bit banging the pixels to the CRT with a dot clock frequency of around 16MHz (15.6672 MHz to be precise). Bit banging means that you send out the state of a pixel (HIGH = white, LOW = black pixel) via the DATA line in an exact timing, one bit after another:

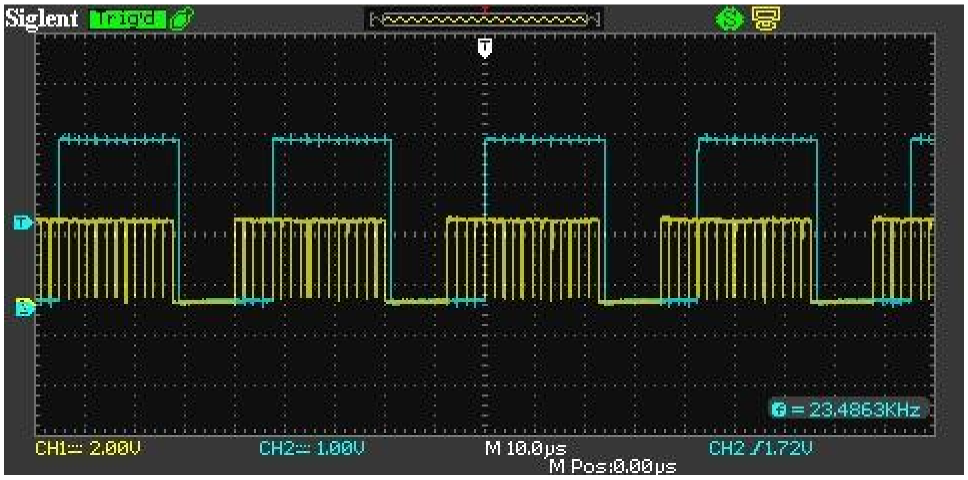


Fig. 3: You can see the HSYNC signal (ch2, cyan) start a new frame and the DATA (ch1, yellow)
As you can see in fig. 3, the DATA does not start exactly at the falling edge of the HSYNC signal, it starts (roughly) 10µs after the falling edge. This time is needed by the CRT to position the electron beam at the start of the next line. There is also a time needed after a VSYNC to reposition it to point at the first pixel's position on the screen. You can get a better understanding of the timings of these two signals by looking at fig. 4:

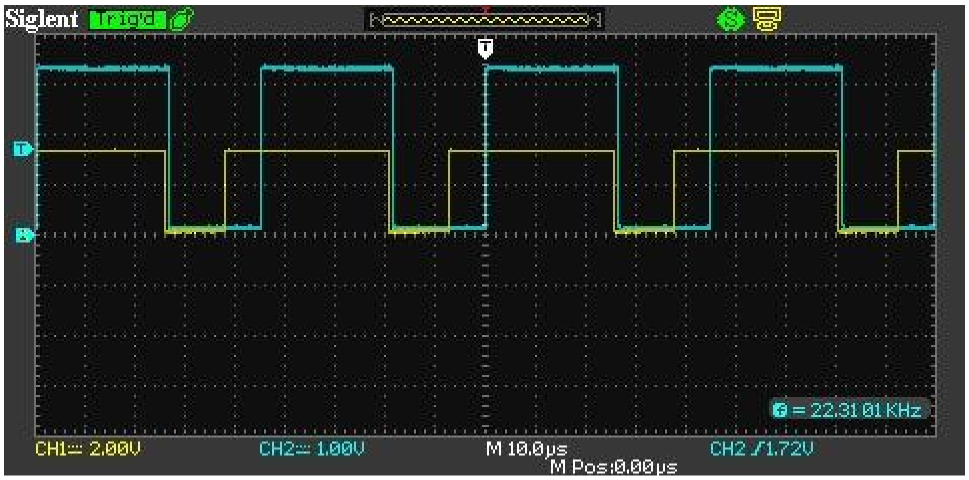


Fig 4: Sending completely white lines.

Note: The DATA line should be high while no pixels are sent to the CRT, fig. 3 and 4 show the DATA line getting low after the pixels were transmitted, to make the pulse duration visible.

This was basically everything that is needed to display an image on the CRT. The signals have to run synchronized and with exactly the frequencies they need to have. Otherwise, the image might get distorted.

Important numbers

If you want to rebuild this or a similar project, here are some important numbers for this particular CRT monitor:

`HSYNC-Frequency`

```
22.25kHz, 45µs period, 18.45µs low 59% PWM duty cycle
```

`VSYNC-Frequency (Refresh rate)`

```
60.15Hz, 16700µs period, 180µs low 99% PWM duty cycle
```

`**DATA**`

```
15.6672MHz, 512 pixels in roughly 32.8µs Signal has to be high
while no pixels are sent
```

If you can find a datasheet or a table with the exact timings for your display, you can get the information from that. Otherwise, you can try to measure the signals produced by the original hardware's video interface. Luckily, most older computers had such documents and you can easily find them on the interweb. Macintosh Classic Developer Note, Macintosh Classic II Developer Note.

Old Links and Notes:

https://nerdhut.de/2020/03/17/raspberry-pi-dpi-control-crt/#

https://www.hackster.io/news/revive-your-old-macintosh-classic-with-a-raspberry-pi-0e679f8bd1
68

Detailed article:

https://nerdhut.de/2020/03/17/raspberry-pi-dpi-control-crt/

CRT Timing notes
https://nerdhut.de/2016/06/26/macintosh-classic-crt-1/

11/28/2024, back at it.

Found an AirPlay server for rPi

https://github.com/FD-/RPiPlay/blob/master/README.md

--- couldn't get the Raspberry Pi working, so I decided to mess with the FPGA for a bit.

Got it working-ish, but couldn't get the clock hooked up--no activity on the pins. Simple examples work, but not the big project.

The internal clock is 12mhz, and I need 16.59mhz for the pixel clock. There's a PLL onboard, but I don't know how to get to it. Asked the Discord channel.
Waiting on a response.

https://discord.com/channels/1151201027765321768/1151207332227862781

May need to make an external clock. Adafruit has one.

https://www.adafruit.com/product/2045

Can't do it straight from an arduino.

https://arduino.stackexchange.com/questions/16698/arduino-constant-clock-output

Esp32 vnc client
https://www.reddit.com/r/arduino/s/SRpbpATAuS