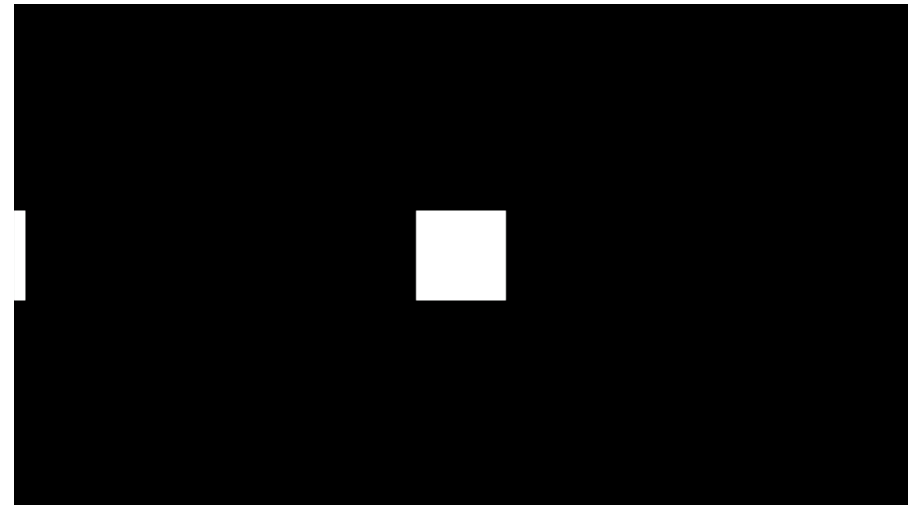


Video and Primitives

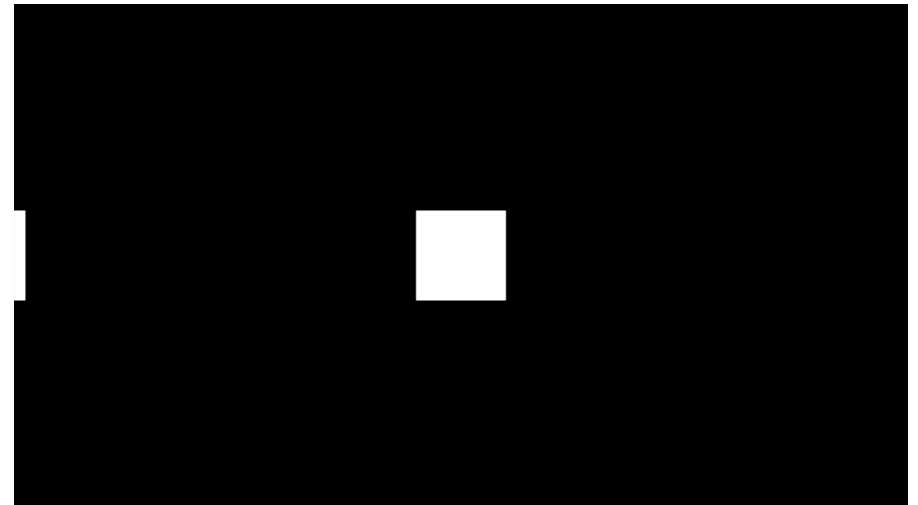
6.205

Fall 2026



Admin

- Week 03 was due last night
- Week 04 out after class: Video, babiiiiieeeeeee
- Tutorial session tomorrow
- Hudson River Trading Tech Talk and food after class today



Primitives

Primitives on the FPGA



- While the FPGA is a reprogrammable circuit, it is still a bunch of small, *unchangeable pieces of hardware* that are themselves programmable. These are called *primitives*.
- We've mostly just focused on the the CLBs so far
- These have LUTs to perform logical functions (combinationally) and then flipflops to act as our state-storage

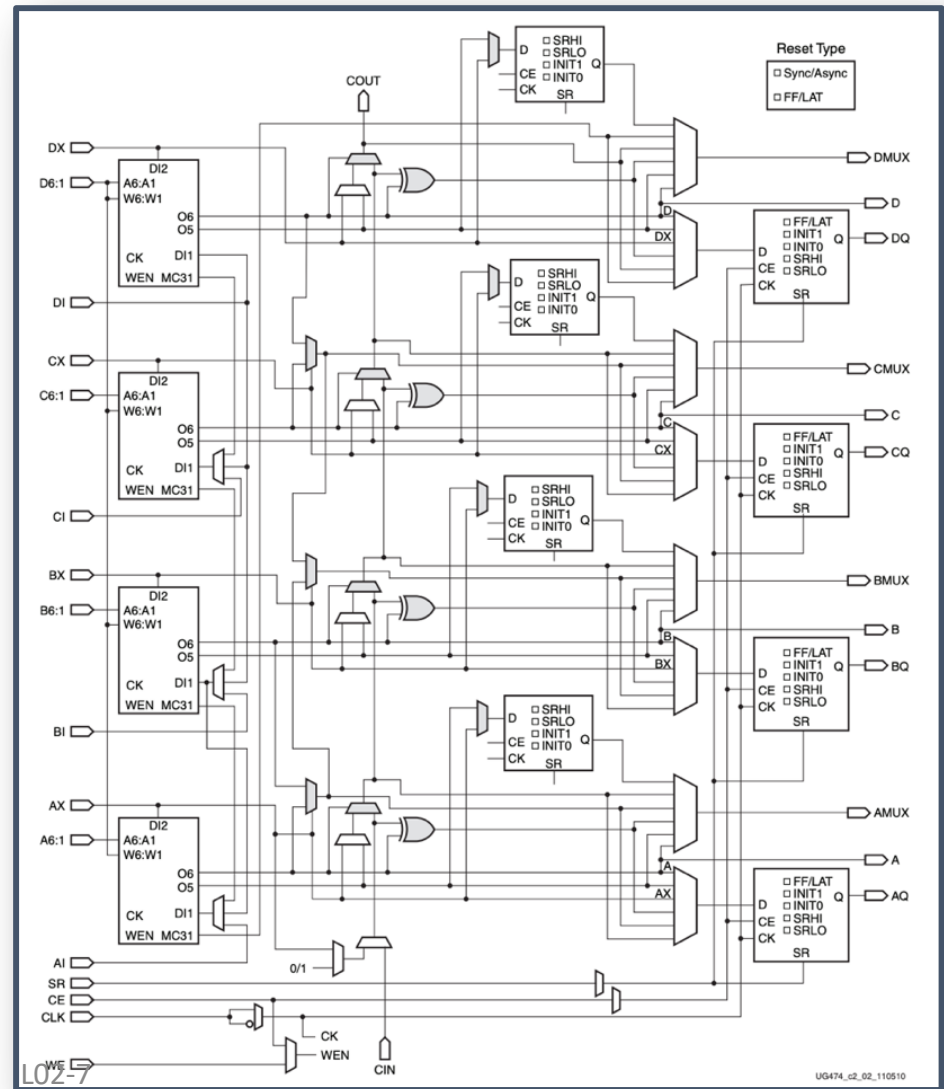
Primitives



- The difference between an FPGA and a GPU or even a CPU is the relative size and number of primitives.
- CPU has one primitive...you program it.
- GPU has hundreds to thousands primitives, each simpler than a CPU...mostly the same type, mostly with fixed data paths.
- FPGAs have thousands to millions of primitives, all much simpler with much less fixed datapaths

FPGA (Xilinx) Configurable Logic Blocks

- Our FPGAs have about 8500 of these →
- Called “Logic Slices”
- Each slice has four CLBs (“Configurable Logic Blocks”) that form the clay from which we sculpt our digital functions

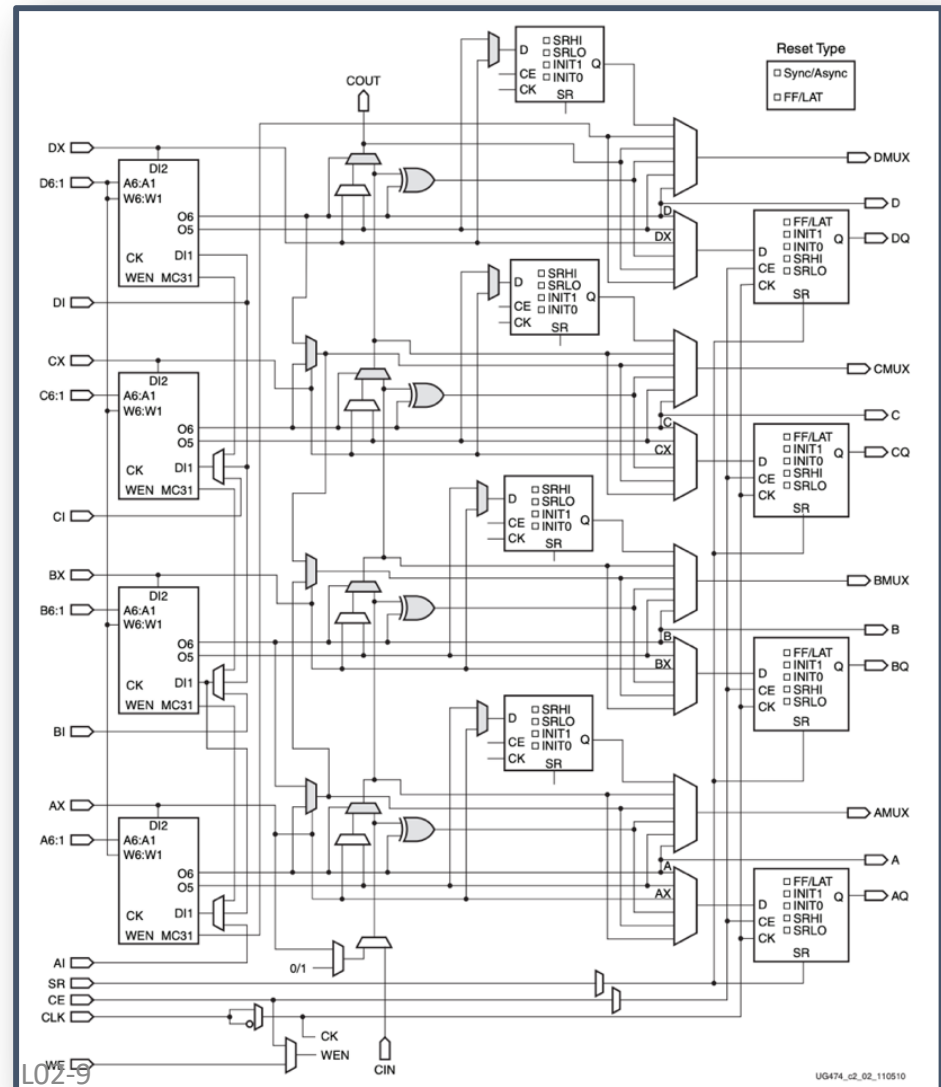


What Other Primitives?

- CLB (Configurable Logic Block)
- MMCM (Mixed Mode Clock Manager)
- BRAM (dense in-chip memory)
- DSPs (small math-performing modules)
- SERDES (serializers/deserializers)
- BUFS (buffers of many varieties)
- Interconnects

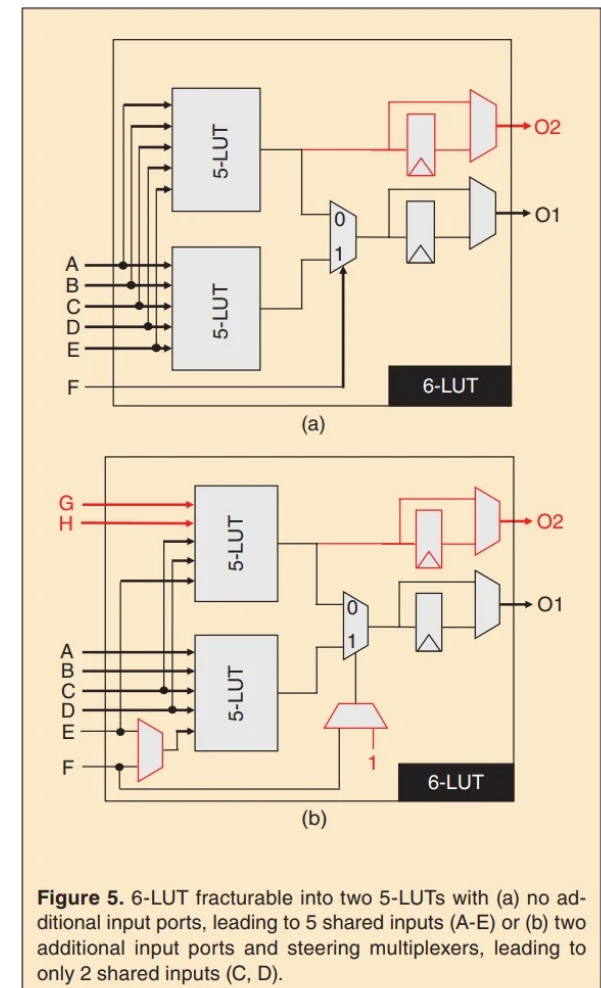
FPGA (Xilinx) Configurable Logic Blocks

- Our FPGAs have about **8500** of these →
- Bulk of our FPGA are these:
 - Four LUT6's
 - Eight Flip flops
 - Carry Chain



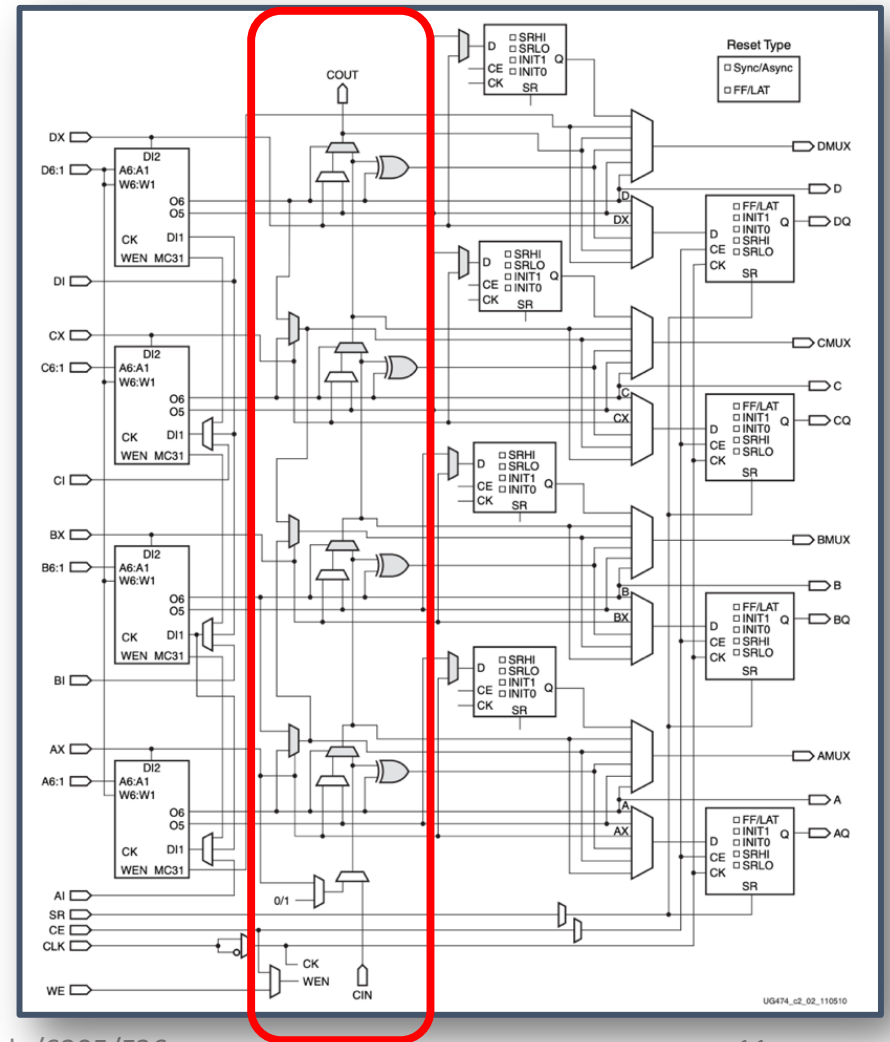
Each of the CLB parts has its own details

- Each LUT6 can either be a LUT6 (or any smaller LUT)
- OR Can be a pair of LUT5s with a shared subset of inputs



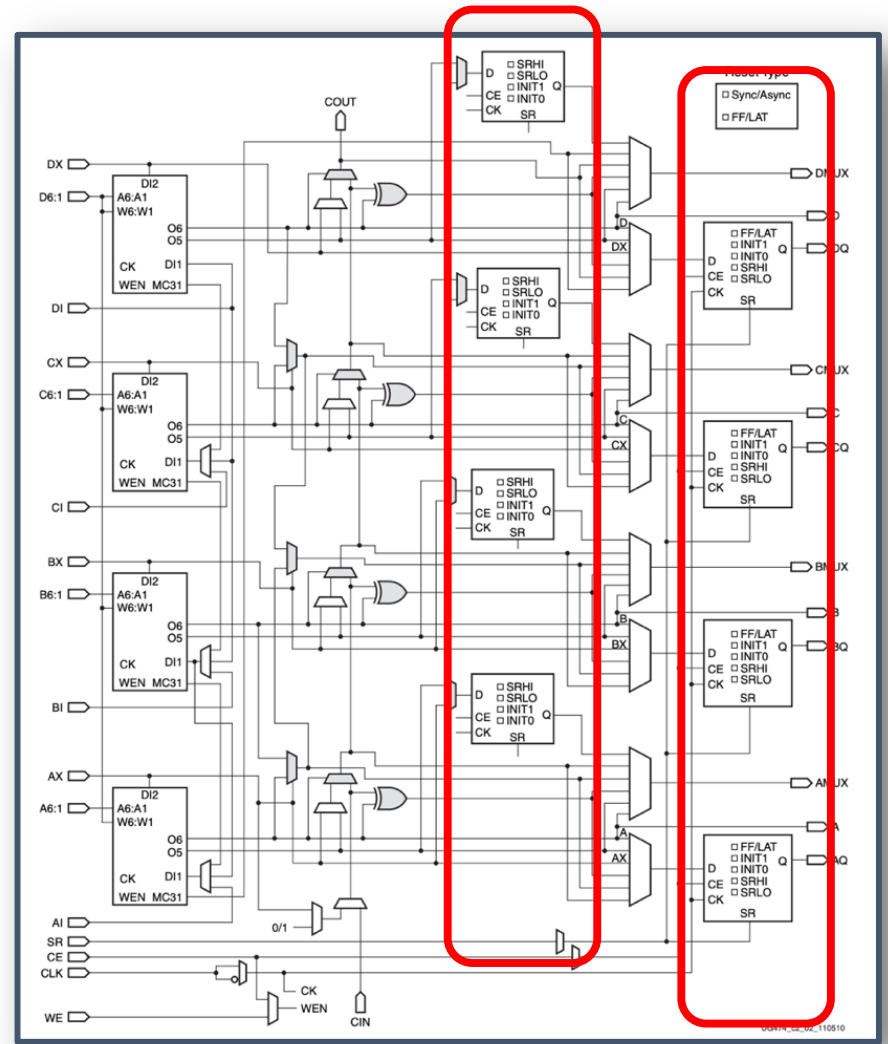
Each of the CLB parts has its own details

- The fast-carry chain allows different CLBs to transfer data with little propagation delay for the purpose of synthesizing chained adders



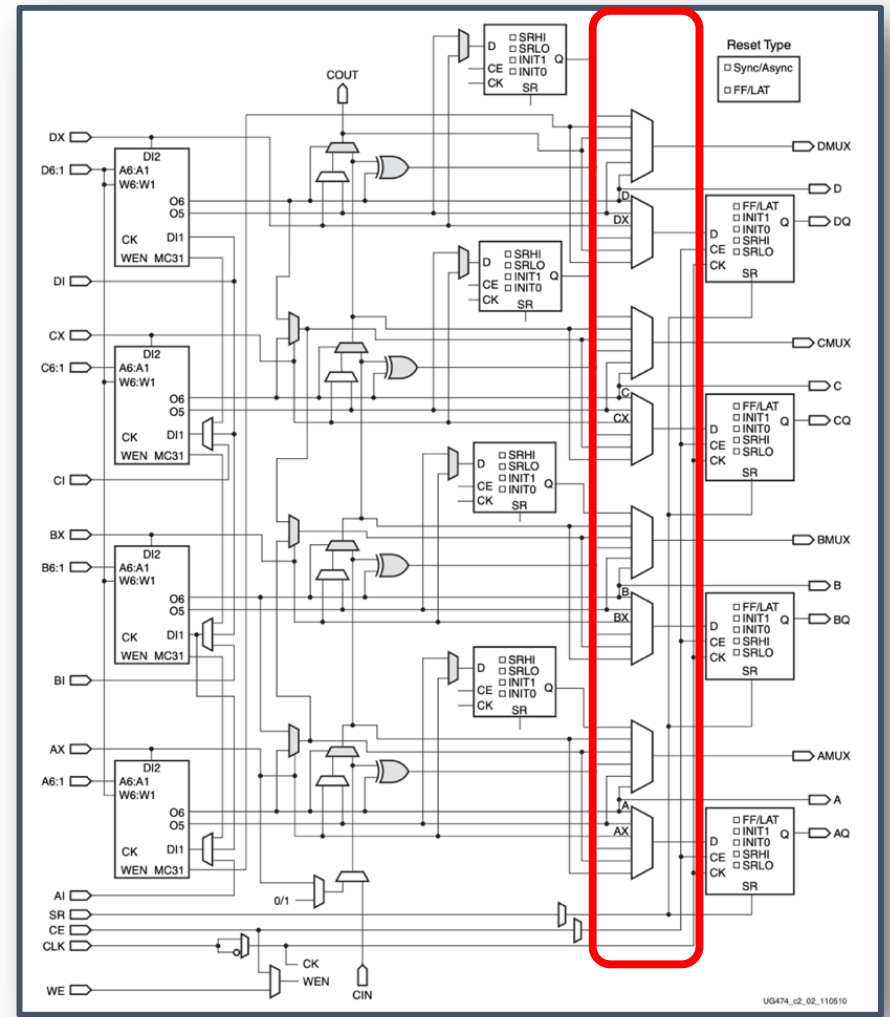
Each of the CLB parts has its own details

- The eight flip-flops can be used for sequential logic...



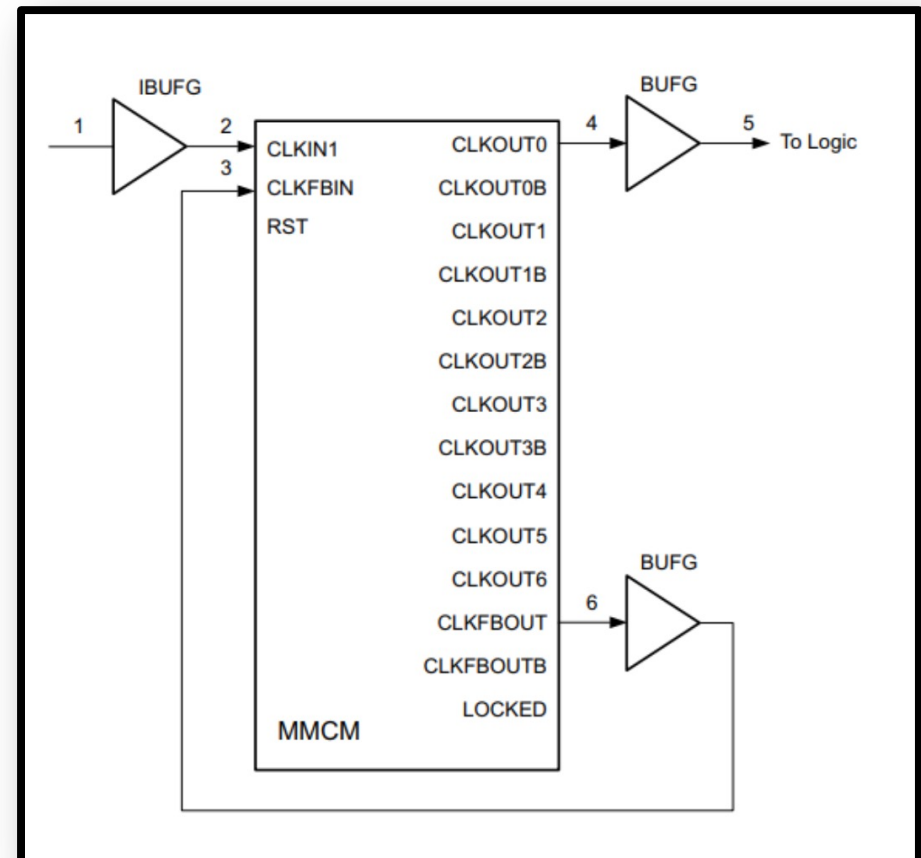
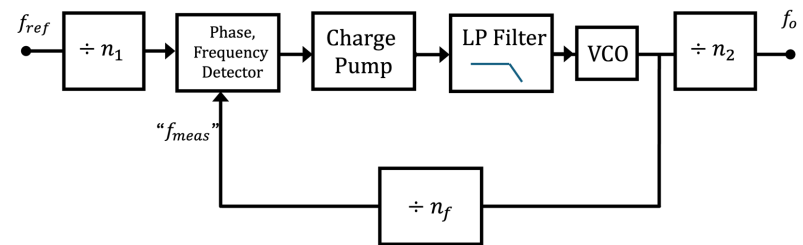
Each of the CLB parts has its own details

- ...Or bypassed if stacking lots of combinational logic together...



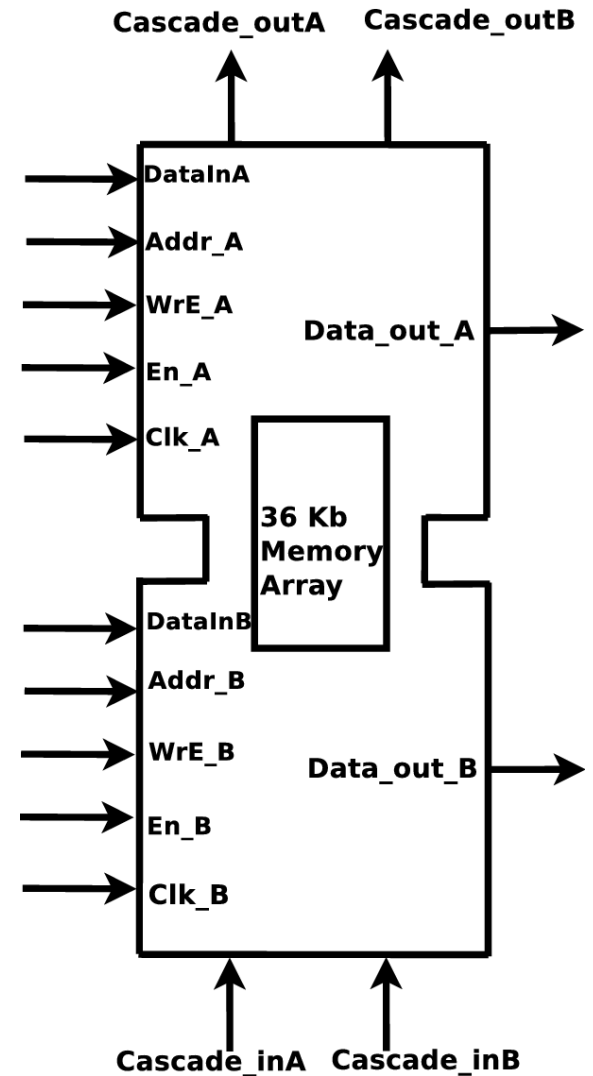
MMCM

- Configurable PLLs with tons of additional features that allow the generation of clean stable clocks



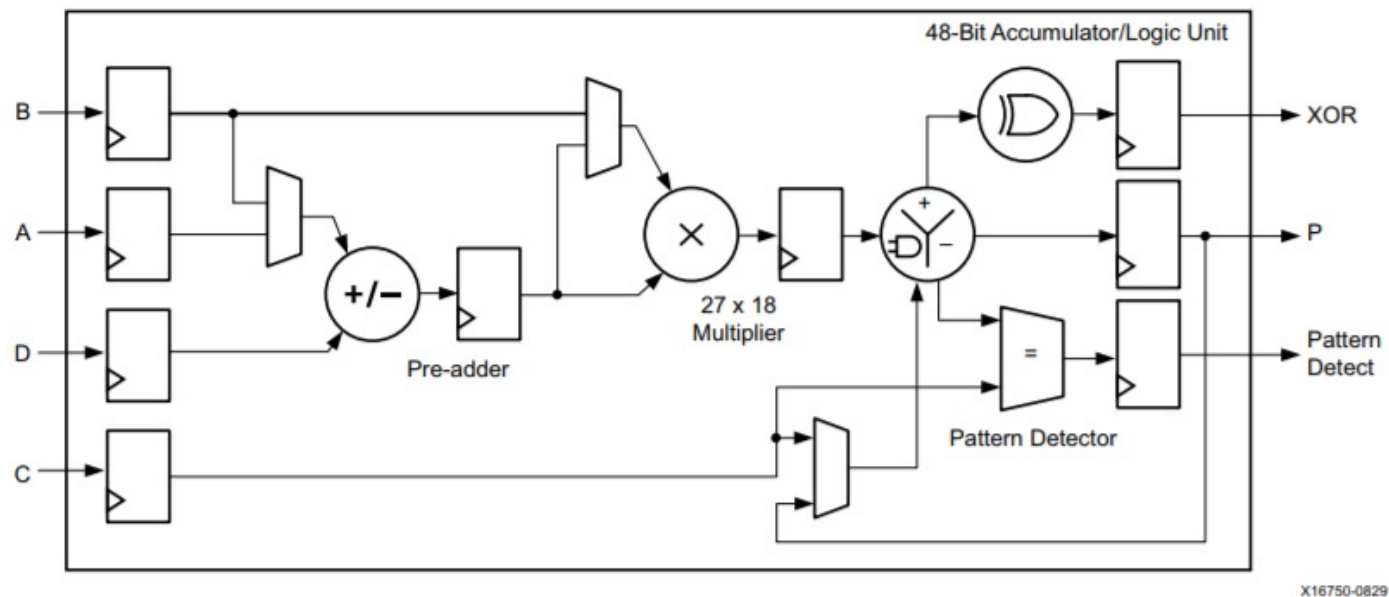
Block RAM

- Very high-density memory units
- Meant for storing larger amounts of data compared to a few flip flops.
- Talk about next week

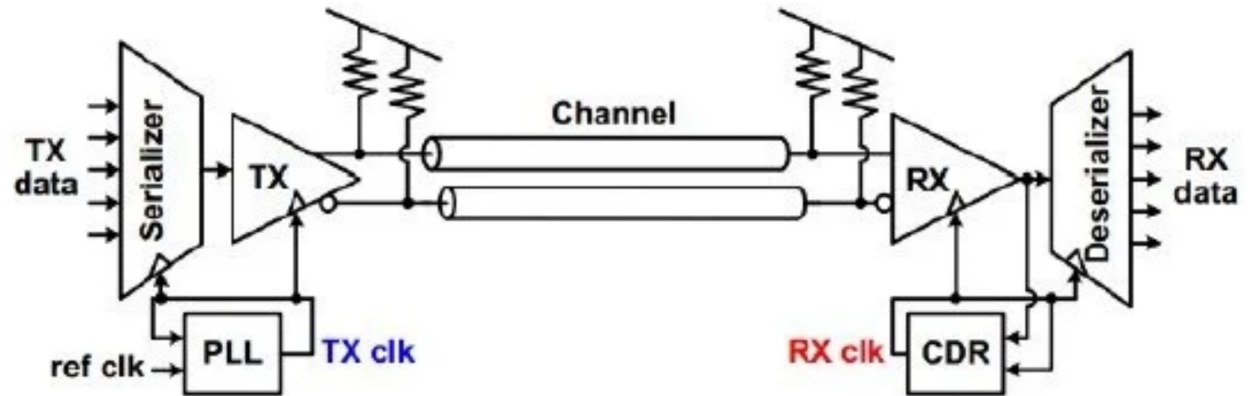


DSPs

- A “hardened” multiply-accumulate module that can do certain math faster than if built using LUTs and FFs from the CLBs



SERDES



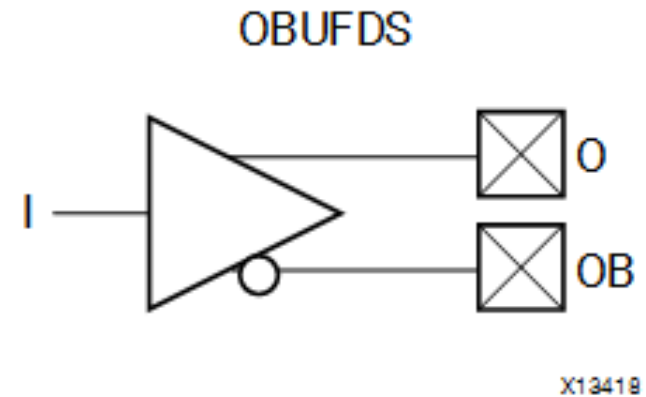
1: A basic block diagram of SerDes architecture

- High-Speed “hardened” data serializers and deserializers
- Allow us to get the functionality of things like our SPI and UART controllers, but at speeds up towards 1 Gbps

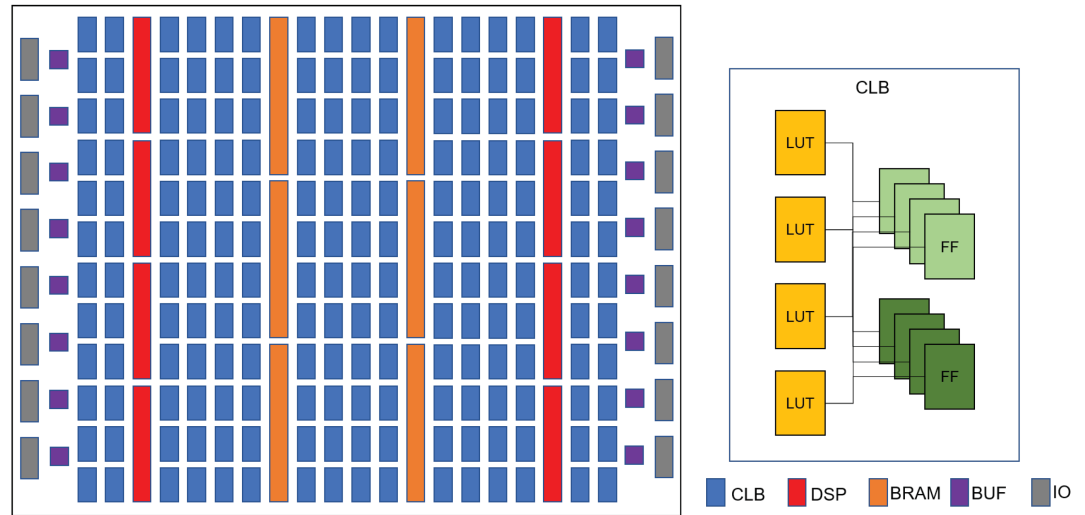
https://www.researchgate.net/figure/A-basic-block-diagram-of-SerDes-architecture_fig1_380760498

OBUFs IBUFs

- Variety of input and output buffers that can convert our internal 1's and 0's into other varieties for outside consumption



Routing



- The CLBs, the Memory, the DSPs, the MMCMs, the IO...these are all spread out over the FPGA in a giant grid.
- How Does One block Connect to Another?

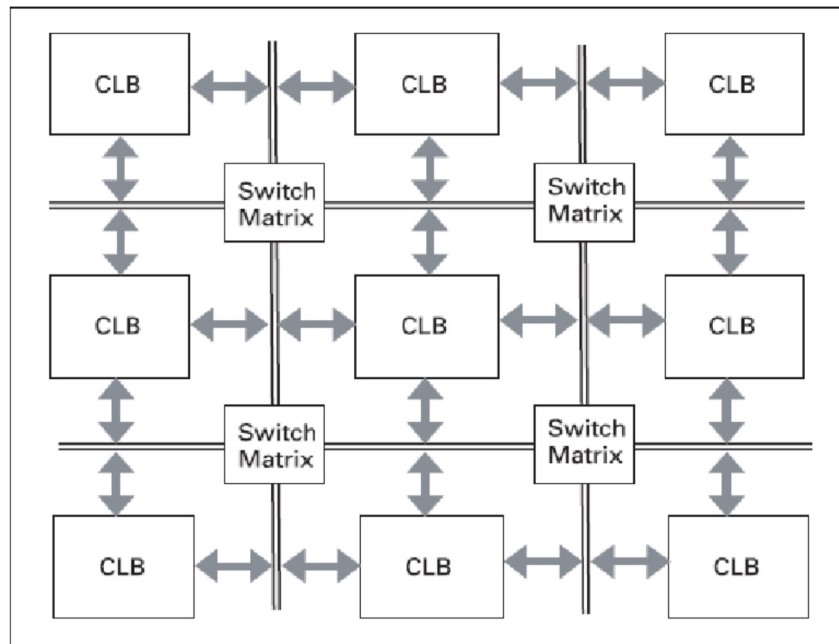
<https://www.mdpi.com/2079-9292/13/1/37>

Switch Matrices

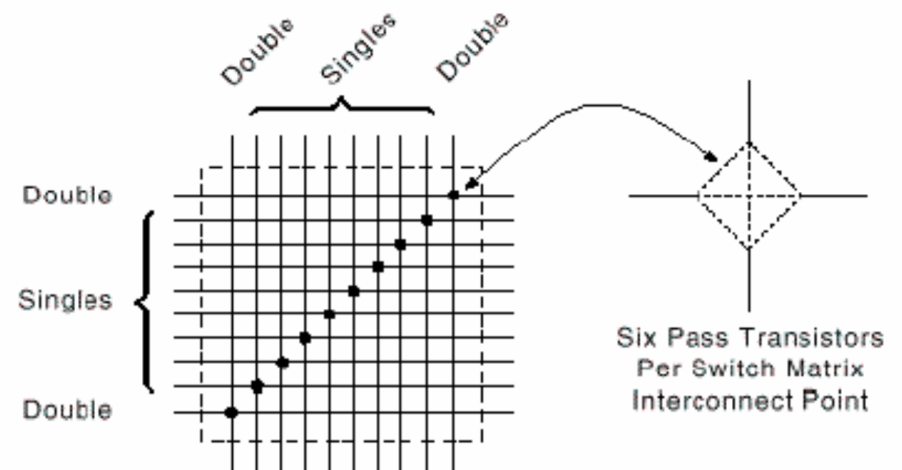


- At the input and output of every primitive there are on-ramps and off-ramps to data busses.

Switch Matrices



6 Basic Architecture of a Xilinx FPGA



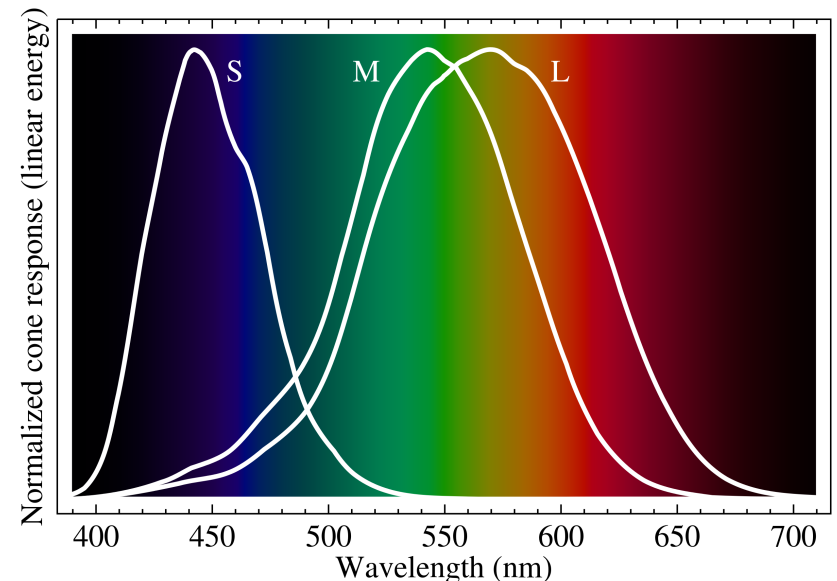
2 Programmable switch matrix.

https://www.researchgate.net/figure/Basic-Architecture-of-a-Xilinx-FPGA_fig5_324678474
https://www.researchgate.net/figure/Programmable-switch-matrix_fig8_313617498

Video

Video and Displays are for *Human Eyes*

- Human color perception comes from three types of cone cells in the center of the eye. Each type generally has an abundance of one photoreceptive protein (which causes electrical stimulation):
 - S cones with protein from **OPN2** gene
 - M cones with protein from **OPN1MW** gene
 - L cones with protein from **OPN1LW** gene
- A human eye therefore has three independent inputs regarding visual EM radiation*
- Called "**trichromatic**"

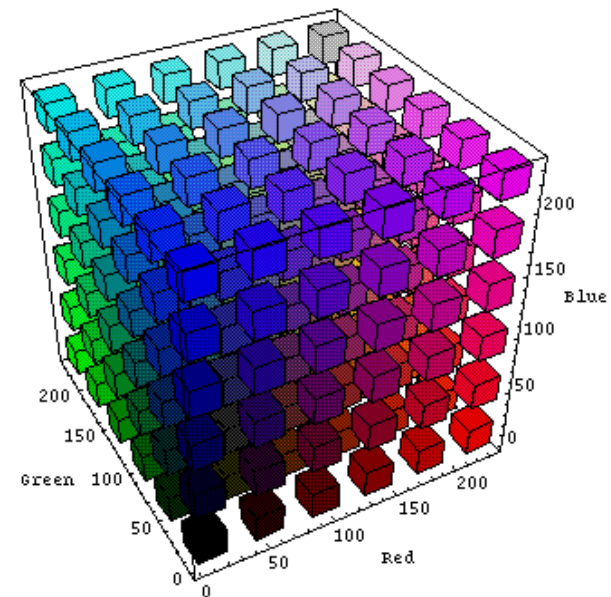


Color Space

- Human trichromatic vision is comprised of three inputs, therefore the most general way to describe these inputs is in a 3-dimensional space
- Because the L, M, and S cones “roughly” line up with Red, Green, and Blue, respectively a RGB space is often the most natural to us*
- There are others, though

*One form of RGB space
(not the only way to
display it)*

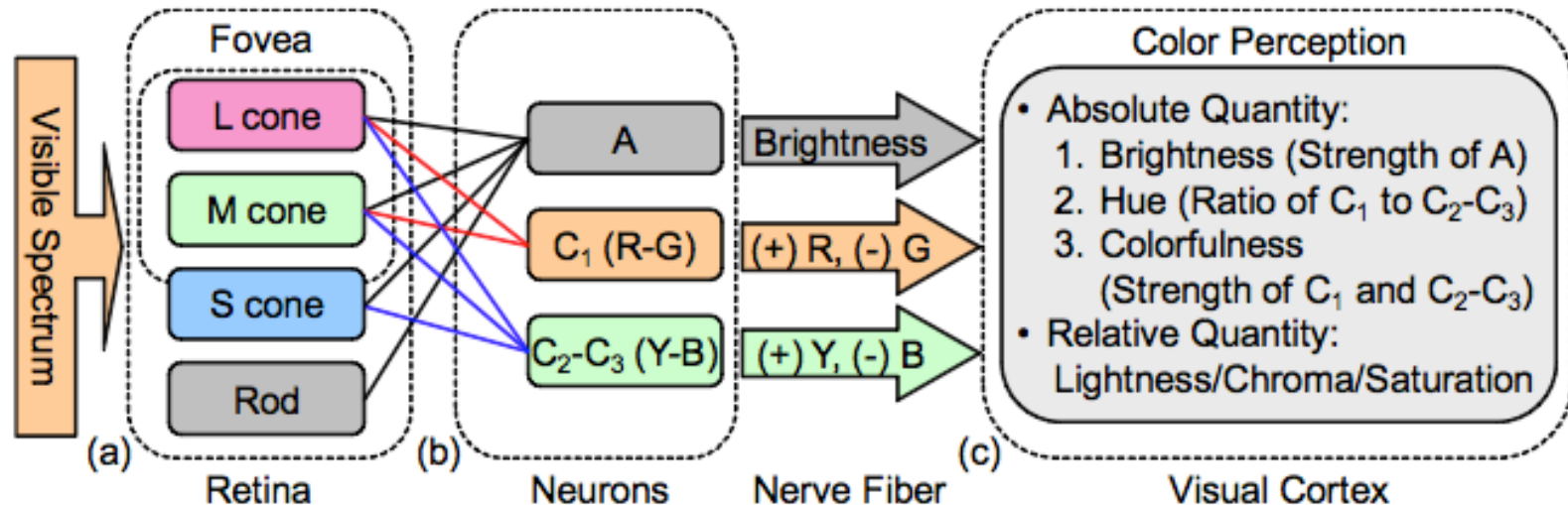
*it actually isn't



<https://engineering.purdue.edu/~abe305/HTMLS/rgbspace.htm>

Opponent Process Color Theory

- It actually isn't as simple as trichromatic vision
 - Firstly human eyes technically have four light input channels
 - The signals are combined in the second layer of the artificial neural net of our brain to give three different signals



https://en.wikipedia.org/wiki/Color_vision#:~:text=Two%20complementary%20theories%20of%20color,green%2C%20and%20red%2C%20respectively.

Revealing these signals

- On the following page, we'll stare at a white dot for ~ten seconds, and then the slide will change to an all white slide with a black dot
- (The dot is merely there for your eye to have something to focus on...nothing to do with what you're about to experience)



<https://muireall.space/opponent/>

October 1, 2026

<https://fpga.mit.edu/6205/F26>

27



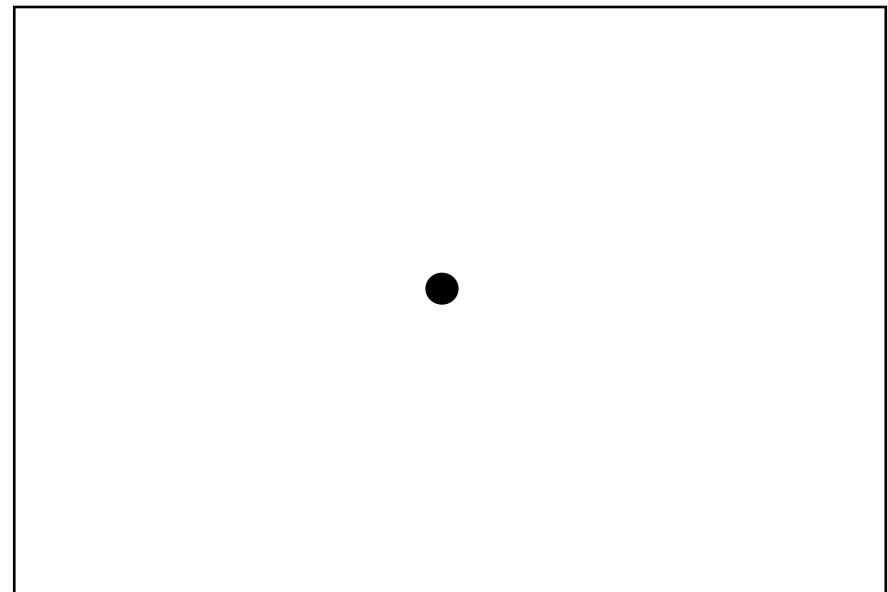
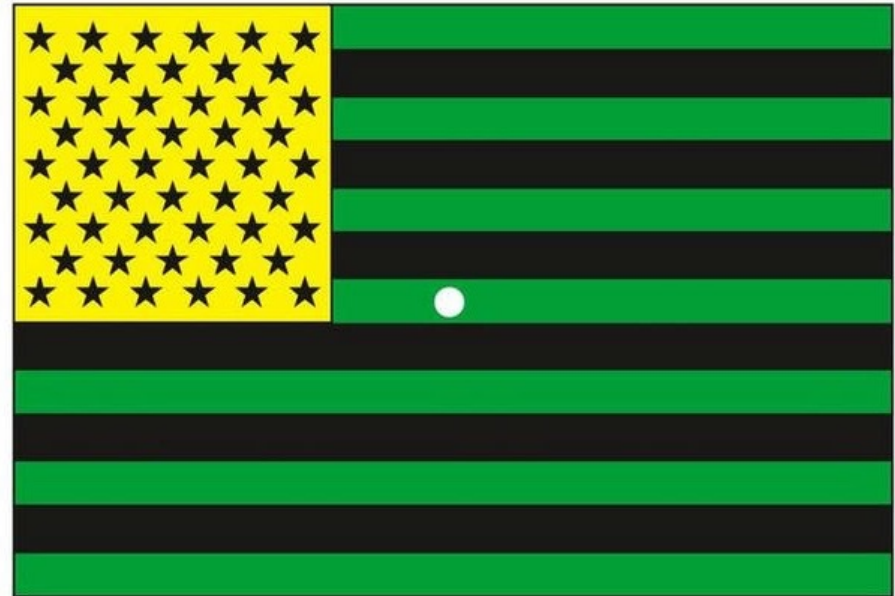
<https://muireall.space/opponent/>

October 1, 2026

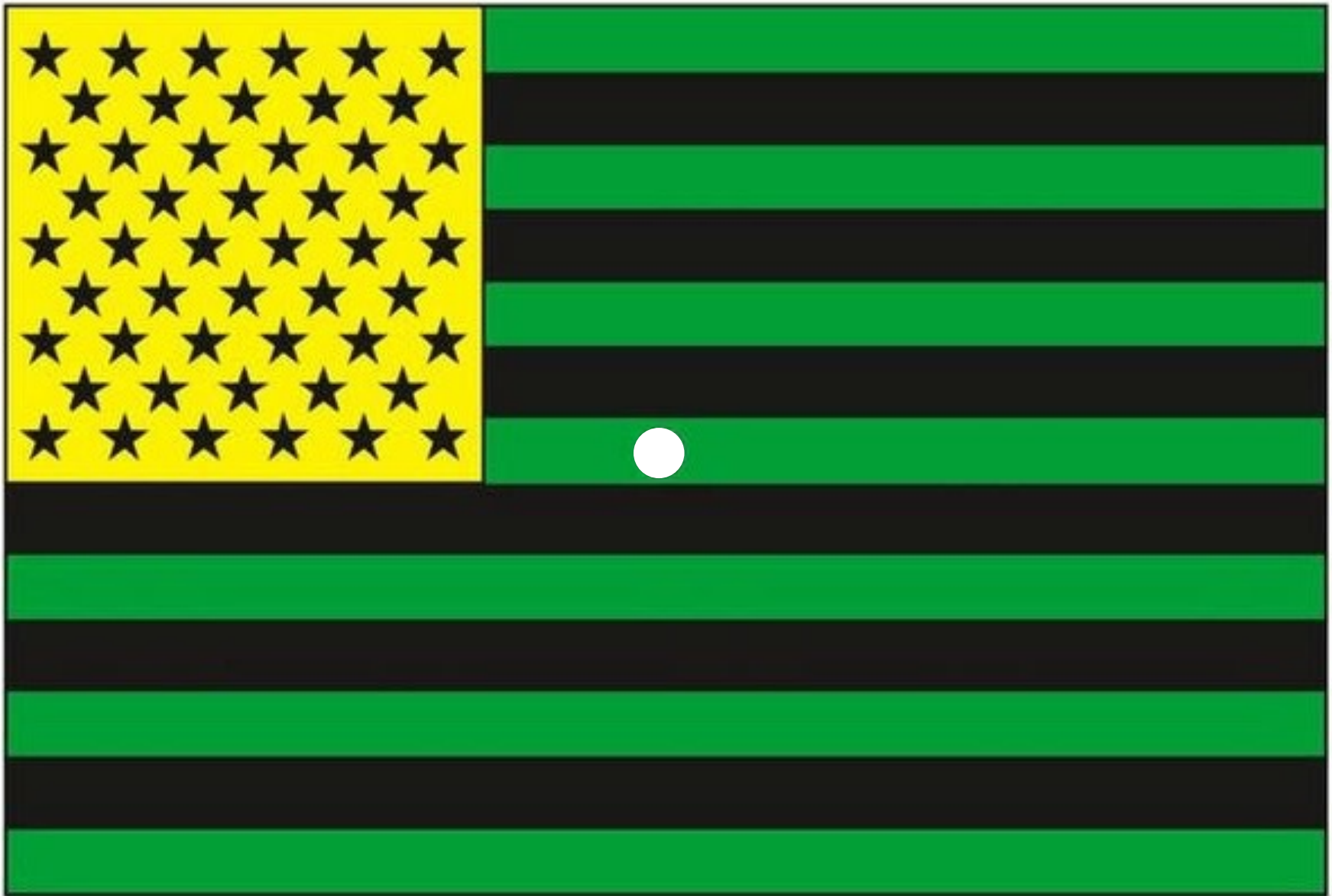
<https://fpga.mit.edu/6205/F26>

Revealing these signals

- Stare at the white dot in the middle of the flag to the right for ~30 seconds
- Then look at the black dot
- The properly colored US flag will appear in the white box



<http://therefractedlight.blogspot.com/2010/07/optical-illusion-stare-at-white-x-for.html>



<https://muireall.space/opponent/>

October 1, 2026

<https://fpga.mit.edu/6205/F26>

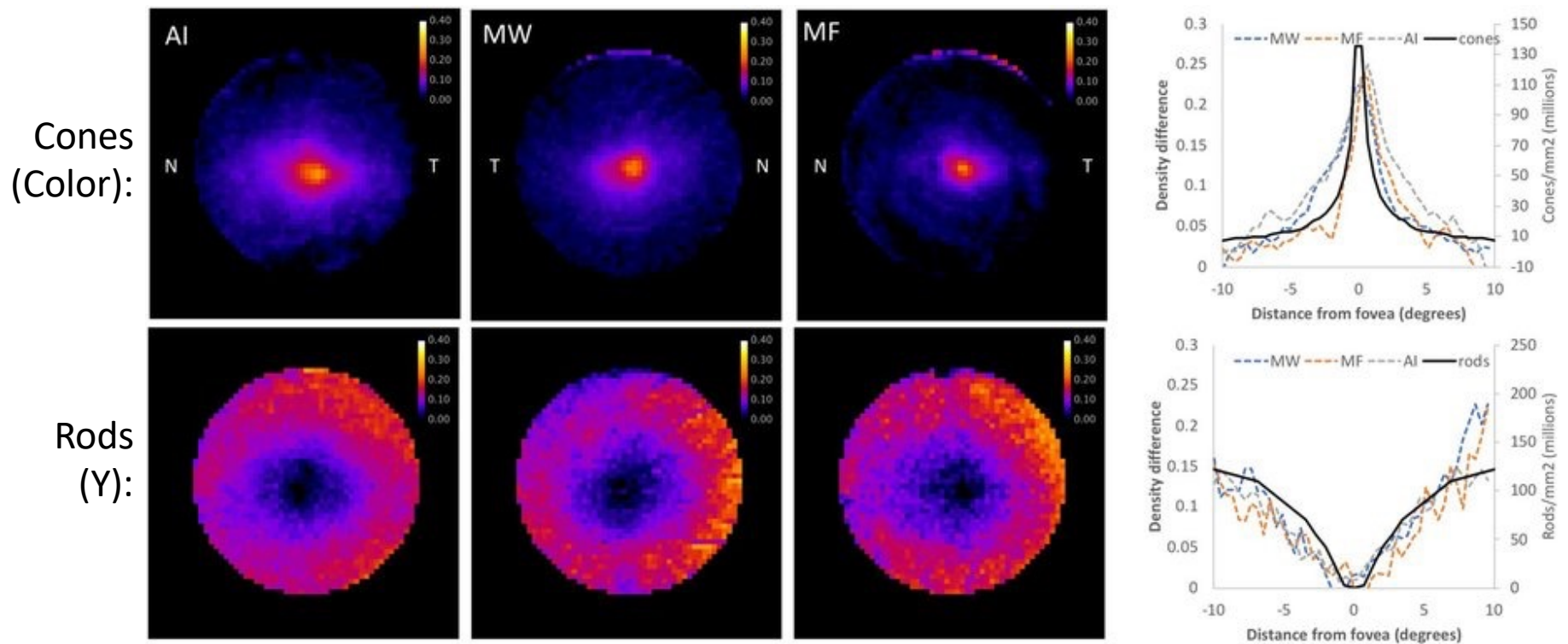


<https://muireall.space/opponent/>

October 1, 2026

<https://fpga.mit.edu/6205/F26>

Rods and Cones are Distributed Differently



Margrain, Tom & Atkinson, David & Binns, Alison & Fergusson, James & Gaffney, Allannah & Henry, David & Jones, Chris & Lamb, Trevor & Melotte, Dave & Miller, Chris & Todd, Stephen & Wood, Ashley. (2020). Functional Imaging of the Outer Retinal Complex using High Fidelity Imaging Retinal Densitometry. Scientific Reports. 10. 4494. 10.1038/s41598-020-60660-9.

Rods are *much more* sensitive to light. This is why...

- You see better out of the corner of your eye at night.
- Why at night you think you see a spirit or monster out of the corner of your eye...then you look and it has disappeared.

Rods Have Spectral Sensitivity Too!

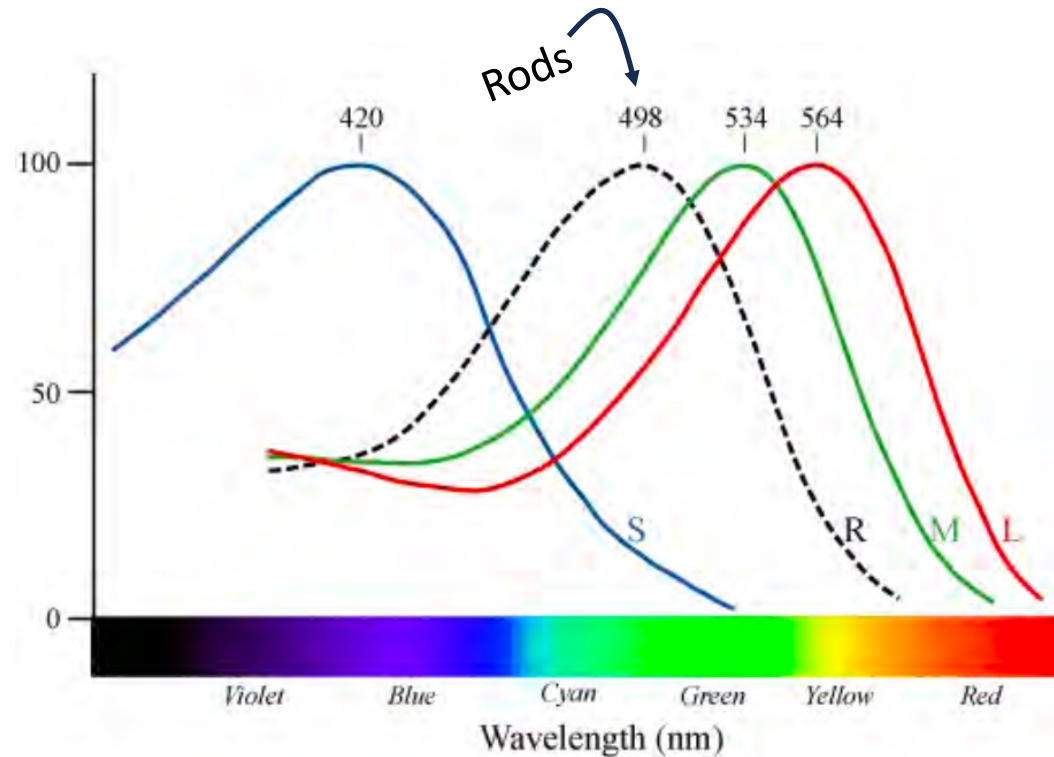


Figure 2.7: Normalized spectral sensitivity of retinal rod and cone cells

Perz, Malgorzata. Flicker perception in the periphery.

https://www.researchgate.net/publication/265155524_Flicker_perception_in_the_periphery

October 1, 2026

<https://fpga.mit.edu/6205/F26>

34

This is why...

- When submarines (or anybody) go to battle stations they illuminate with red
- Why astronomers use red flash lights
- Why at night you'll have a hard time seeing red things out of the corner of your eye

Summary...Worst Case Scenario

- If a person has all color receptors working...
- because of noise limitations in our naturally-evolved encoding scheme that communicates from the cone cells up to the brain...
- we can perceive about 7-10 million unique colors depending on your research source...
- How many bits do we need to encode all possible colors for this worst case?

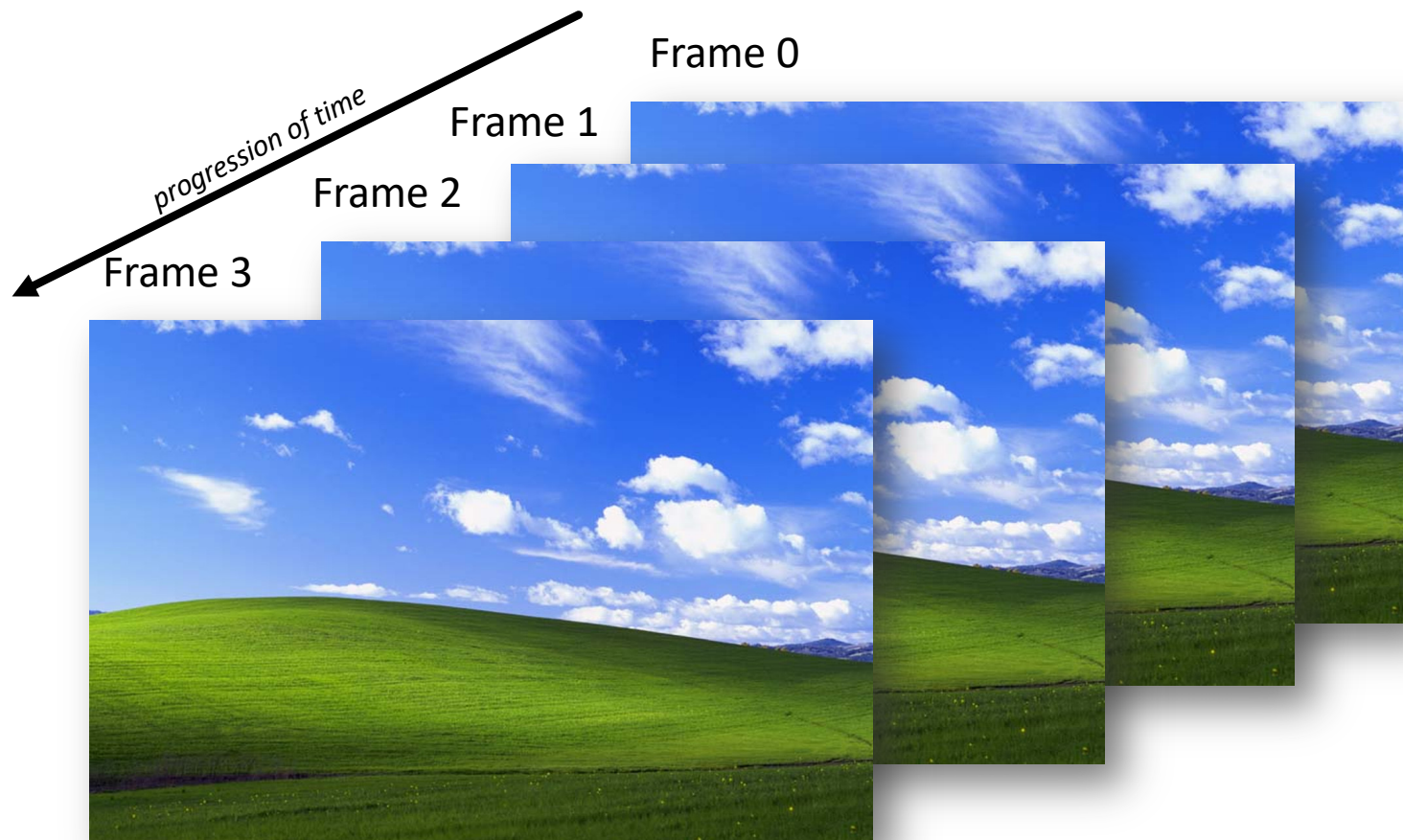
Image or Frame

- An image/frame can be thought of as a 2-dimensional array of 3-tuples:
 - 2 spatial dimensions
 - 3 color dimensions
- Each color tuple is a “pixel”



Video (just draw a bunch of frames quickly)

- Rely on the poor time constants of our eye's to "fake" motion.



How to Transmit 6-dimensional data?

- Ideally need to convey enough 5D values quickly enough to render images fast enough that they show up as one...
- AND we also need to do the above fast enough so that fresh images appear quickly enough
- Just like we've seen in the first few labs, we'll send this data **serially**

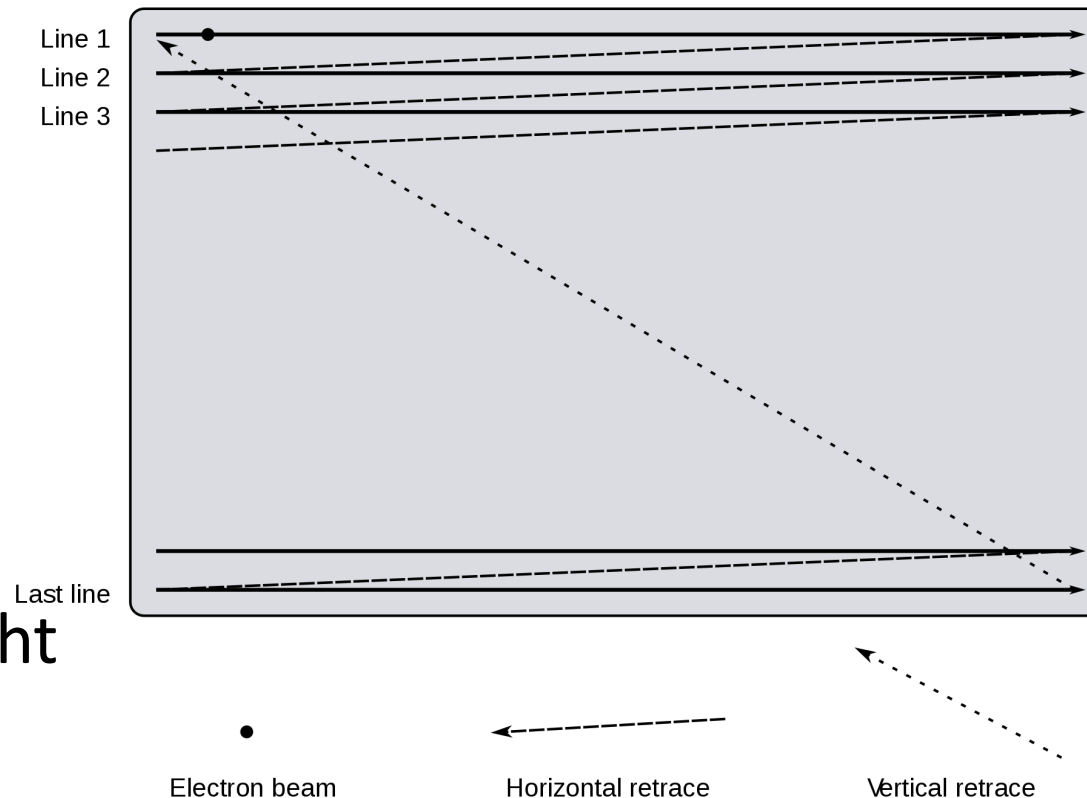
How to Draw: The Raster Scan



*Rastrum, used
for drawing
musical staff*

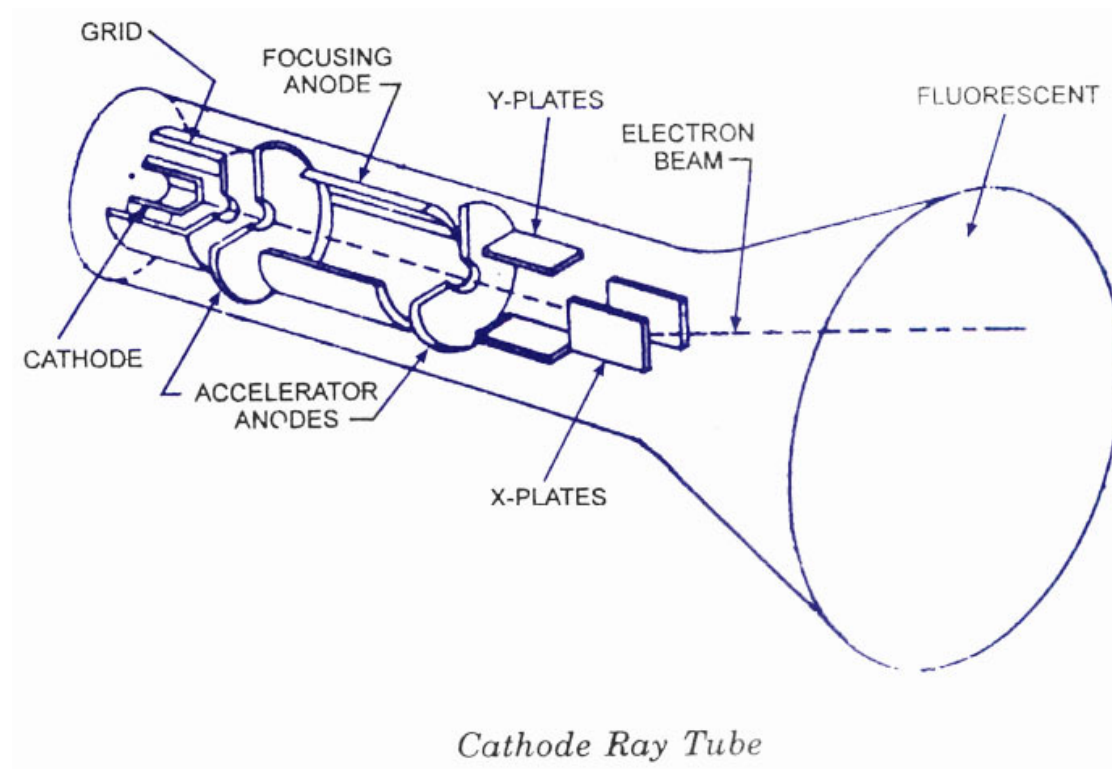
- Spread the drawing out over time
- Images are drawn on a display almost invariably in a “raster” pattern.
- The sequence starts in the upper left, and pixels are drawn:

- Left → Right
- Down a line/back
- Left → Right
- Down a line/back
- Etc...
- End at bottom right
- Return to top left



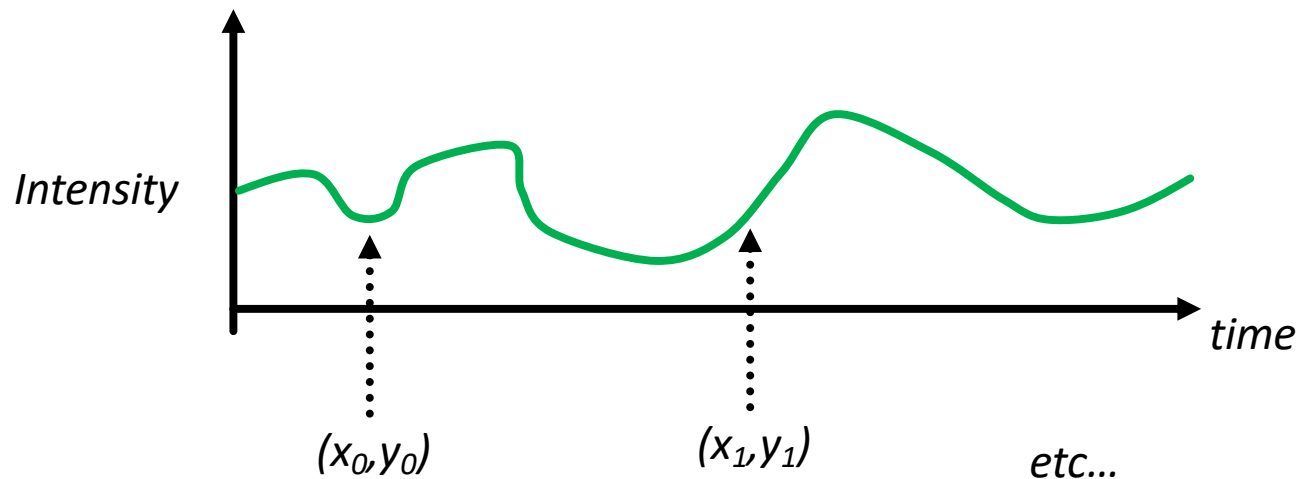
Raster Scan Became Norm because of Early Tech (Cathode Ray Tube)

- Electron beam of varying intensity would be quickly rastered on a fluorescent screen making image



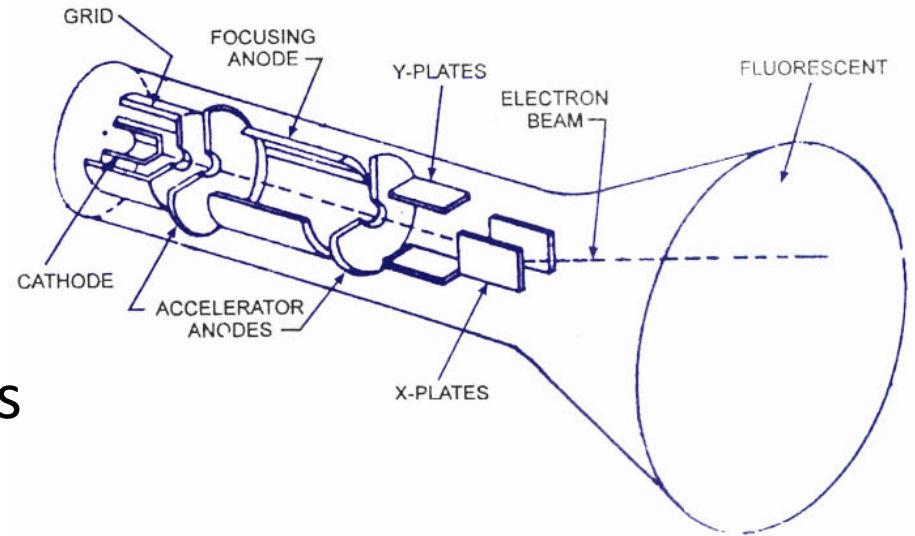
Raster Pattern of Drawing

- Allows time $\leftrightarrow$ position!
 - Takes care of two of the dimensions of info we need to convey!



First Video (Black and White)

- Early technologies prevented ability to detect and *display color*.
- Instead only **brightness (Luminance)** of the image was transmitted/rendered since color couldn't be rendered anyways
- So transmitting an image only involved 3 dimensions of information
 - Two dimensions were conveyed in time
 - One dimension in amplitude



Cathode Ray Tube

<http://www.circuitstoday.com/crt-cathode-ray-tube>

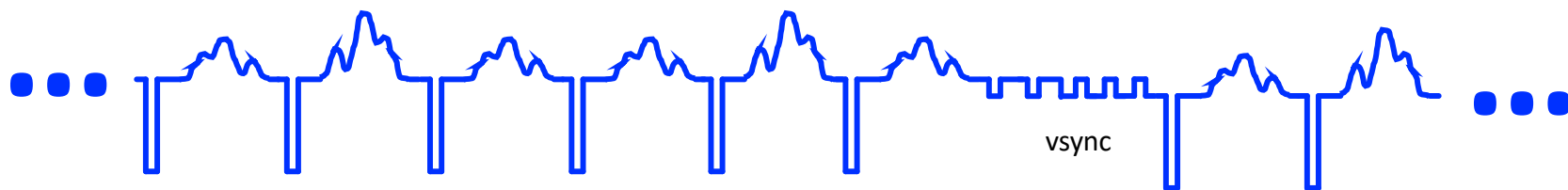


Black and White Video Signal

- A line of video would have:
 - Visual information to be drawn followed by...
 - Special horizontal sync signal that conveyed end of line/move to next line(back to left)
- After many lines of video, you'd have:
 - A second special vertical sync signal that conveyed end of the image, retrace to upper left corner
- Do this forever and ever

Black and White Video signal

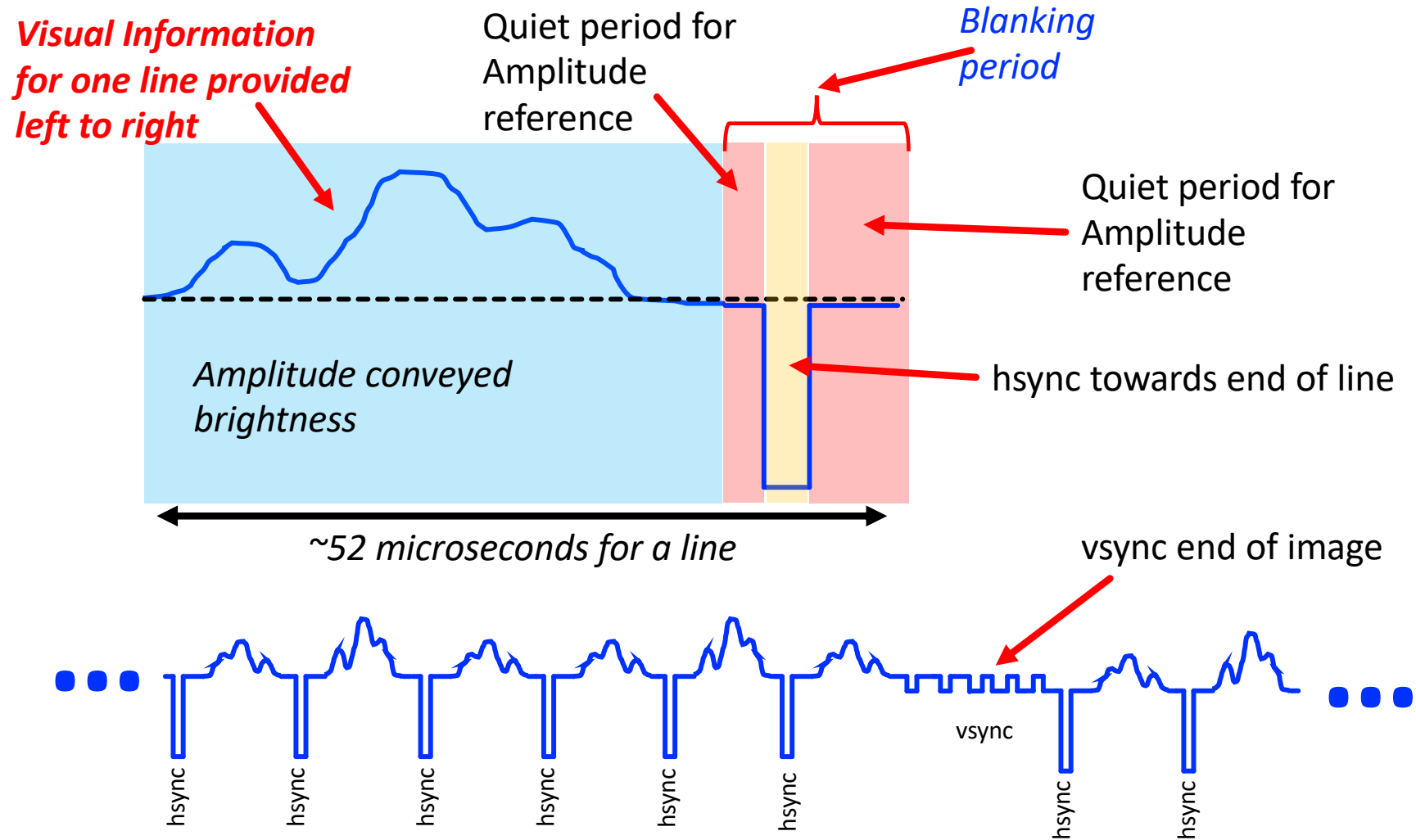
- An **analog** signal conveying luminance (brightness) and synchronization controls (end of line, end of frame)



- Using one single continuously-varying analog voltage, all image information as well as spatial information would be encoded
- Circuits would listen for particular signals in time/frequency and amplitude to know how to control the drawing beam

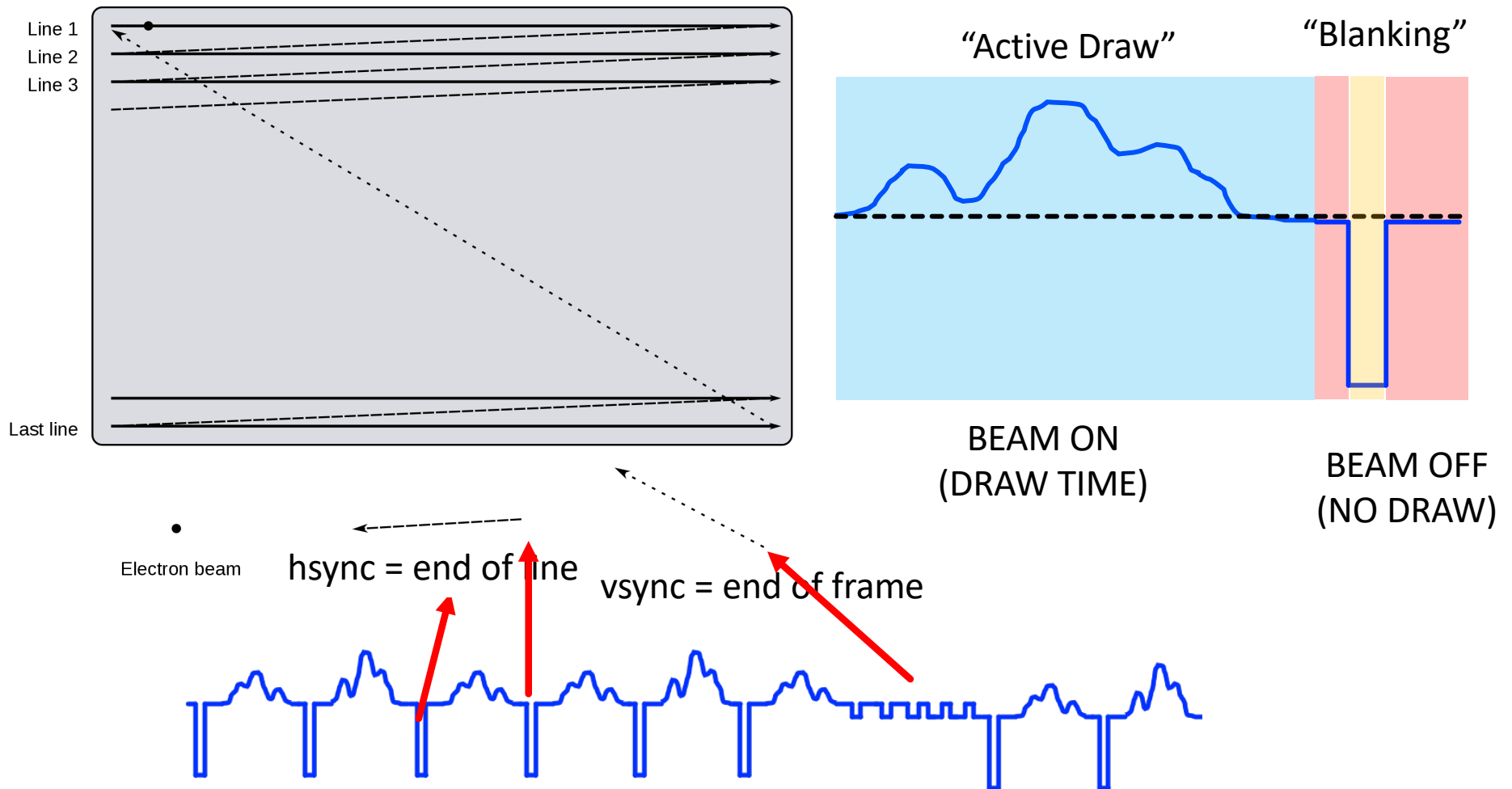
Black and White Video signal

- An **analog** signal conveying luminance (brightness) and synchronization controls (end of line, end of frame)



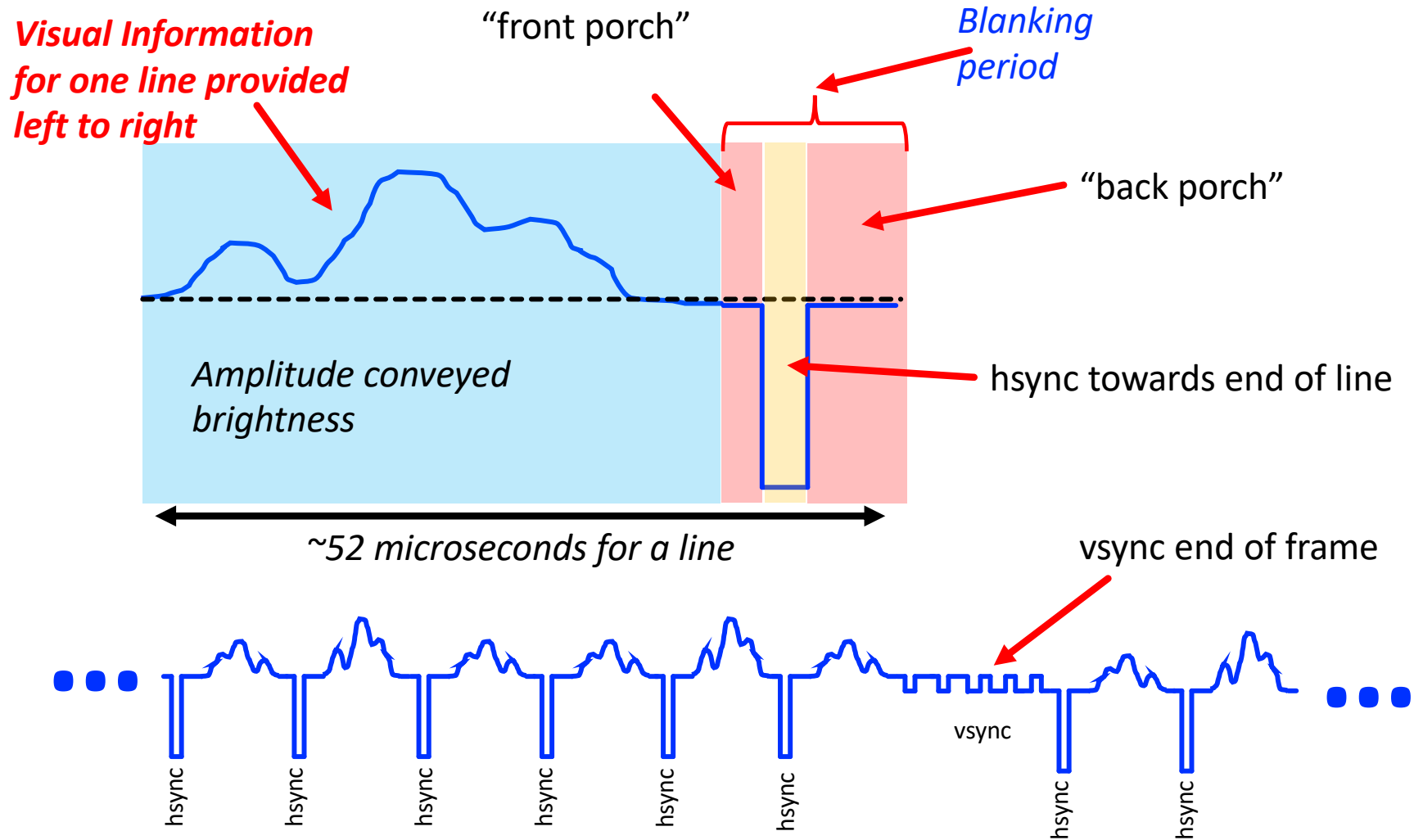
Controls in Action

- Signal completely controls beam location and intensity!



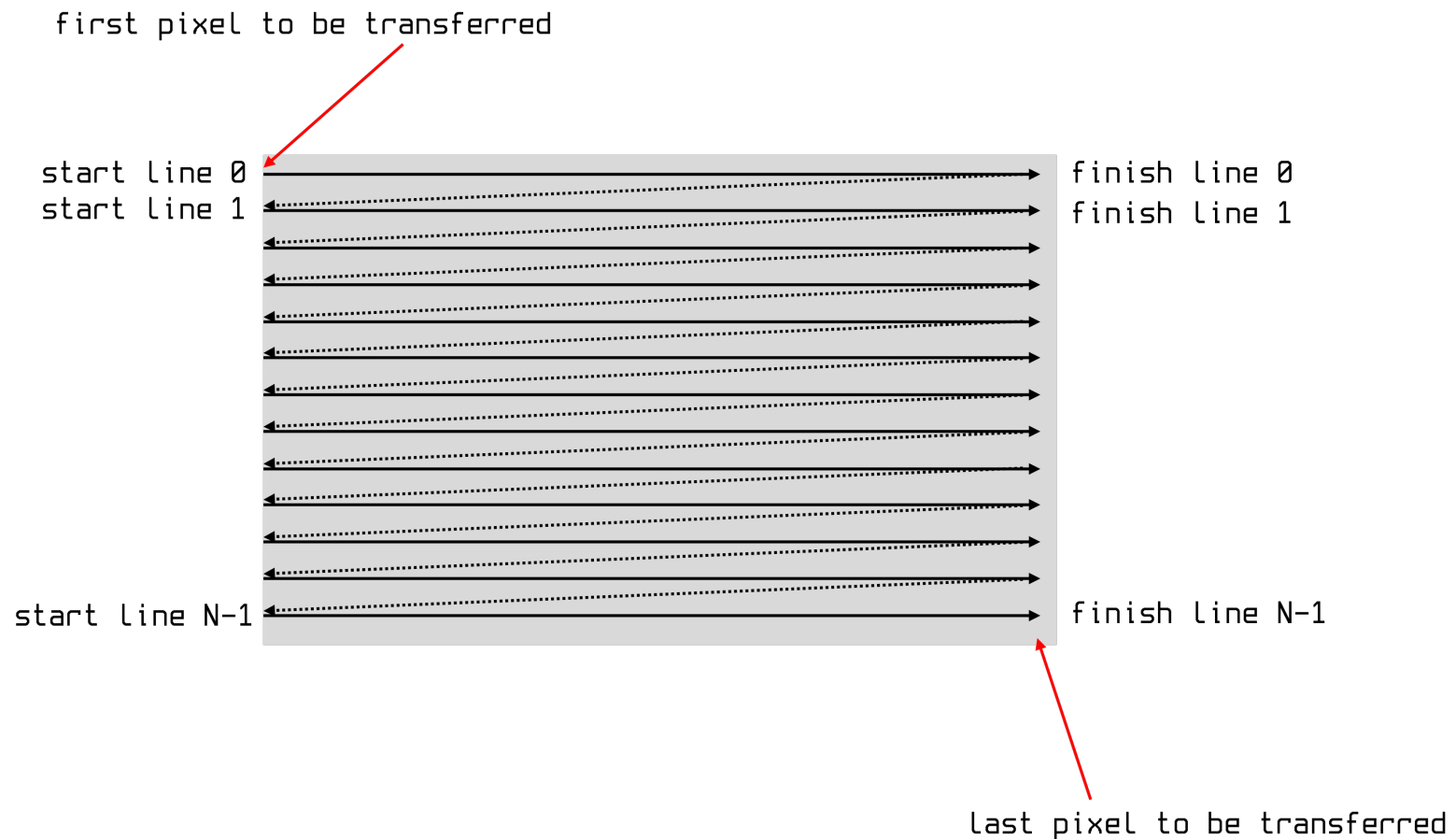
Black and White Video signal

- An **analog** signal conveying luminance (brightness) and synchronization controls (end of line, end of frame)



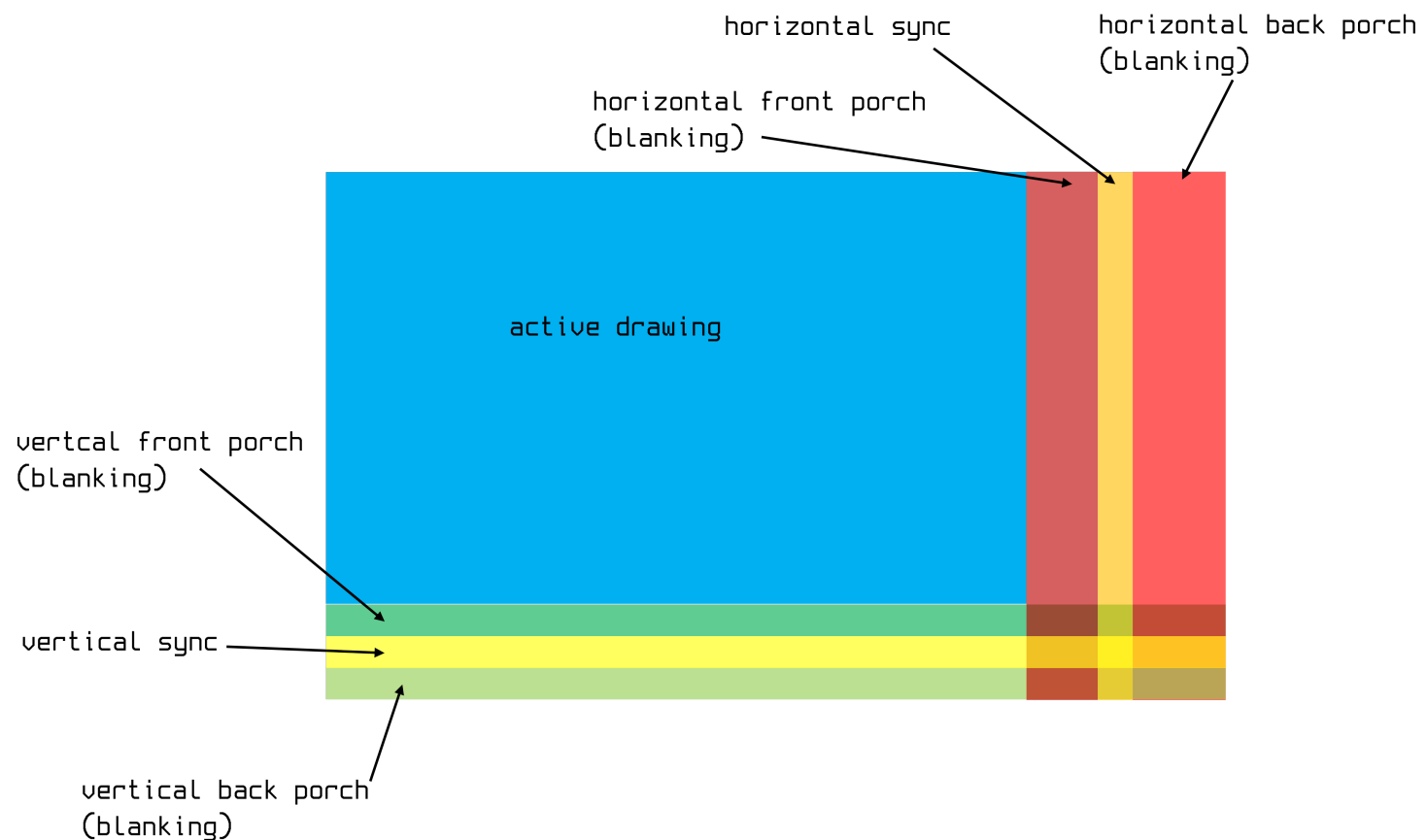
Frame

- So when a “frame” of video was sent it was just a raster pattern of only visual information:



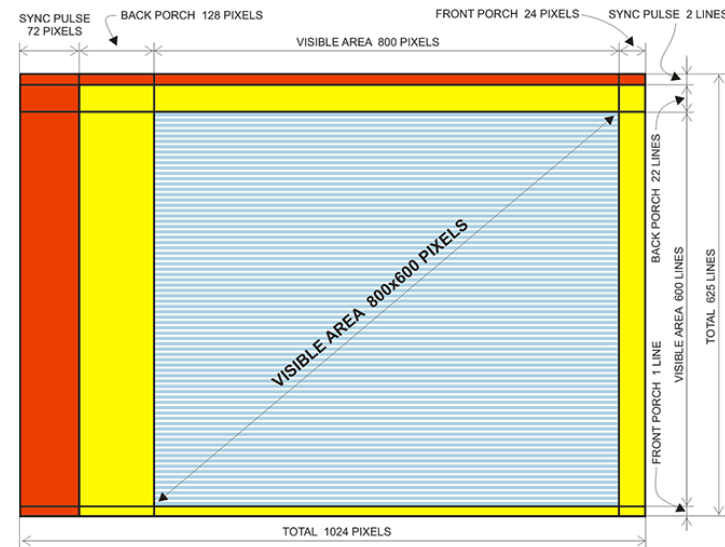
Frame

- Putting it all together...when a frame is sent, only a portion of it is actually the “image” The rest of the frame is control data living on the edges.

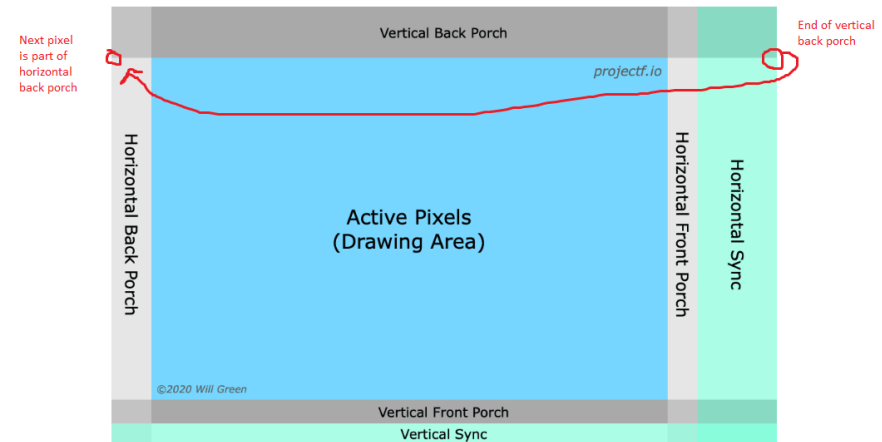


“Location” of Regions of Frame

- Sometimes people will put parts of the blanking region on different sides:



http://www.voja.rs/PROJECTS/GAME_HTM/3.%20VGA.htm

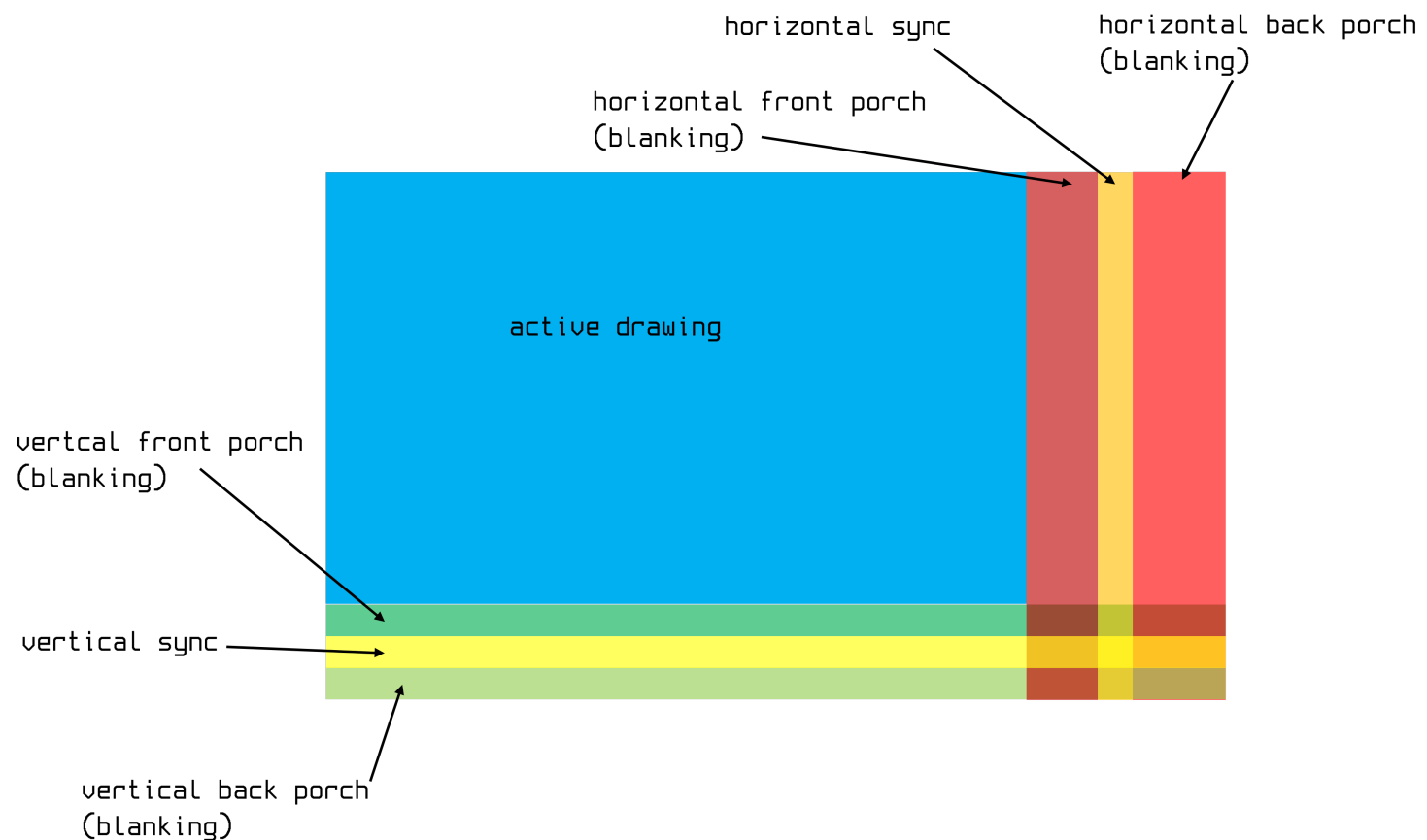


<https://electronics.stackexchange.com/questions/614207/correct-order-of-monitor-display-timing>

- It really doesn't matter. We're sending a serial stream of data. However you want to visualize it (within reason) is fine. **For our class we're doing it the way we show!**

Frame

- Putting it all together...when a frame is sent, only a portion of it is actually the “image” The rest of the frame is control data living on the edges.



Weakness of Analog

- Everything was represented using the same signal and with noise could get very hard to discriminate what was image vs control



Original Video

- With one analog signal using different amplitudes and timings to completely:
 - Specify the grayscale intensity of a pixel,
 - Specify its position on a screen
 - Do so fast enough that enough frames could be drawn in sequence to give the illusion of motion.
- But what about...

Color Cathode Ray Tubes Appear

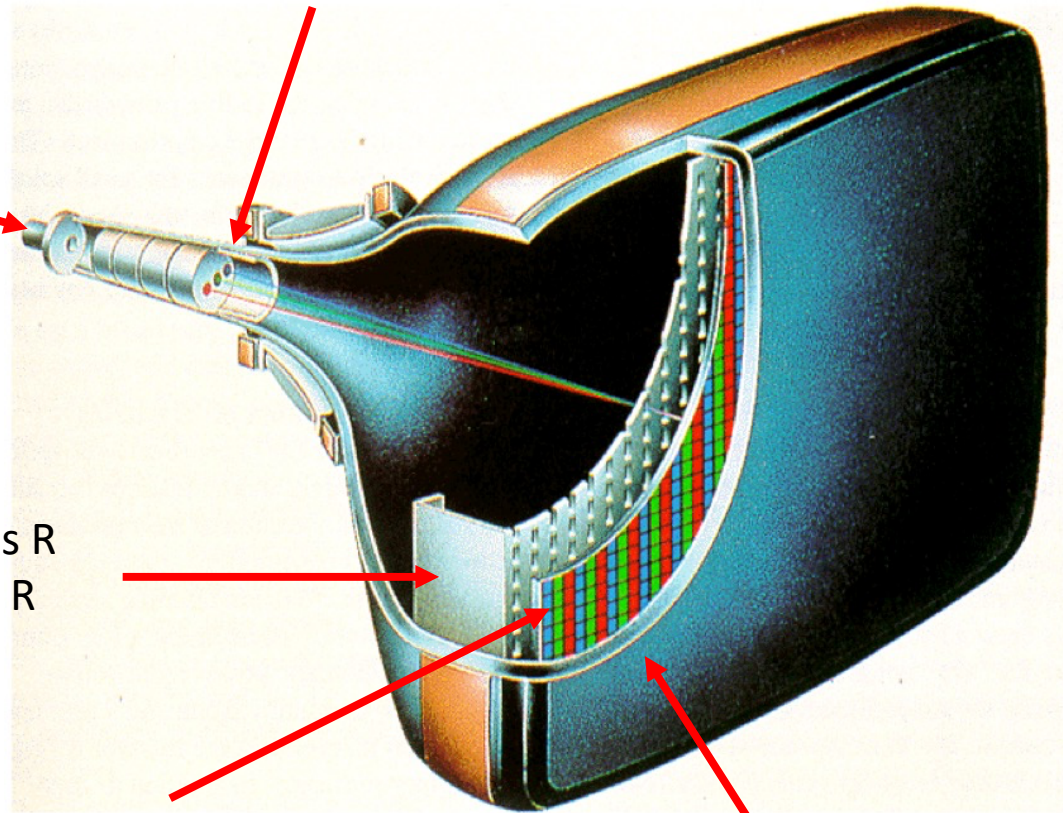
One shared set of deflection coils to sweep all three beams together

Cathode: separate beams for R, G and B

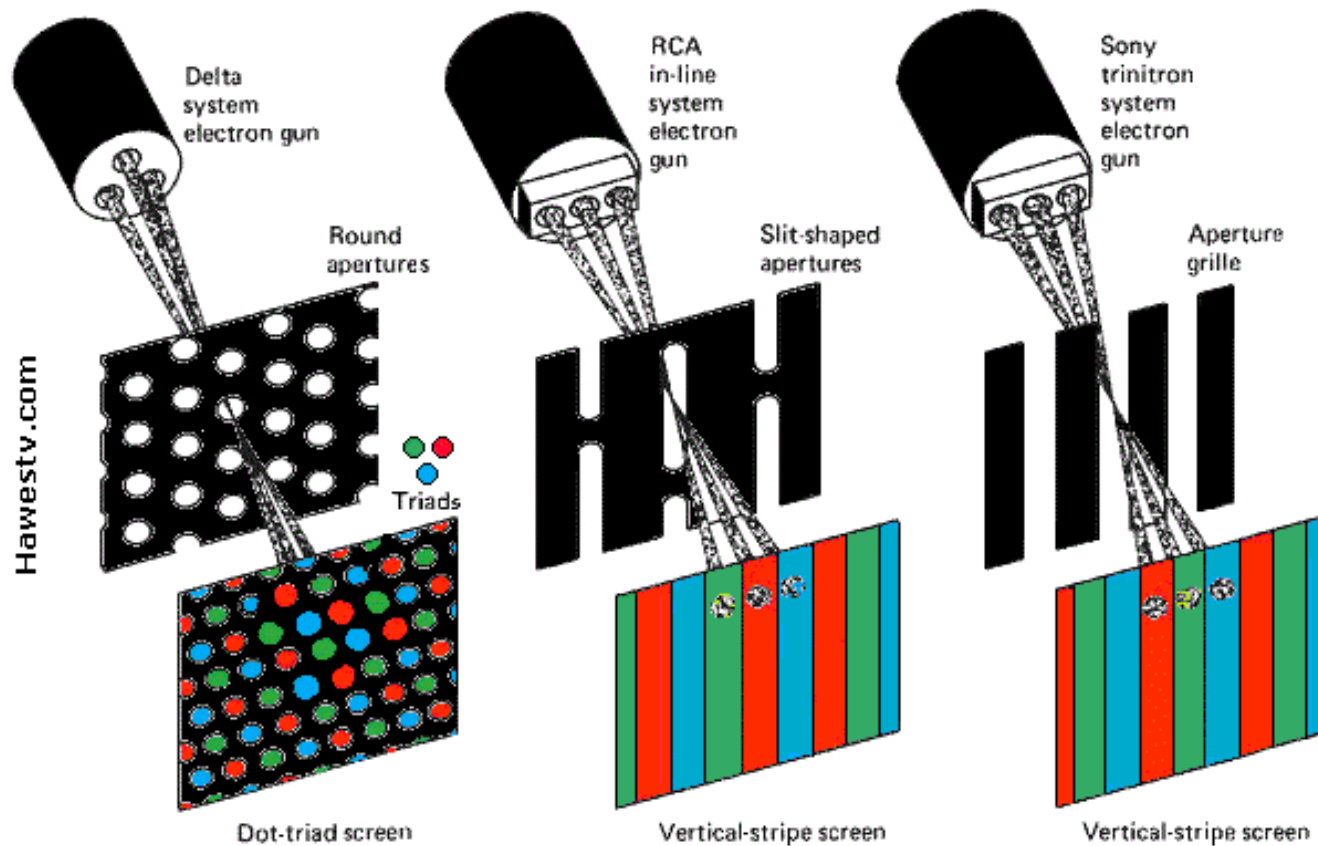
Shadow mask: ensures R beam only illuminates R pixels, etc.

Three different phosphor screens

Anode



Shadow Mask



http://www.hawestv.com/etv-crts/crt-flehsig/flehsig_1st_color_crt.htm

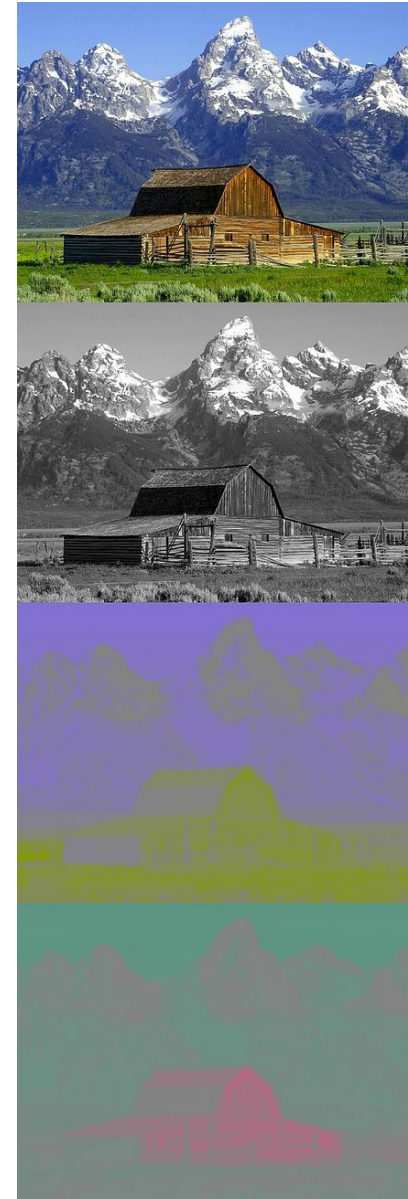
How to Upgrade video standards, but let black and white displays still work?

- Color TV invented in 40's but took until 70's for color TV to surpass B&W TV in sales
- How do you do it? Can't send out R, G, and B signals since old TVs won't know what that is
- Still must send out old signal
- Remap our 3D RGB color space into something else!



YCrCb (sometimes YUV)

- Color space composed of three values:
 - Y: Luminance
 - Cr: Red Chrominance
 - Cb: Blue Chrominance
- Together they can represent the full color space



Full color

Y

Cb

Cr

<https://en.wikipedia.org/wiki/YCbCr>

YCrCb $\leftrightarrow$ RGB

- Just one 3-tuple to another (linear algebra)

Figure 1. YCbCr/RGB color conversion

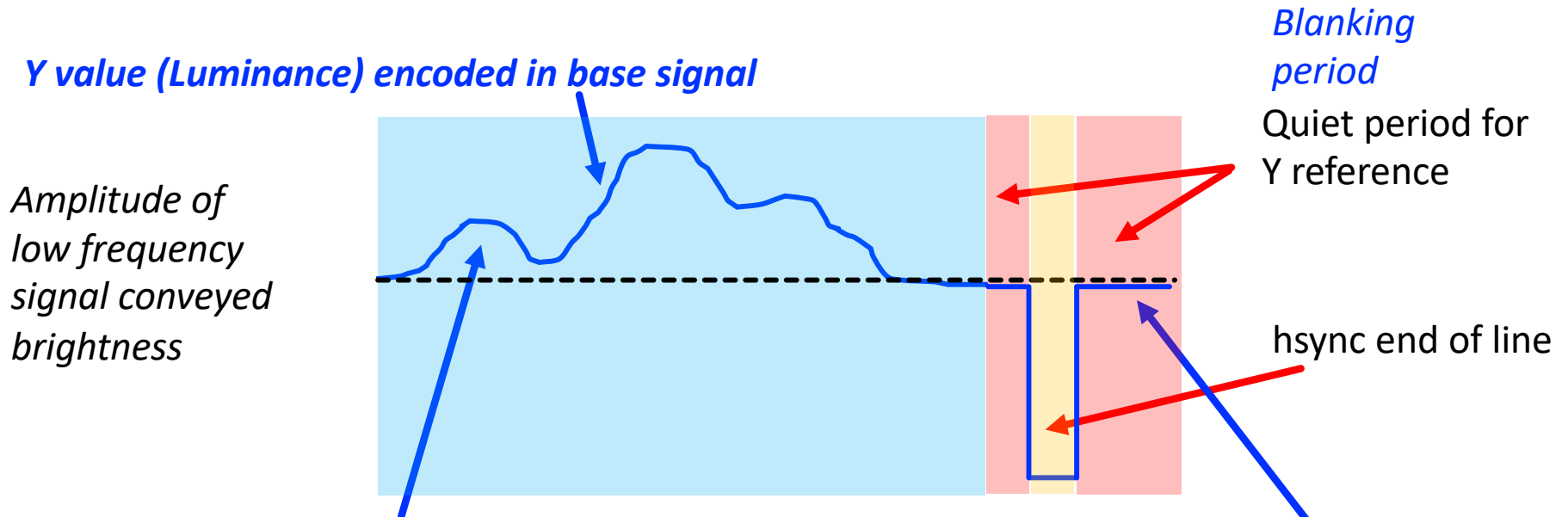
$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.16874 & -0.33126 & 0.5 \\ 0.5 & -0.41869 & -0.08131 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}$$
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.77200 & 0 \end{bmatrix} \begin{bmatrix} Y \\ Cb - 128 \\ Cr - 128 \end{bmatrix}$$

YCrCb $\leftrightarrow$ RGB

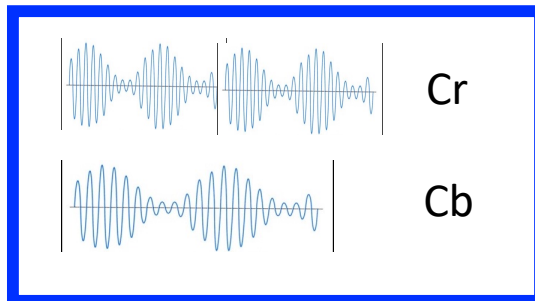
- 8-bit data
 - $R = 1.164(Y - 16) + 1.596(Cr - 128)$
 - $G = 1.164(Y - 16) - 0.813(Cr - 128) - 0.392(Cb - 128)$
 - $B = 1.164(Y - 16) + 2.017(Cb - 128)$
- 10-bit data
 - $R = 1.164(Y - 64) + 1.596(Cr - 512)$
 - $G = 1.164(Y - 64) - 0.813(Cr - 512) - 0.392(Cb - 512)$
 - $B = 1.164(Y - 64) + 2.017(Cb - 512)$
- Implement using
 - Integer arithmetic operators (scale constants/answer by 2^{11})
 - 5 BRAMs (1024x16) as lookup tables for multiplications

Color Analog Video signal

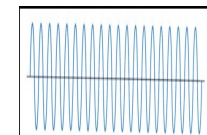
- Keep signal the same as before but add other stuff to it!



Superimpose two slightly different sine waves on top of Luminance signal that encode Cr and Cb data (not to scale) in amplitude modulation:



Add a Cr/Cb “color burst” to region of blanking period to calibrate for Cr/Cb
Amplitude Demodulation



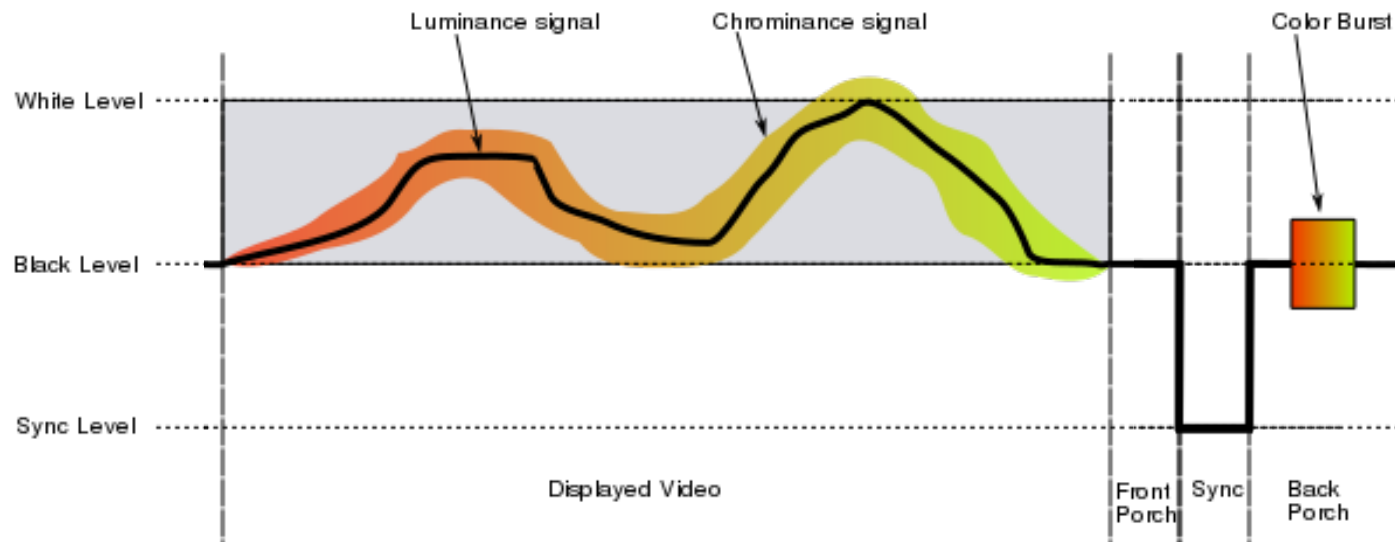
Backwards Compatibility

- Because the Luminance is all that old TVs and displays responded to anyways, you could transmit full-color video and not have it break backwards compatibility!
- Super nice of them! This is just like when Apple updated the cables on their phones and then made sure to...oh...oh wait.
- Or when Apple knew that their Apple Intelligence needed a lot of memory to run so they made sure to not enable it by default on older Macs during updates...oh...oh wait

Composite Video Encoding:

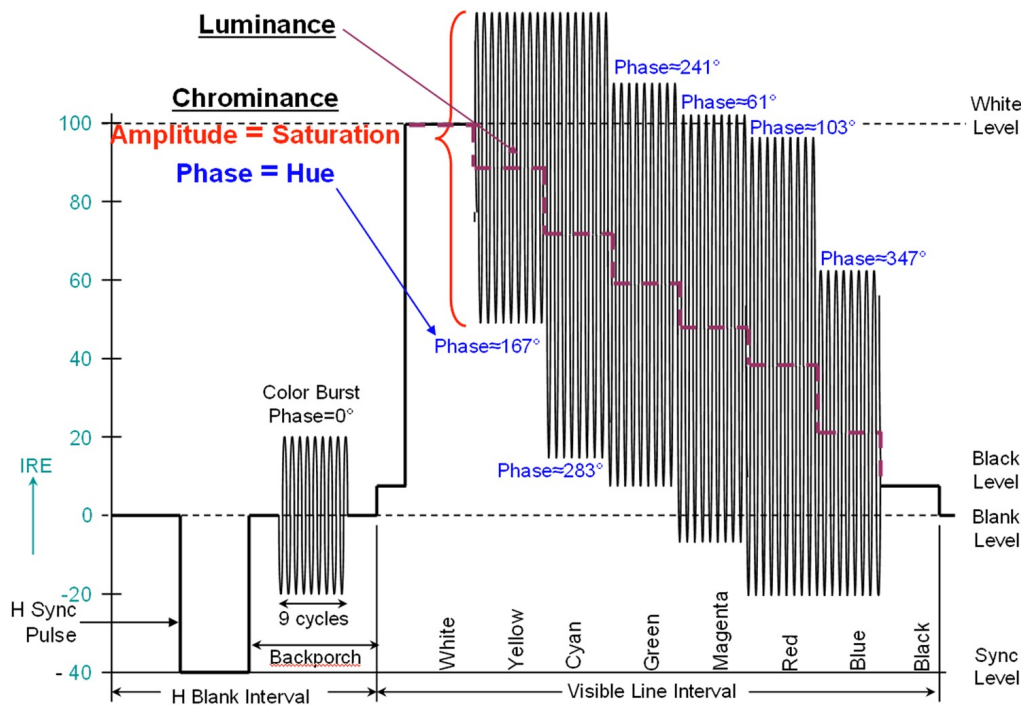
Used for most color TV transmissions and component video up until early 2000's

Use colorburst to remind receiver frequency and amplitudes for interpreting luminance and chrominance signal correctly



Encoding Color

- If you do math out, the two chrominance signals construct/deconstruct to form a signal where:

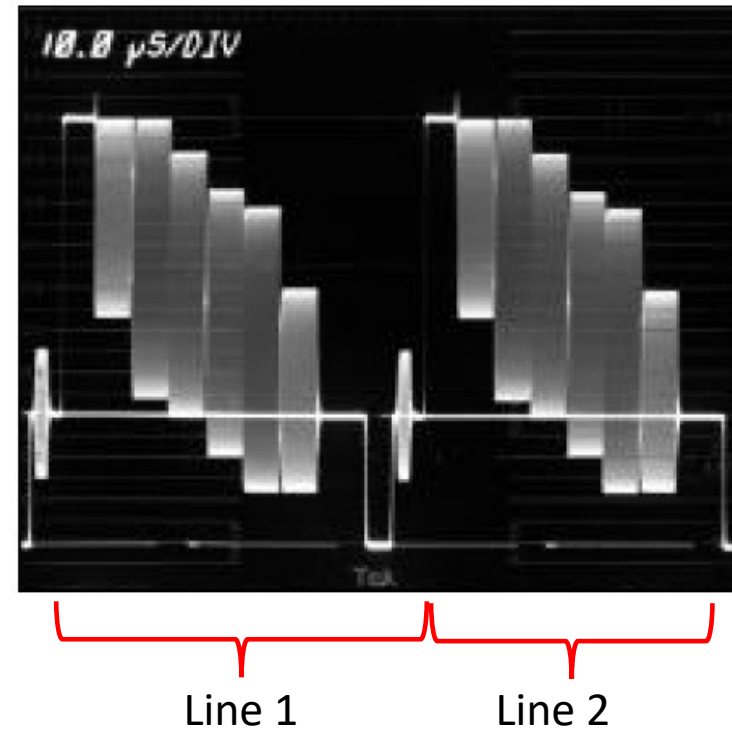
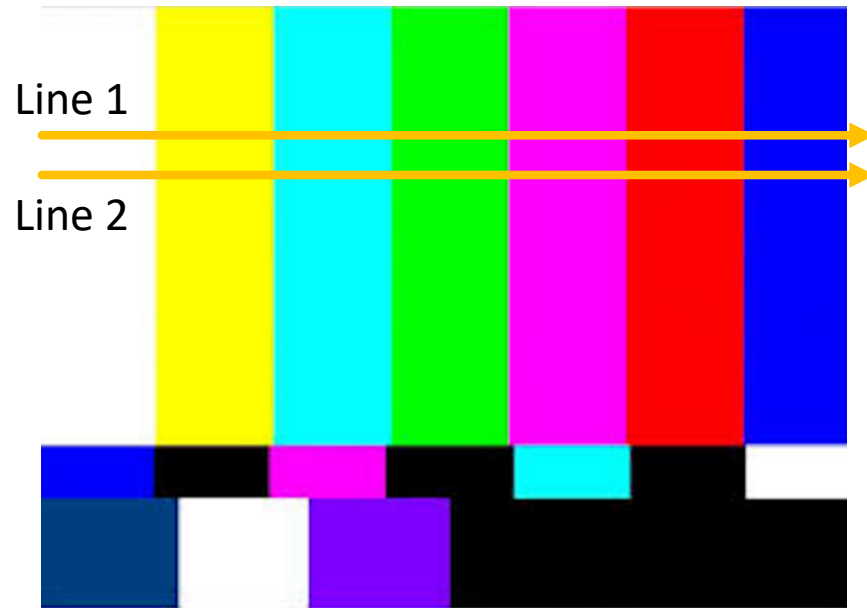


- Amplitude is **Saturation**
- Phase is **Hue**
- **Luminance** is low-freq original value
- Hue, Saturation, Luminance (HSL) is a cylindrical color space that is used a lot!

https://www.eetimes.com/document.asp?doc_id=1272387#

NTSC*: Composite Video Encoding

Captures on a Scope



Old old Labkits work with Cameras that produce composite video out



Two conductors:

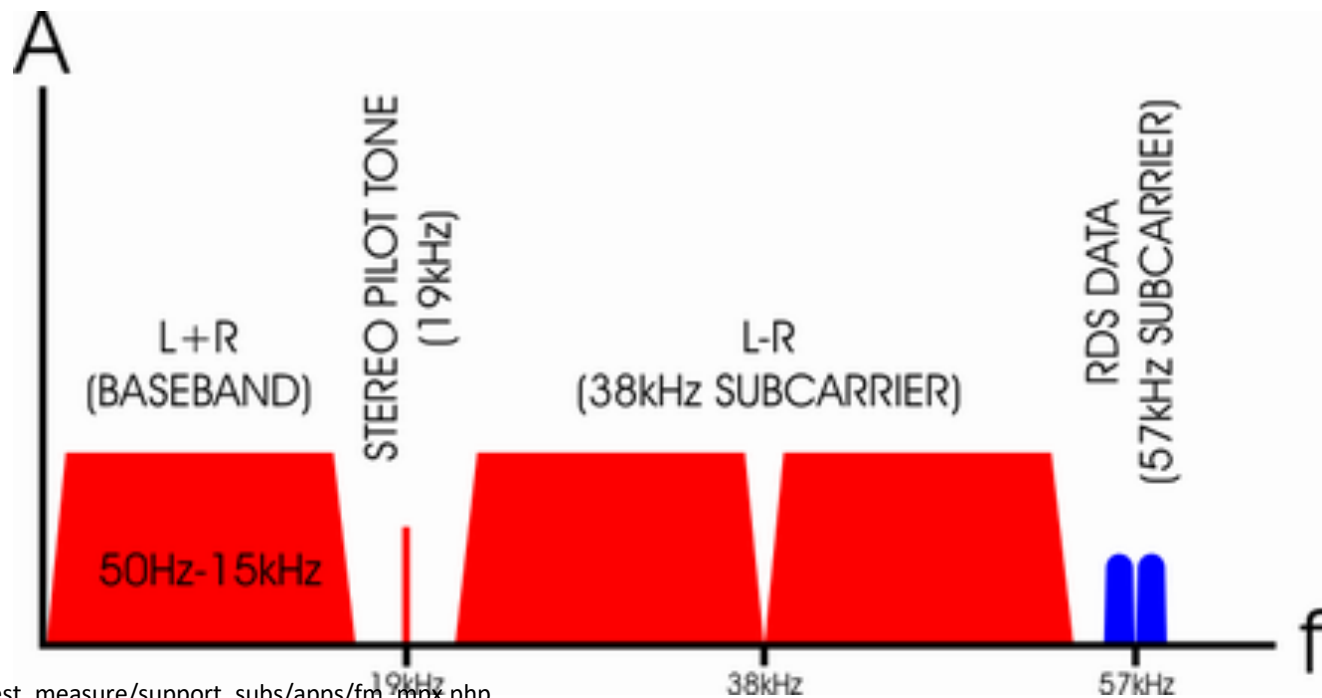
- Shield (ground)
- Middle thing (signal)

Aside: Another Great Backwards Compatibility Achievement

- Prior to 1960s all radio was Mono (one channel played in both left and right audio)
- They wanted to start selling newer radios that could handle Stereo (left and right channels separate), but still support the chumps that already owned older mono-only radios
- How to do?

Re-label your original signal

- And also broadcast a different combination of what you want.



http://www.prismsound.com/test_measure/support_subs/apps/fm_mpx.php

Back to video...

- Continuing on with video history...

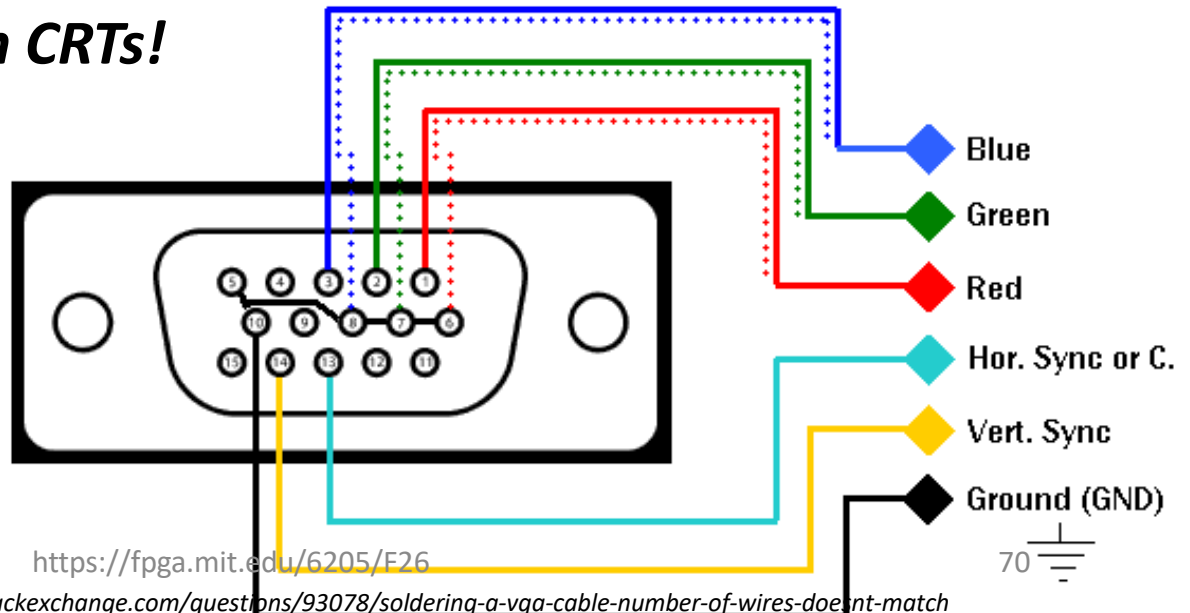
VGA (Video Graphics Array)

- Development of personal computers motivated a rethink of video display!
- IBM (late 1980s)
- Data conveyed primarily **analog**
- *Did not have to be reverse compatible with B/W (chose to use RGB rather than Y Cr Cb as a result)*
- *Used separate wires for different signals (easier)*
- **BUT Still had to deal with CRTs!**
 - *Need blanking!*
 - *Need sync signals!*

Still some backwards compatibility requirements



DB15 Connector



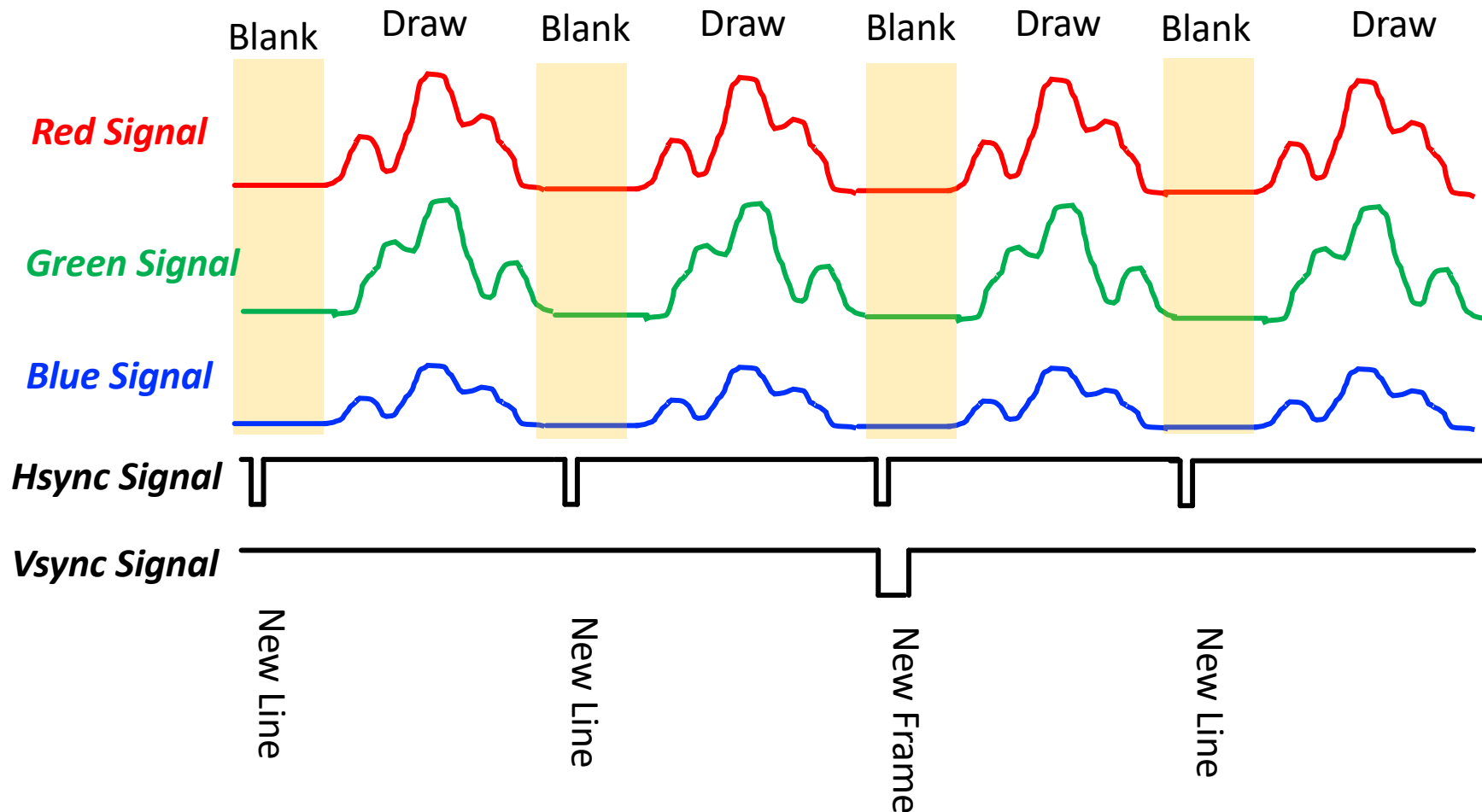
October 1, 2026

<https://fpga.mit.edu/6205/F26>

<https://electronics.stackexchange.com/questions/93078/soldering-a-vga-cable-number-of-wires-doesnt-match>

VGA Signals

- Similar as Before, but split analog signals (easier to interpret as human)



And with VGA...

- We entered into an era of increasing video resolutions.
- Original TV (PAL or NTSC) were in the ~500 or 600 lines of video per image range
- Original VGA was 640x480...but it just blew up from there.

Digital Transmission...

- VGA...and then more recently digital transmission (in a little bit) cut down on noise significantly and allowed an expansion of greater and greater resolutions (all of which just required the transmission of more and more data quickly)

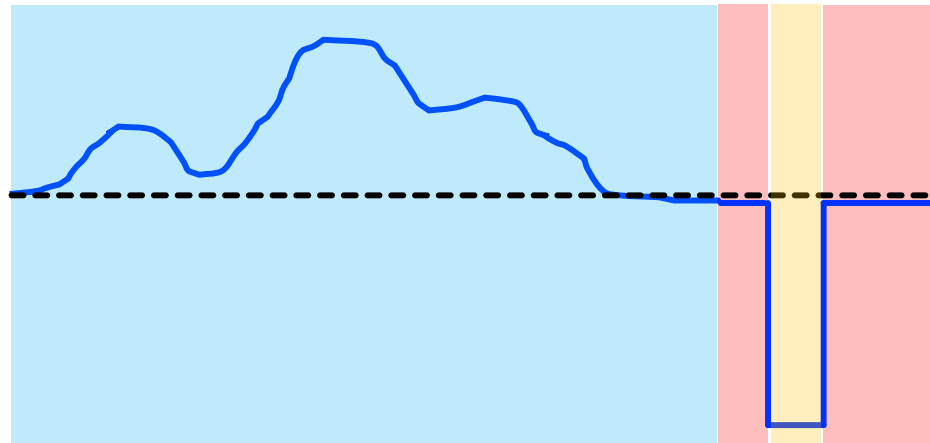
Figure out Display Resolution

Generally need to draw 60 frames per second regardless of resolution(can go faster):

Resolution	Pixels	Aspect Ratio	Products
VGA	640x480	4:3	
SVGA	800x600	4:3	
XGA	1024x768	4:3	iPad, iPad Mini
SXGA	1280x1024	4:3	
Crappy HD TV 720p	1280x720	16:9	6.205 F26
HD TV	1920x1080	16:9	
iPhone 6 Plus	1920x1080	16:9	
iPhone 16 Plus	2796x1290	16:9	
iPad Retina	2048x1536	4:3	iPad Air, iPad Mini Retina
Macbook Retina	2560x1600	16:10	13" Macbook Pro
Kindle Fire	1920x1200		HDX 7" (3 rd Generation)
4K HD TV	3840x2160	16:9	
8K UHD TV	7680x4320	16:9	Really expensive TVs
16K ?HD TV	15360x8640	16:9	Reeeeeeally expensive TVs (2025)

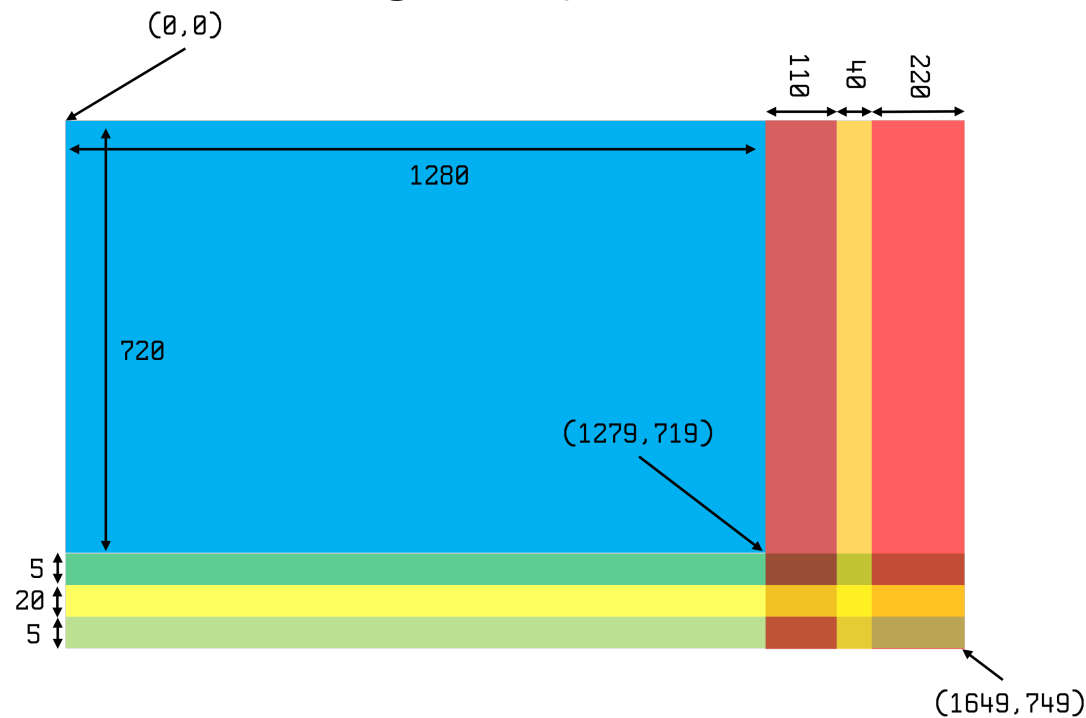
720p

- In lab this week we're going to create 720p video.
- The images are 1280 x 720 pixels in size (where 720p comes from)...not full story though...
- We still have to draw like this:



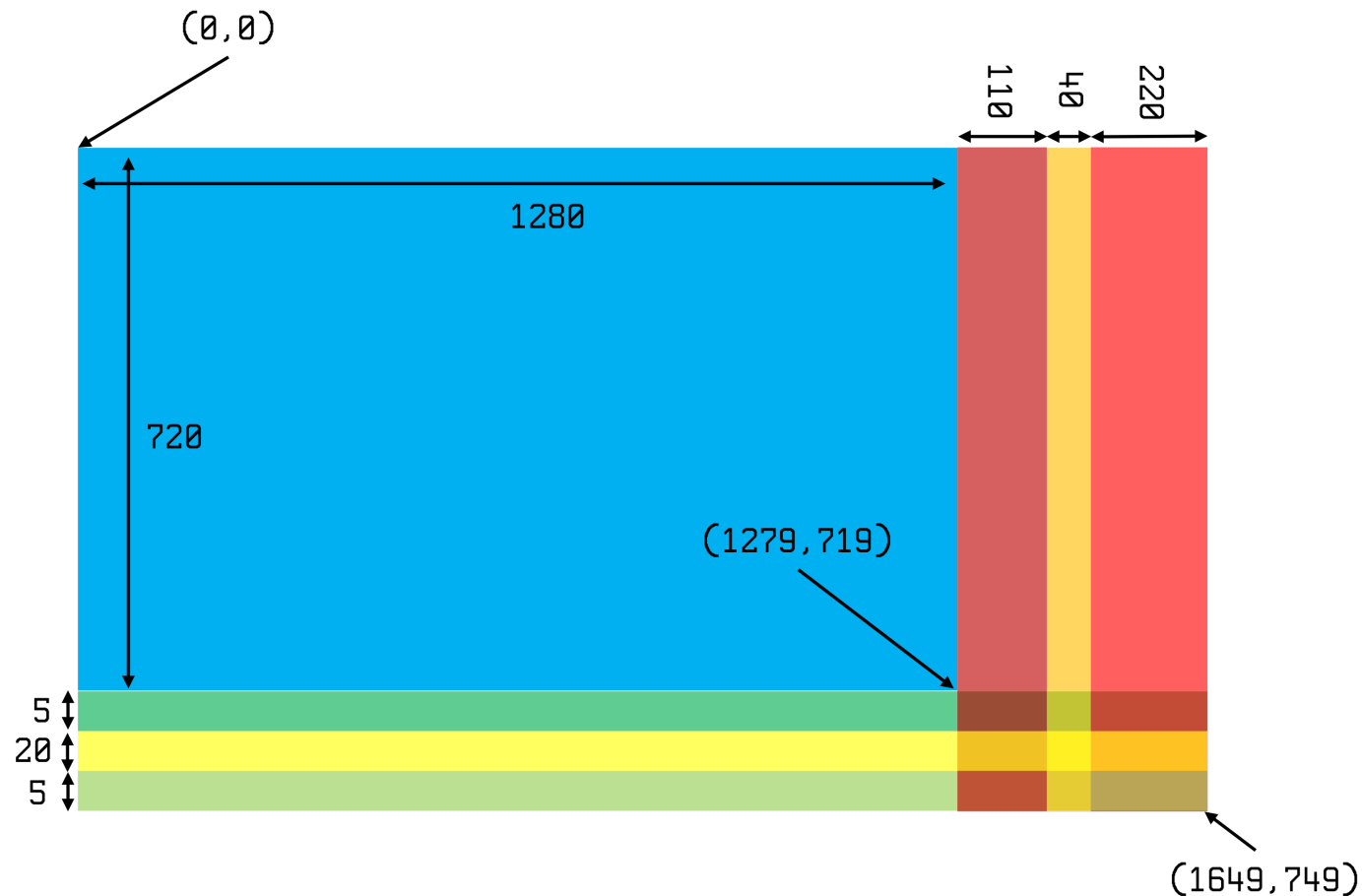
720p

- In lab this week we're going to create 720p video.
- The images are 1280 x 720 pixels in size (where 720p comes from)...not full story though...
- All video standards have particular sizes associated with all their parts (not just the active drawing area!)



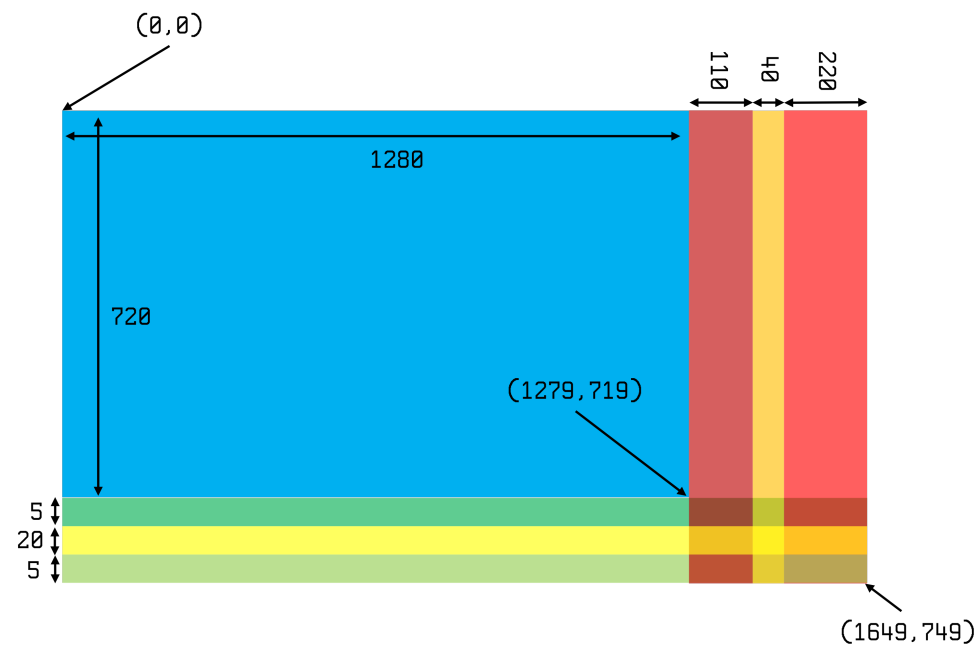
720p Timing

- The dimensions of a 720p frame are shown below including blanking and sync periods



How Big is this Frame?

- 1650 pixels wide
- 750 lines tall
- So 1.2375 million pixels per frame.
- About 75% is meant for drawing... the rest is blanking/sync



How Many Pixels Per Second?

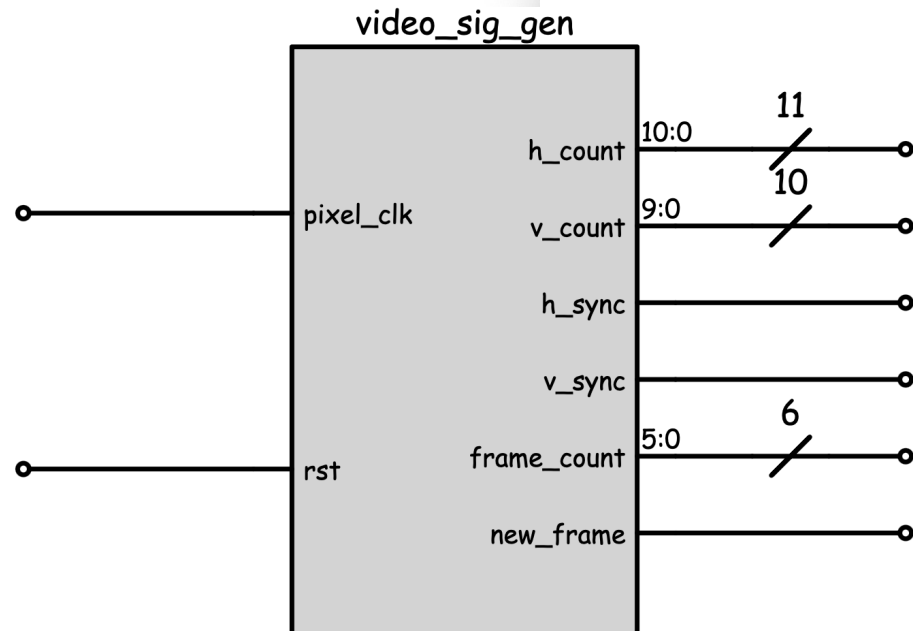
- We'll be generating 60 fps 720p video. That means we need to deliver 60 full frames per second.
- If 1.2375 million pixels per frame
- 60 frames per second...
- We need to deliver 74.25 Million pixels per second.
- The clock we drive our entire system at then is based off of this frequency.
- We'll use a MMCM/PLL to generate a 74.25 MHz clock from 100 MHz. (previous lecture)

Week 04 Part 1

- You'll generate the full raster pattern control signal for 720p

- `h_count`: The current horizontal count on the screen.
- `v_count`: The current vertical count on the screen.
- `h_sync`: The horizontal sync signal, high when in the horizontal sync region
- `v_sync`: The vertical sync signal, high when in the vertical sync region
- `active_draw`: The active drawing indicator, low when in blanking or sync period, high when actively drawing.
- `new_frame`: Single cycle indicator of a new frame (see below)
- `frame_count`: Current frame with a rolling second-long window (ranges from 0 to 59 inclusive)

*Drive with a pixel clock of
74.25 MHz*



Modern Displays and Technologies

VGA is dead, Joe. Also nobody uses CRTs anymore. My computer only has HDMI and a Display Port and I use an OLED display because I'm bad like that. All that you've said is irrelevant.

History

- Display technologies and all the associated protocols are a classic example where prioritization of backwards compatibility has really affected decisions going forward.
- We still use the same general pattern of digital transmission mainly because lots of things assume that pattern and nobody wanted to break old stuff and there was very little advantage to deviating.

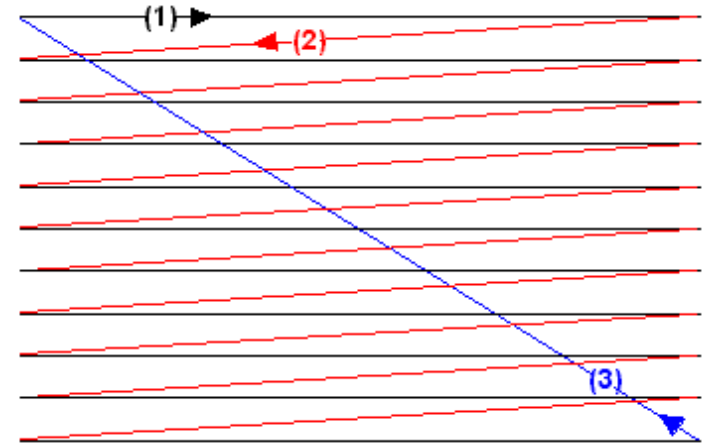
Modern Displays

- Organic Light Emitting Diode (OLED) Displays
- Liquid Crystal Display (LCD)
 - requires backlight source,
 - constant power
- Cathode Ray Tube (CRT)

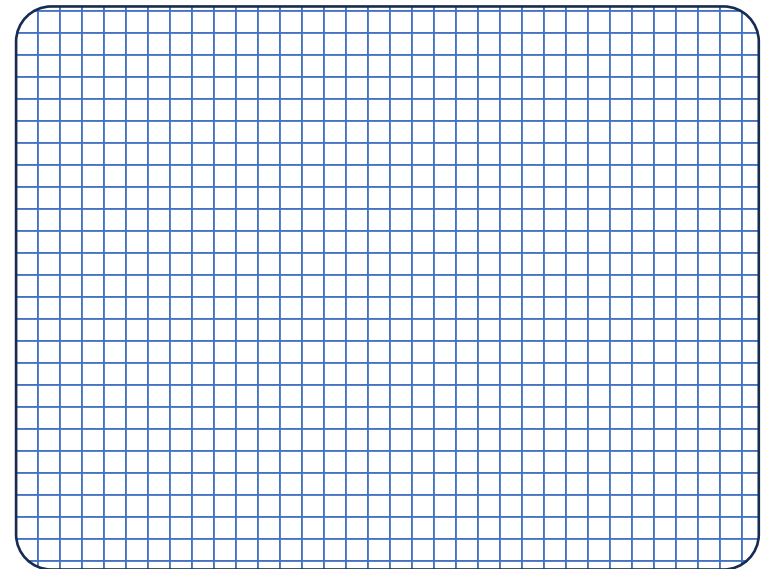


*Back in
Time*

Old Vs. New



- We don't scan a beam of electrons anymore, but we still have individually-controlled pixels
- Generally use shift buffers from the sides to feed in pixel data as it arrives serially (in order)
- “bucket brigade”

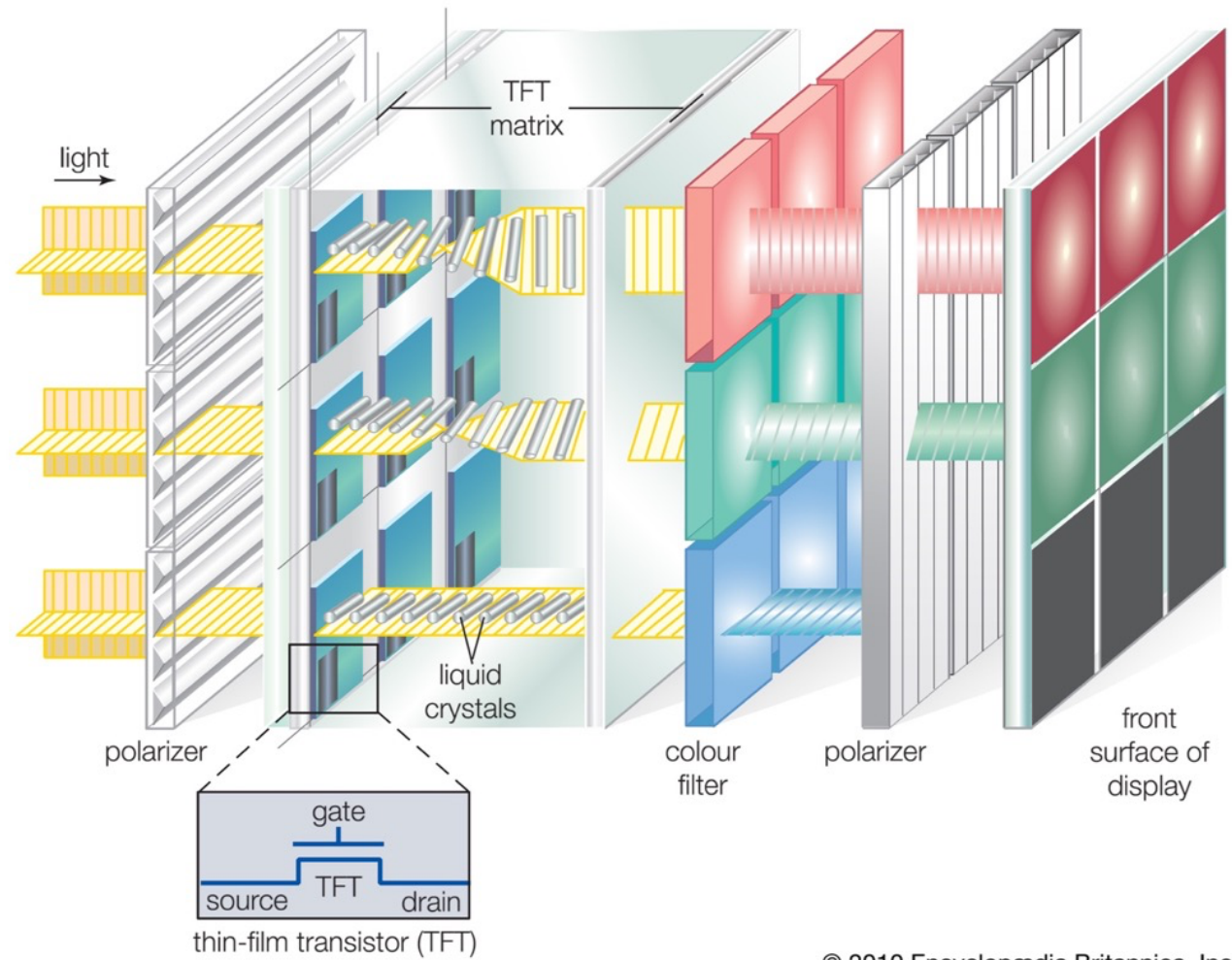


TFT (Thin-Film Transistors)

- Older Technology (now):
- Make a display:
 1. Gigantic white backlight (polarized)
 2. Gigundous array of voltage-variable polarizers (TFTs with Liquid Crystals) (let light through at rest)
 3. One TFT for each color (RGB), three per pixel
- Want black pixel? Turn TFT fully on to block light getting through

TFT LCD

*Used to be Cold Cathode
Now almost always white LEDs*

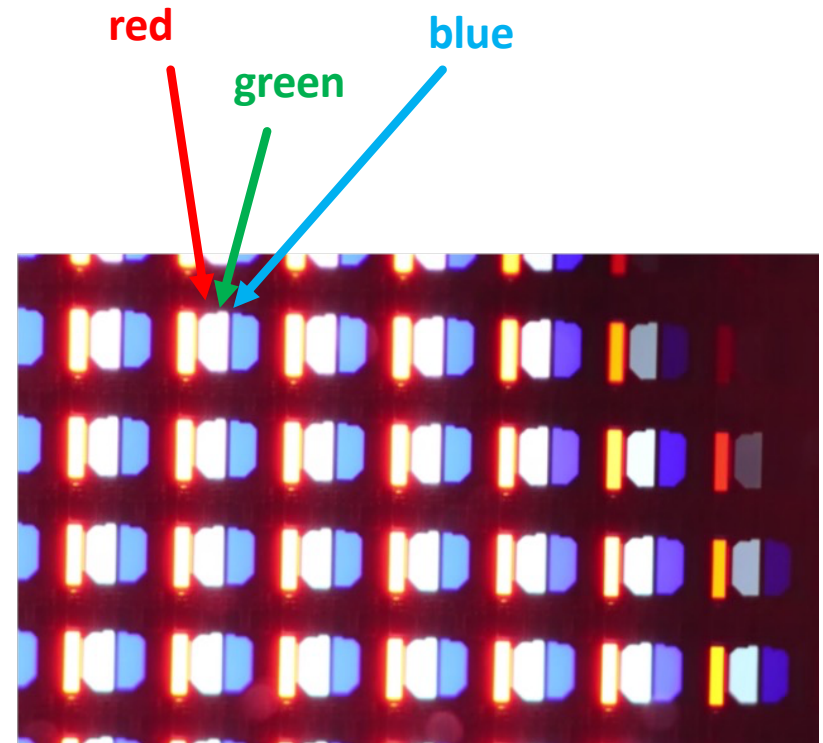


© 2010 Encyclopædia Britannica, Inc.

liquid crystal display: active-matrix TFT liquid crystal display. Art. Encyclopædia Britannica Online. Web.

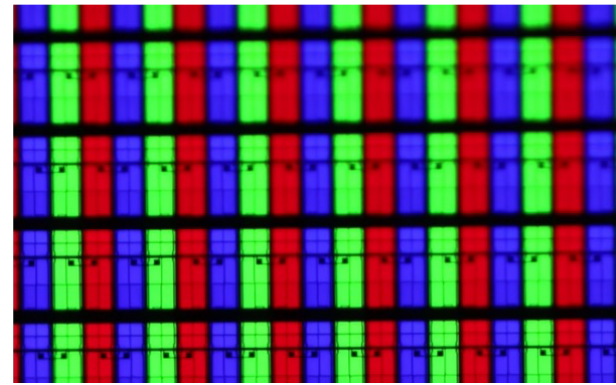
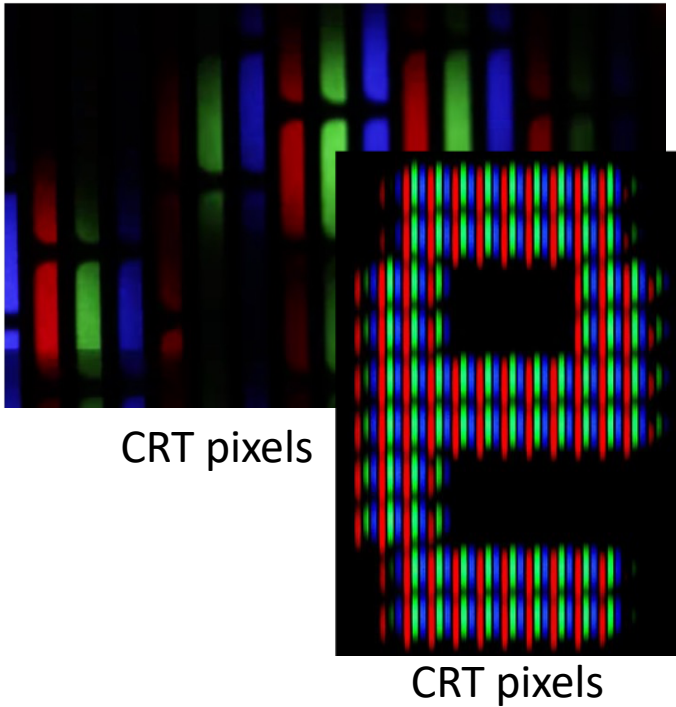
Organic Light Emitting Diodes

- Newest Technology
- Conceptually maybe the simplest/ideal way to do a display
 1. Gigundous array of RGB LEDs
 2. Control RGB amt. at each point
 3. Profit
- 1. Want black pixel? Just don't turn on LED



***Green saturated in this image**

All Color Displays use RGB Pixels



TFT LCD pixels



OLED pixels

Slo-Mo Guys

<https://www.youtube.com/watch?v=3BJU2dr rtCM>

- Video Locations:

- CRT @ 2:13
- TFT LCD @ 7:58
- OLED @ 10:50



- Whole video is a good watch

Now...How to Send Video Data Digitally?

- The great benefit in moving from analog to digital is improved noise performance.
- The great sacrifice in moving from is you now need to send your symbols much faster to make up for the loss of information in any one symbol.

Before..

- We make video by sending images **serially**
- We make images by sending lines **serially**
- We make lines by sending pixels **serially**
- We encode pixels as **analog voltages**

Now..

- We make video by sending images **serially**
- We make images by sending lines **serially**
- We make lines by sending pixels **serially**
- We make pixels by sending 1's and 0's **serially**

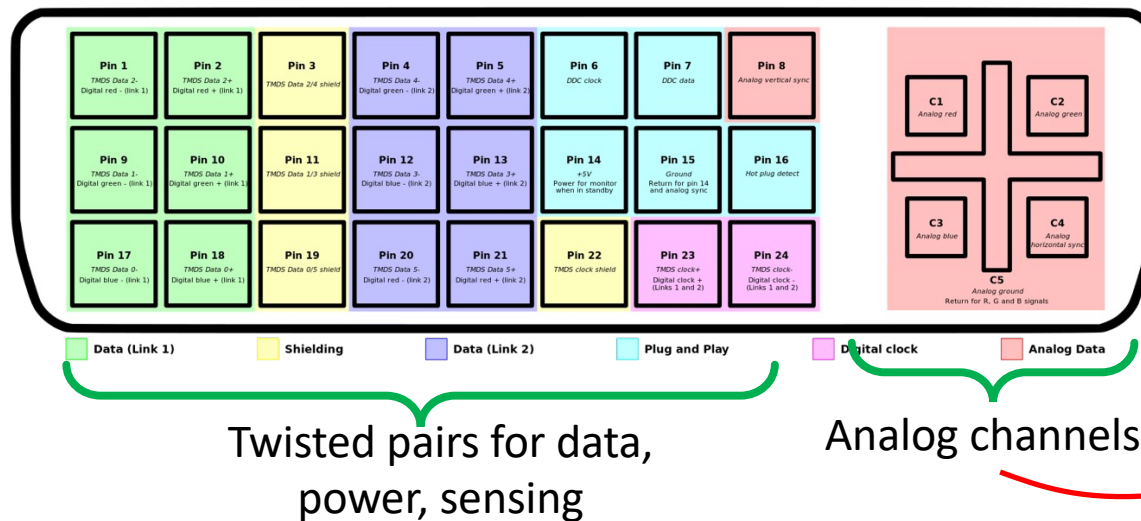
So we have to make electronics that work extremely fast...

DVI (Digital Video Interface)

- 1998ish
- Backwards compatible with VGA to an extent (supposed to support analog)
- However new way to send data digitally over twisted pairs in high-level structure similar to VGA



DVI Cable



Sneaky-snake....the cable shown here is not backwards compatible

DVI → HDMI ?

- HDMI took the open standard of DVI that only focused on video and added a bunch of extra stuff onto it like sound, encryption, avenues for spyware.
- HDMI is proprietary.
- DVI is not.
- Technically in this class we're **making the DVI portion** of HDMI *_not_* true proper HDMI so we're mostly good (plus not selling stuff)
- We'll call it HDMI since we're using the connector, but it is not the same thing.

HDMI

- To use/sell HDMI devices, you need to pay 5K a year and then:
 - 15 cents per device you sell OR
 - 5 cents per device if you make the logo visible OR
 - 4 cents if you enable both the content protection functionality AND have the log somewhere
- I don't even know who HDMI is anymore:
 - Hitachi → Lattice → Invecas → ADI → Cadence → Illuminati? → lunno



HDMI

- It all starts with the cable and connector

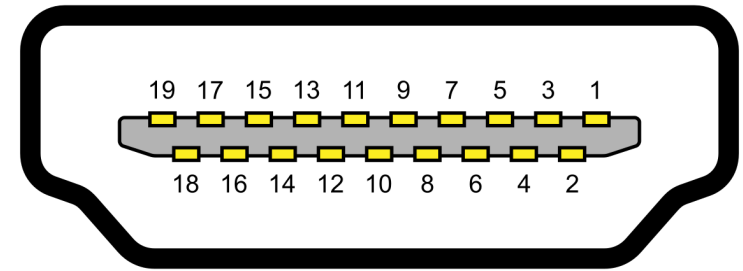


Table 1. HDMI

Pin Number	Assignment
1	Data2+
2	Data2 shield
3	Data2-
4	Data1+
5	Data1 shield
6	Data1-
7	Data0+
8	Data0 shield
9	Data0-
10	Clock+
11	Clock shield
12	Clock-
13	CEC
14	Not connected
15	SCL
16	SDA
17	Ground
18	+5V
19	Hot-plug detect

- You've got three pairs* of wires that carry color
 - Channel 0: Blue
 - Channel 1: Green
 - Channel 2: Red
- Clock Channel
- Few other wires:
 - Resolution info (SCL,SDA)
 - CEC (control things)
 - Power
 - Hot Plug Detect

*each group is a differential signal pair and shield

<https://www.maximintegrated.com/en/app-notes/index.mvp/id/4306>

HDMI Data Transfer

- Modern displays are built around sending the red, green, and blue signals on their own channels.
- Each channel sends that portion of a pixel's information serially.
- For 720p we're sending 74.25 million pixels per second in order to deliver 60 frames per second.
- If we were to do the 8 bits of serially that means the red, green, and blue channels would be sending 594 million bits per second.
- And that's for 720p (pretty “bad” HD TV now)
- This is potentially too much data

High Speeds

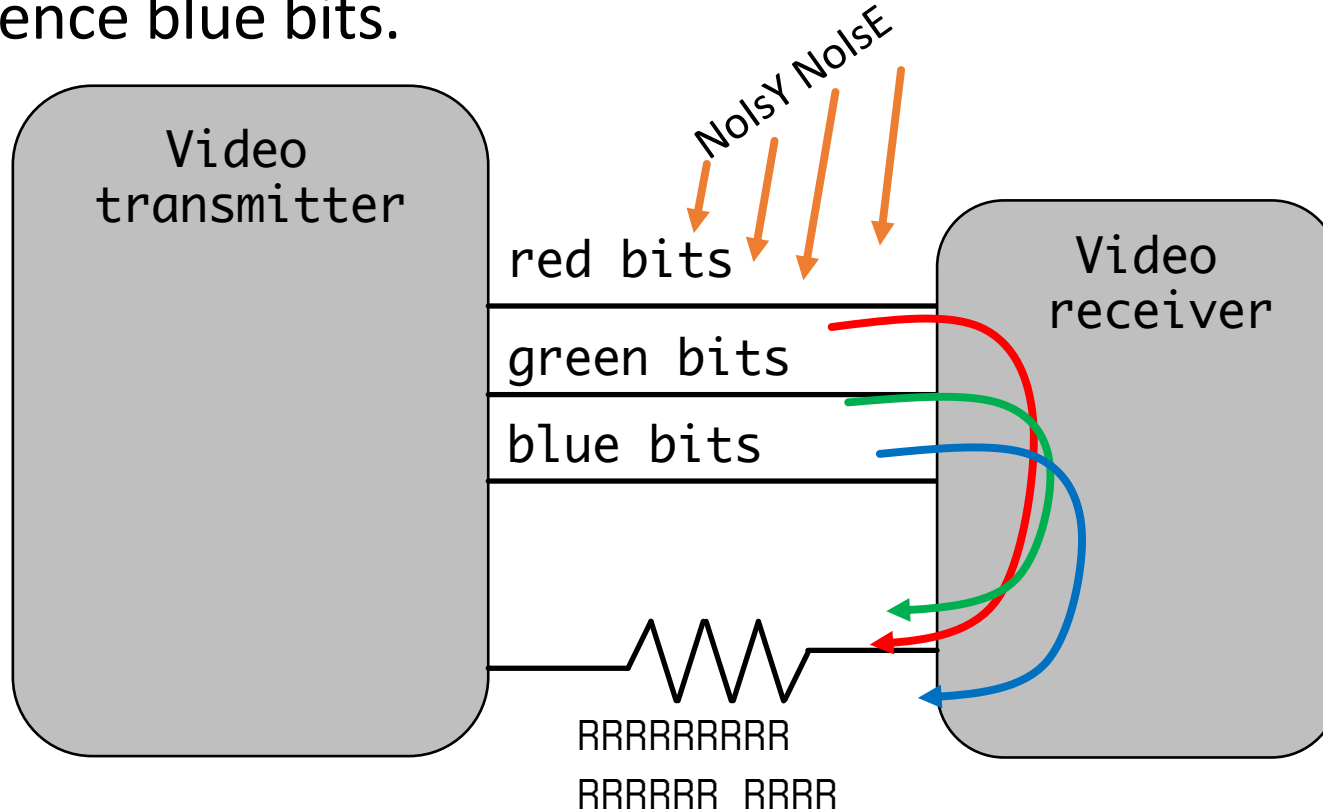
- Sending 1's and 0's down a line at 594 MHz will produce a ton of electrical noise.
- Every $0 \rightarrow 1 \rightarrow 0$ transition is a charge/discharge and released electromagnetic noise...this can cause interference and prevent the red, green, blue and other things from working all at once.
- You can't do it. You need to figure out a way to send the same bits of information but without so many bit transitions. Must have *Transitions Minimized*

High Speeds

- Sending 1's and 0's down a line at 594 MHz will produce a ton of electrical noise.
- Every $0 \rightarrow 1 \rightarrow 0$ transition is a charge/discharge and released electromagnetic noise...this can cause interference and prevent the red, green, blue and other things from working all at once.
- We fix this problem by sending even more bits...
- (pause for curious students to point out that makes no sense)
- *But doing so intelligently... (part of Week 4)*

Long Distances

- Sending large volumes of data over long cables is also very very prone to noise.
- And a common return path to ground for multiple signals usually results in a lot of interference causing red bits to influence blue bits.

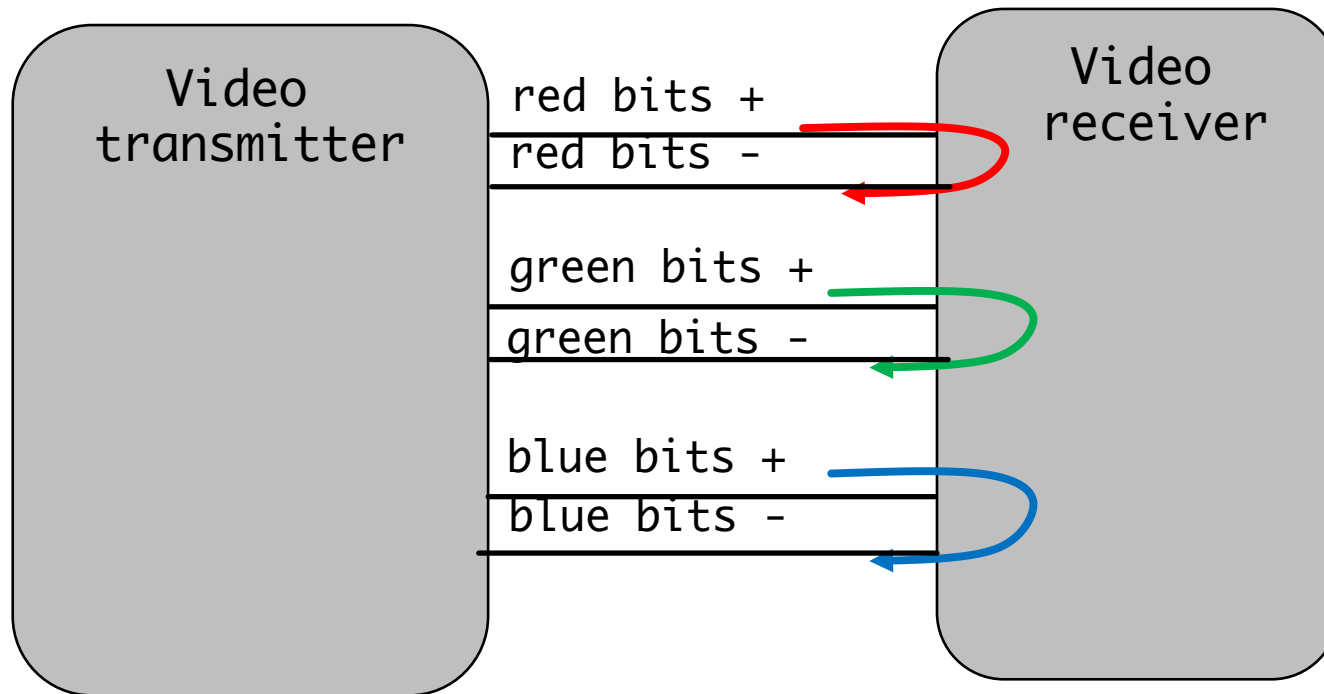


Long Distances

- Instead each data channel gets sent on its own sub circuit comprised of two wires.
- We call this **Differential Signaling**

1 is sent as:
+1/-1

0 is sent as:
-1/+1



HDMI and TMDS

- Transition Minimized Differential Signaling (TMDS) is used to send all data in HDMI
- Instead of sending 8 bits of pixel information we send 10 bits.
- The two extra bits help us:
 - Minimize transitions (using XOR or XNOR encoding)
 - Keep the DC-average voltage on a pair of wires to be about 50% 1's and 0's. which allows recovery circuitry to work on receiver side.

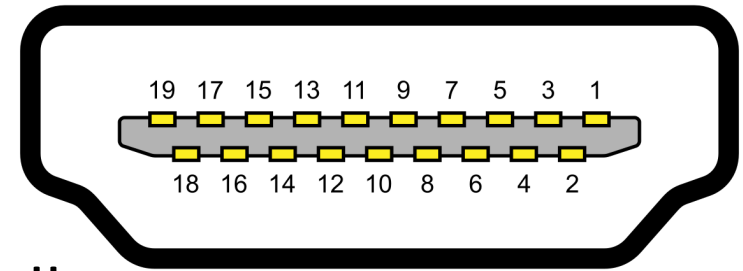


Table 1. HDMI

Pin Number	Assignment
1	Data2+
2	Data2 shield
3	Data2-
4	Data1+
5	Data1 shield
6	Data1-
7	Data0+
8	Data0 shield
9	Data0-
10	Clock+
11	Clock shield
12	Clock-
13	CEC
14	Not connected
15	SCL
16	SDA
17	Ground
18	+5V
19	Hot-plug detect

TMDS Encoding

- There's an easy-to-implement* algorithm to encode using TMDS
- You'll build this in the assignment this week

High-Definition Multimedia Interface Specification

Version 1.3

Table 5-35 Encoding Algorithm Definitions

D	The encoder input data set. D is 8-bit pixel data
cnt	This is a register used to keep track of the data stream disparity. A positive value represents the excess number of "1"s that have been transmitted. A negative value represents the excess number of "0"s that have been transmitted. The expression cnt{t-1} indicates the previous value of the disparity for the previous set of input data. The expression cnt(t) indicates the new disparity setting for the current set of input data.
q_m	Intermediate value.
q_out	These 10 bits are the encoded output value.
$N_1\{x\}$	This operator returns the number of "1"s in argument "x"
$N_0\{x\}$	This operator returns the number of "0"s in argument "x"

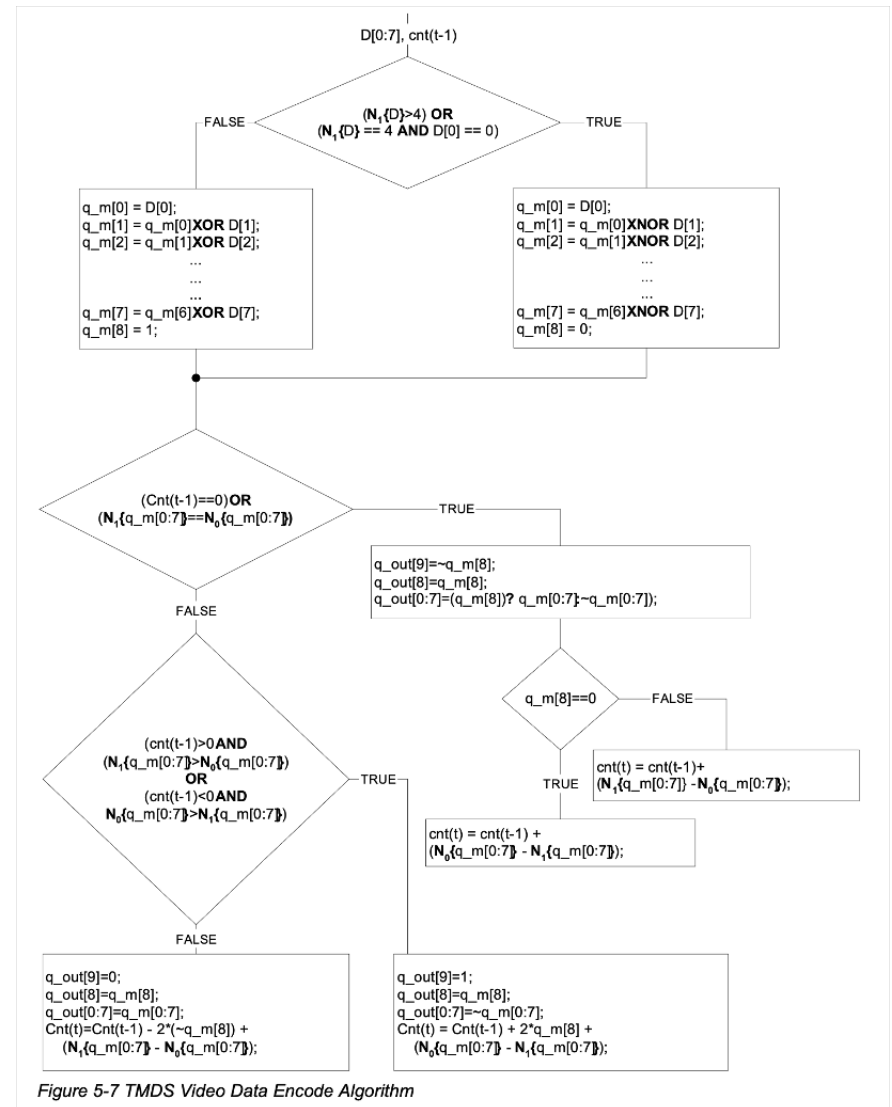
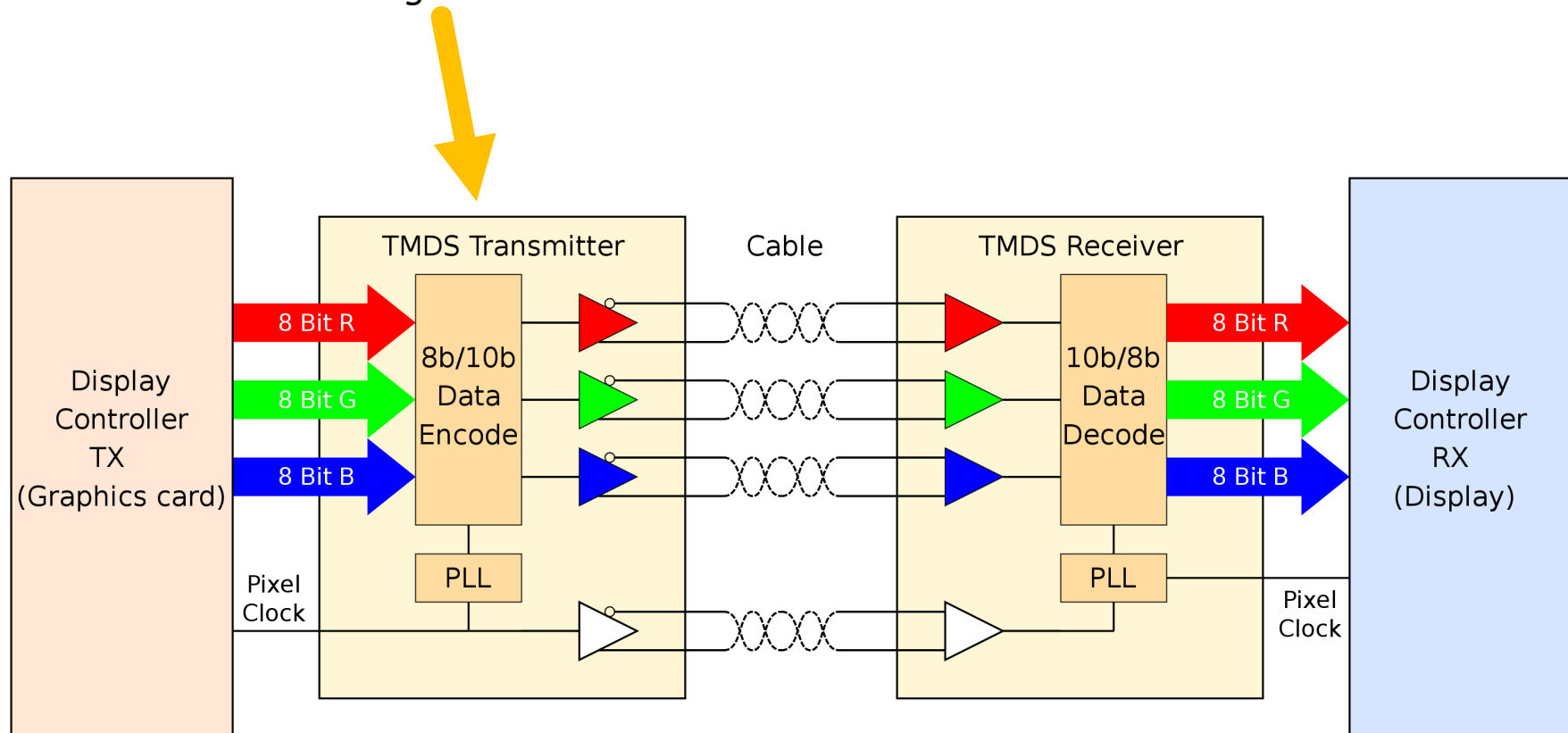


Figure 5-7 TMDS Video Data Encode Algorithm

TMDS system

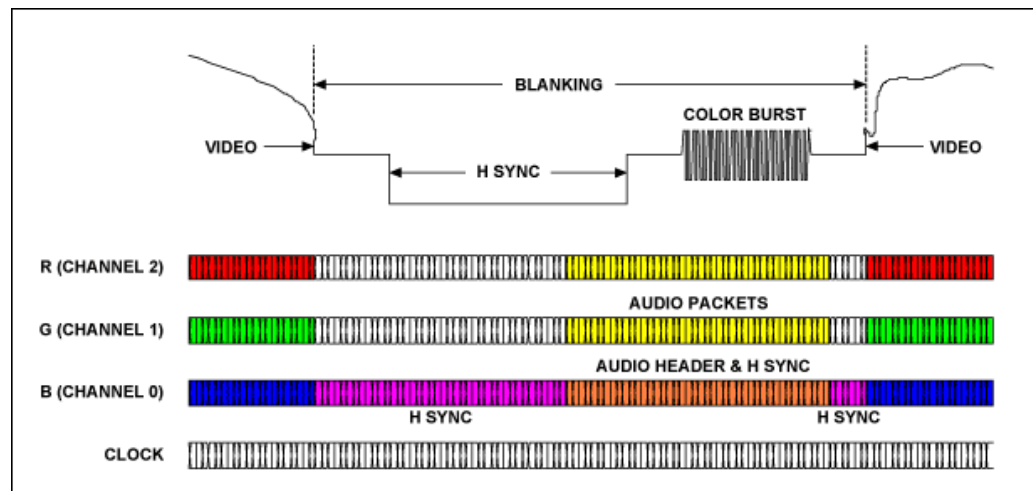
You're making this in week 04



https://en.wikipedia.org/wiki/Transition-minimized_differential_signaling#/media/File:Schematic_TMDS_link.svg

Pixel Color Information

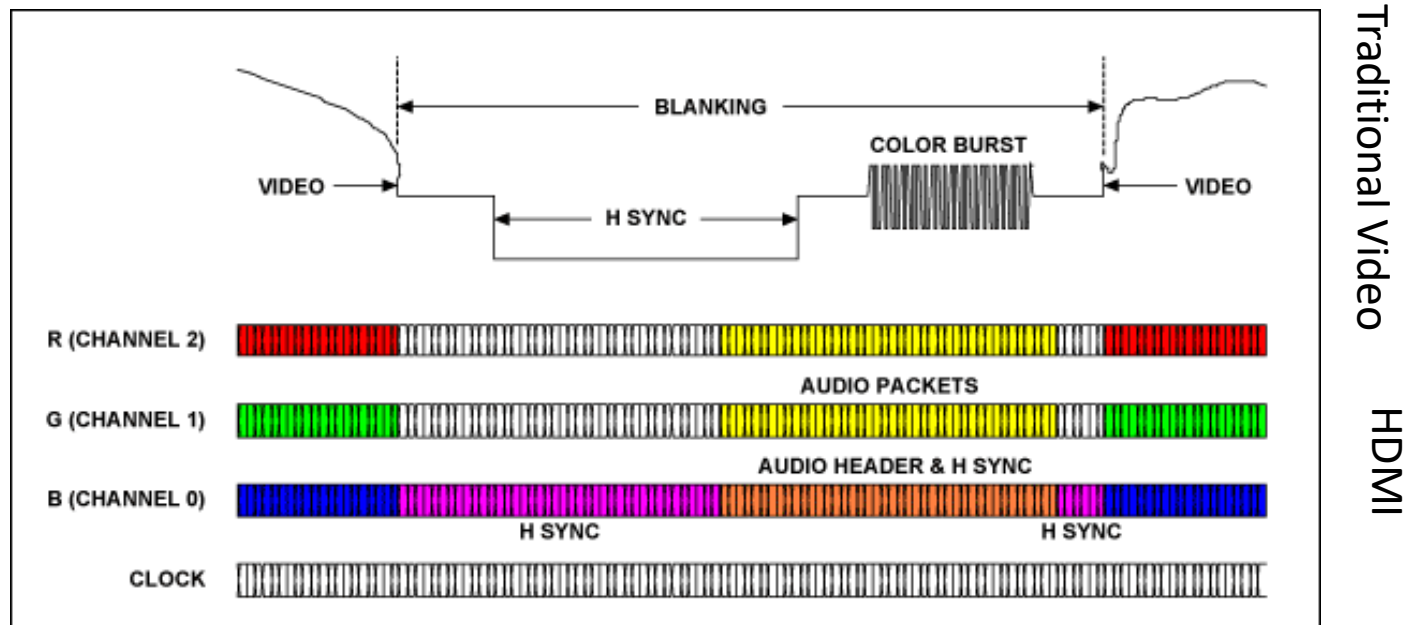
- Sent as serialized data in 10-bit frames using TMDS (week 04)
- One color per pair of wires (red, green, blue wires)
- The blue channel also carries blanking/hsync/vsync info:
 - Encodes those using four 10 bit reserved values:
 - (H = 0, V = 0): 1101010100
 - (H = 1, V = 0): 0010101011
 - (H = 1, V = 0): 0101010100
 - (H = 1, V = 1): 1010101011



One pixel of information per clock cycle (clock is 1/10 bit rate)

Audio Information (HDMI, not DVI)

- During blanking period (when no color needs to be conveyed), there's unused clock cycles on the color lines.
- Shove audio into that region
- Blanking region works out to be about 64 pixels worth of time (64 clock cycles) per line



Audio

- With a screen refresh rate of 60Hz...
- 1080 lines per screen...
- 64 pixels per line (blanking time we have to play with)...
- and 8 bits (of info) per pixel for an HDTV signal...
- The maximum audio information bit rate we could send is:

$$= 60 \times 1080 \times 64 \times 8 = 33.1776\text{Mbps}$$

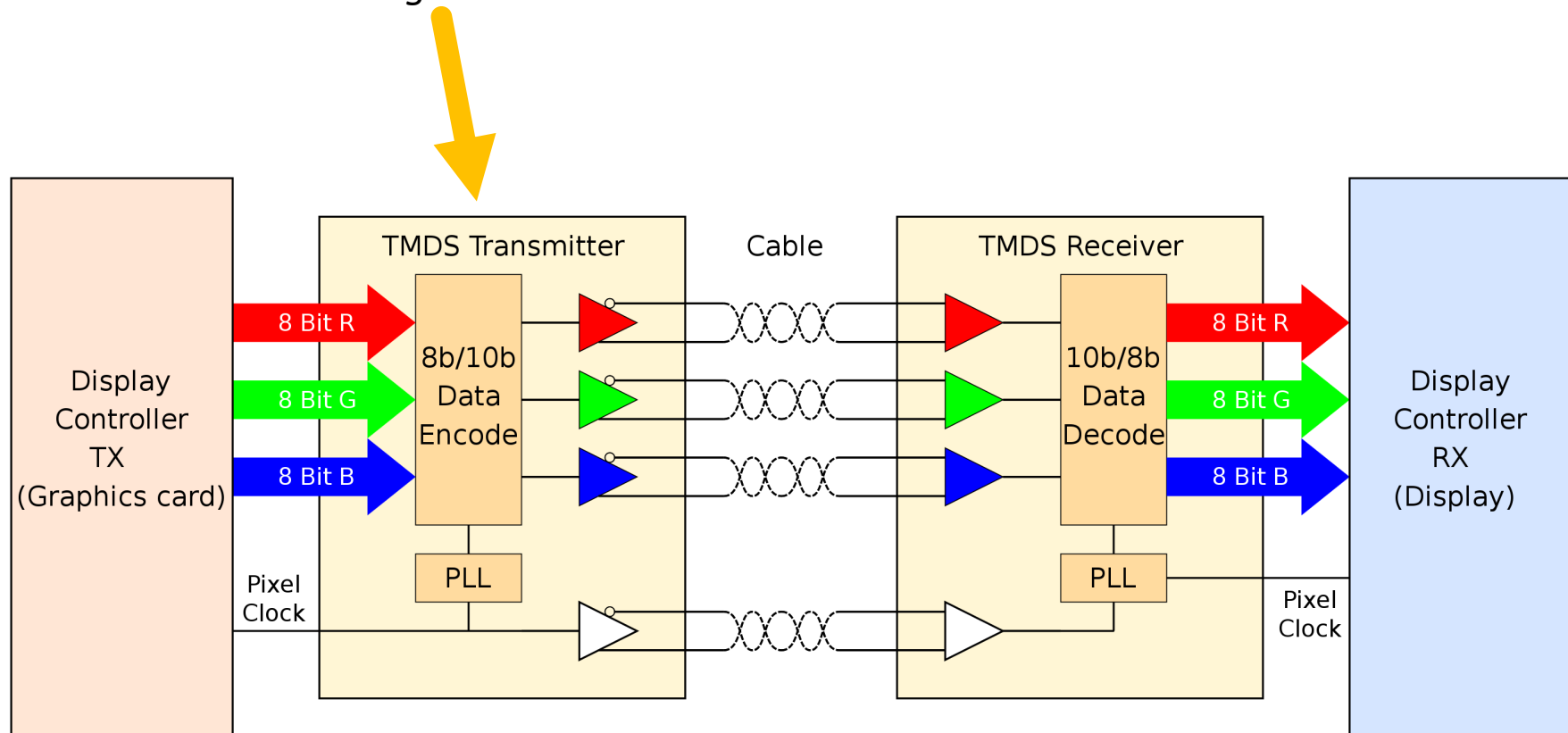
This data rate is more than sufficient to carry any multichannel high-quality audio signals

- (Stereo CD-quality Audio needs 1.411Mbps as a reference)
- Plenty of leftover bandwidth for spyware, malware, the Man telling you what to do, etc...

<https://www.maximintegrated.com/en/app-notes/index.mvp/id/4306>

TMDS system

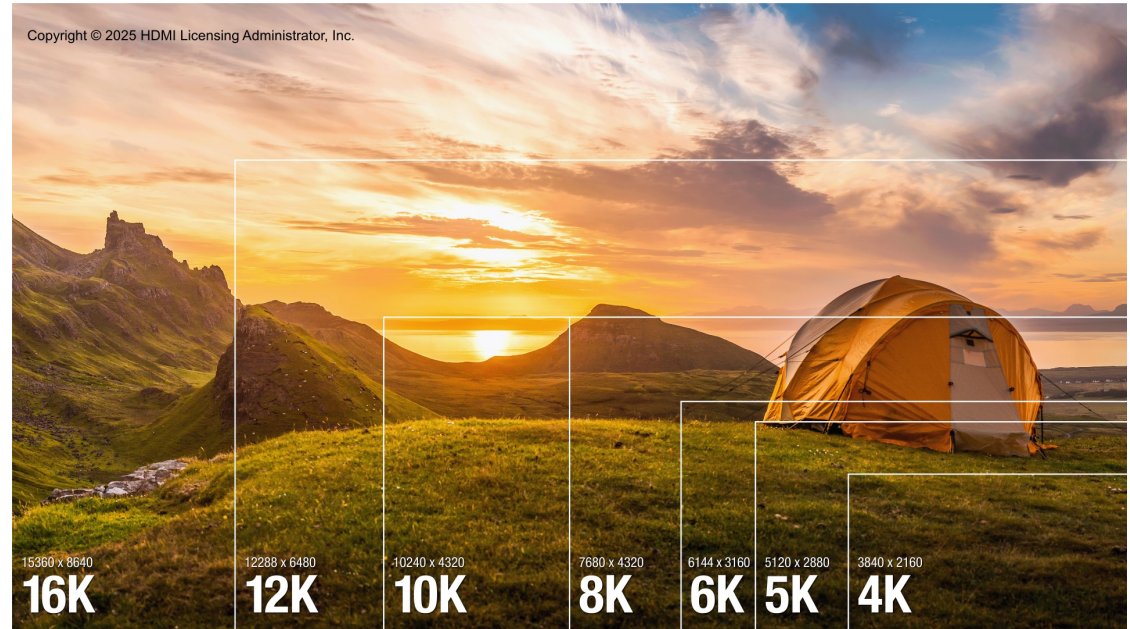
You're making this in week 04



https://en.wikipedia.org/wiki/Transition-minimized_differential_signaling#/media/File:Schematic_TMDS_link.svg

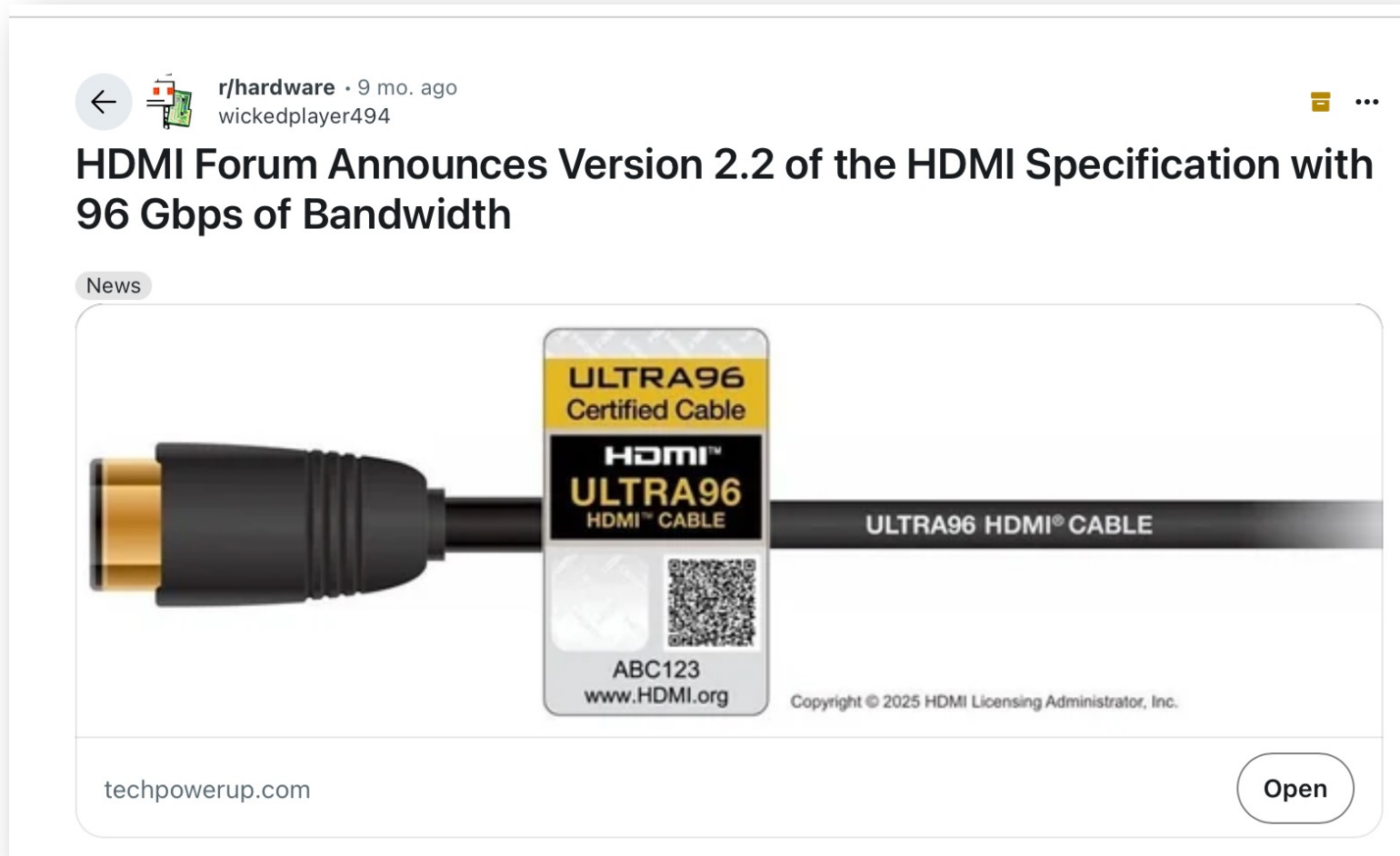
HDMI 2.2

- By varying timing...
- Supports up to 16K



- Supports up to 16K (15360x8640)
 - I can't find anywhere the blanking specs for 16K. Let's assume maybe 80 pixels?
 - So about 8.07 billion pixels per second.
 - For TMDS, data lines need to support 10X that for 8b/10b
 - 80.7 Gbps

Thankfully...



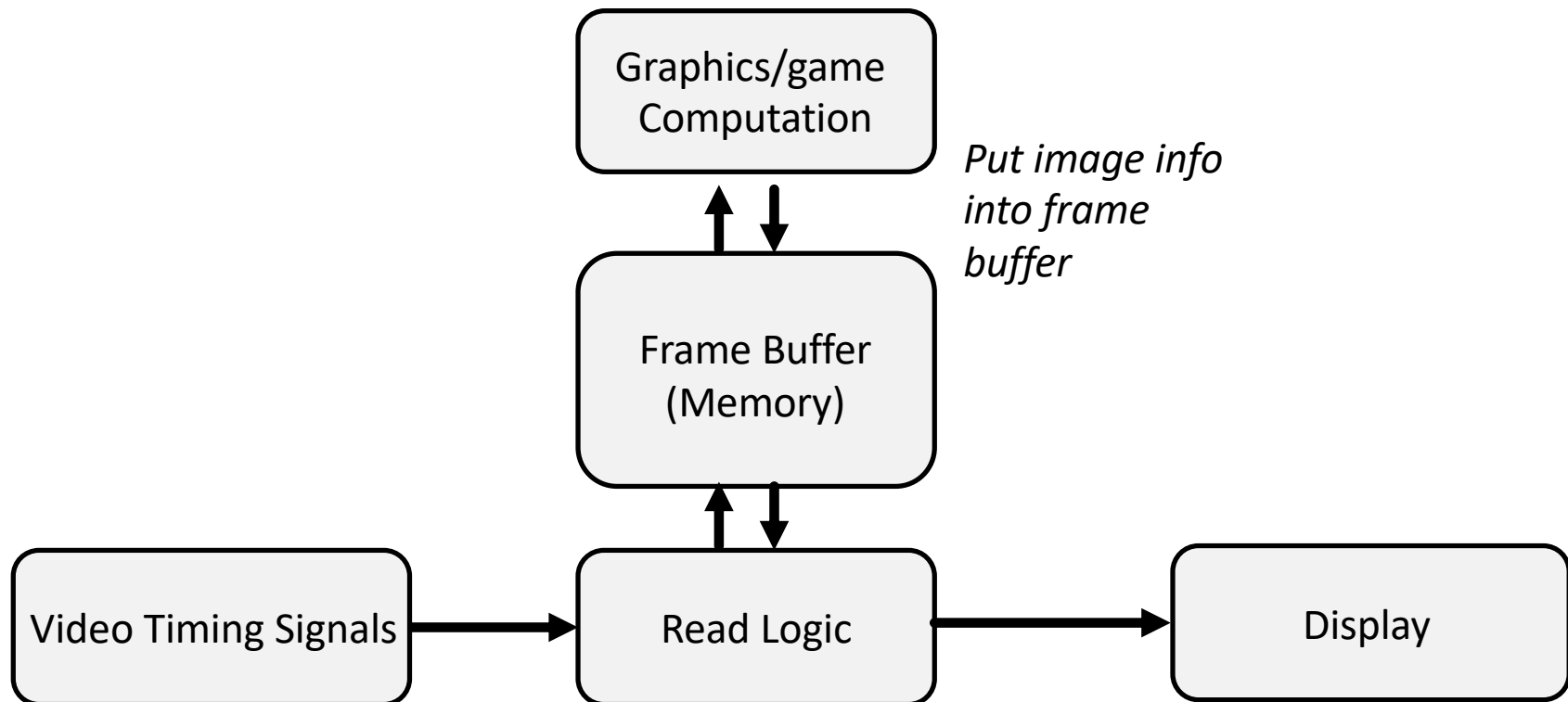
Conclusion

- DVI/HDMI is heavily based off of VGA and is therefore easy to convert.
- Designing for VGA is directly portable (and oftentimes will work without change) for modern video processing (lab kit)
 - Left→Right, Top→Bottom Raster pattern
 - RGB specification of each pixel
 - Blanking (pause periods) where you don't draw and can potentially do heavier calculations if needed!
- Just use different interface circuits and watch your timing!

Generating Video on the FPGA

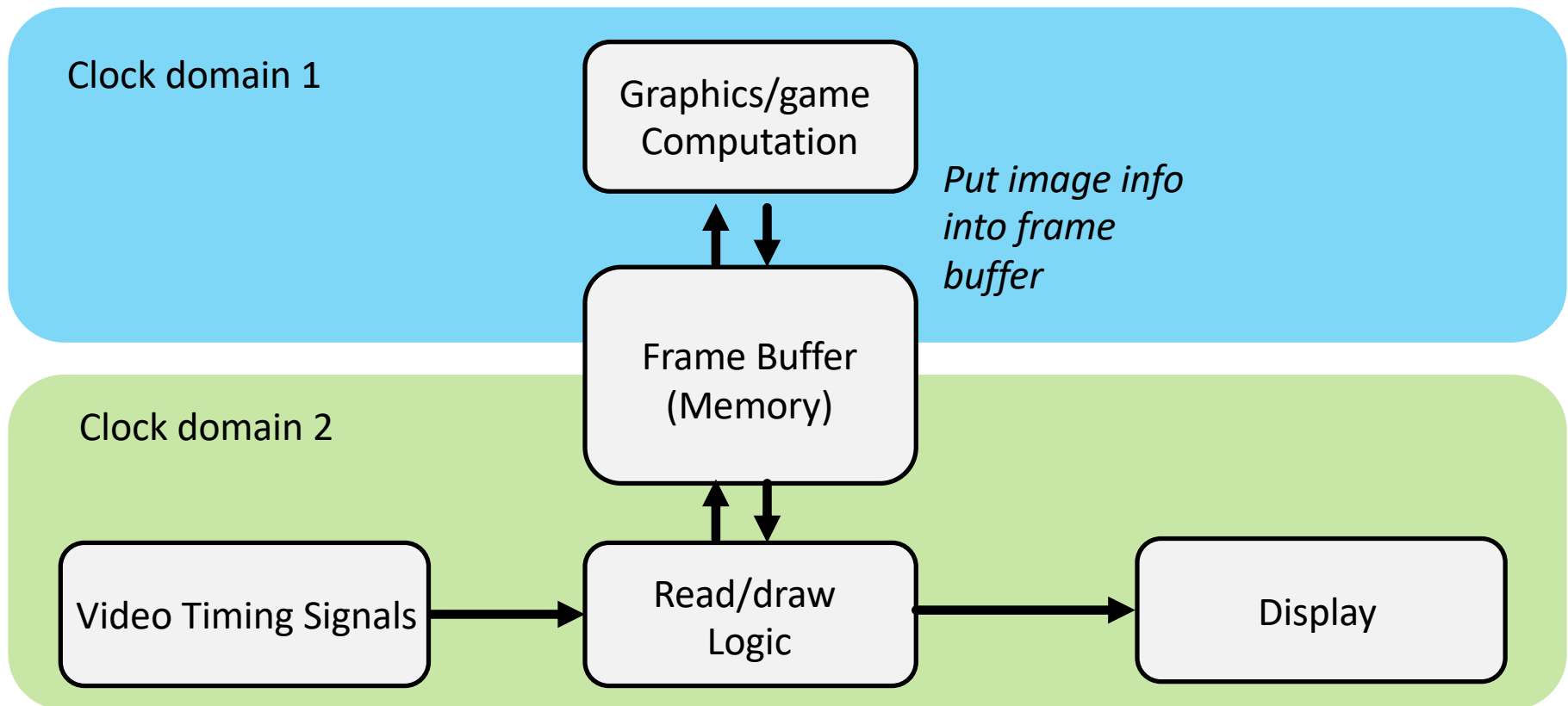
Two General Ways to Produce Video:

- Way One: Frame Buffer
 - Separate computation of drawing from actual rendering of graphics. Use an intermediary memory



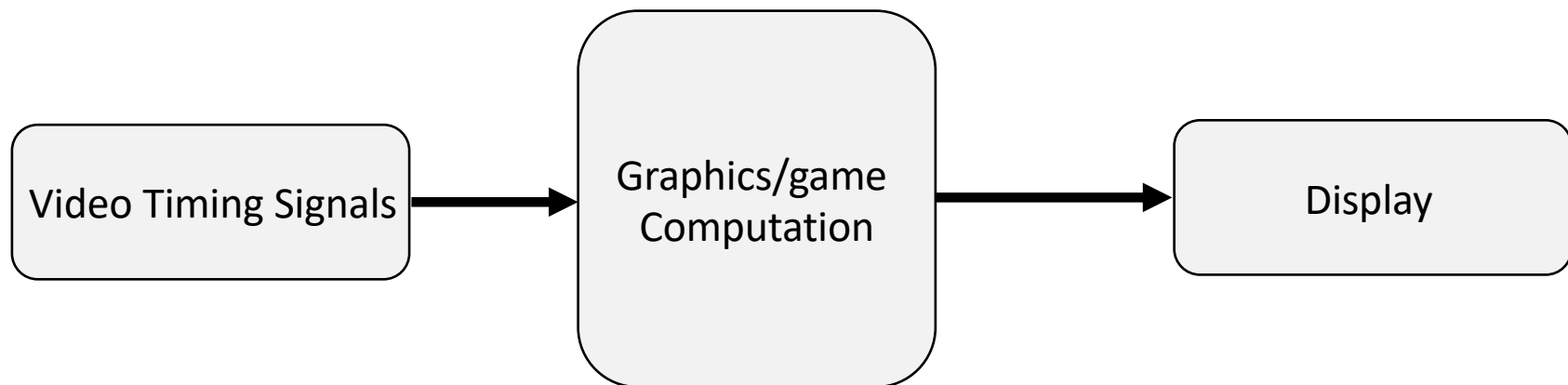
Two General Ways to Produce Video:

- Way One: Frame Buffer
 - Often the Logic will be on a completely different clock domain than the drawing logic

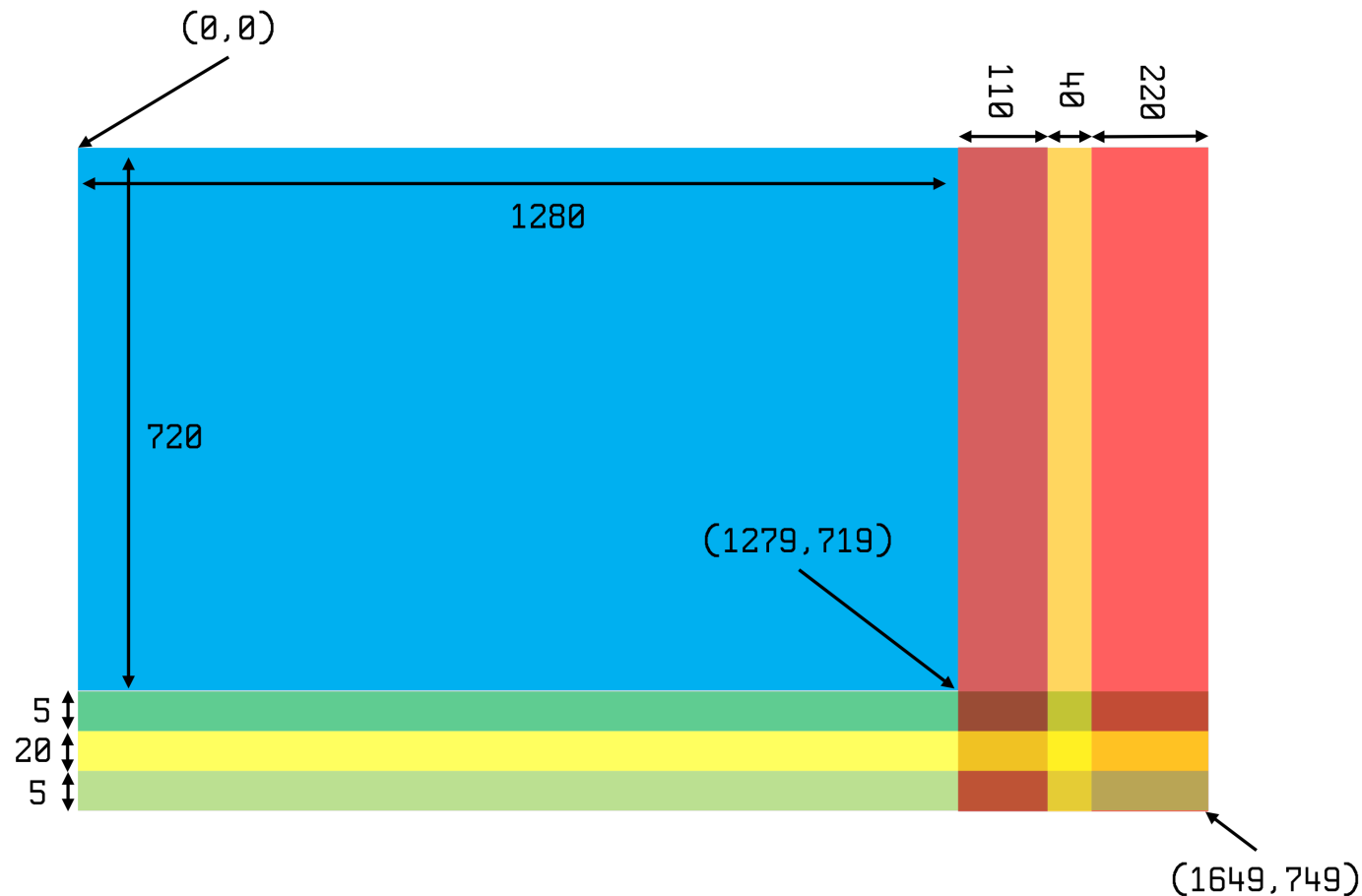


Two General Ways to Produce Video:

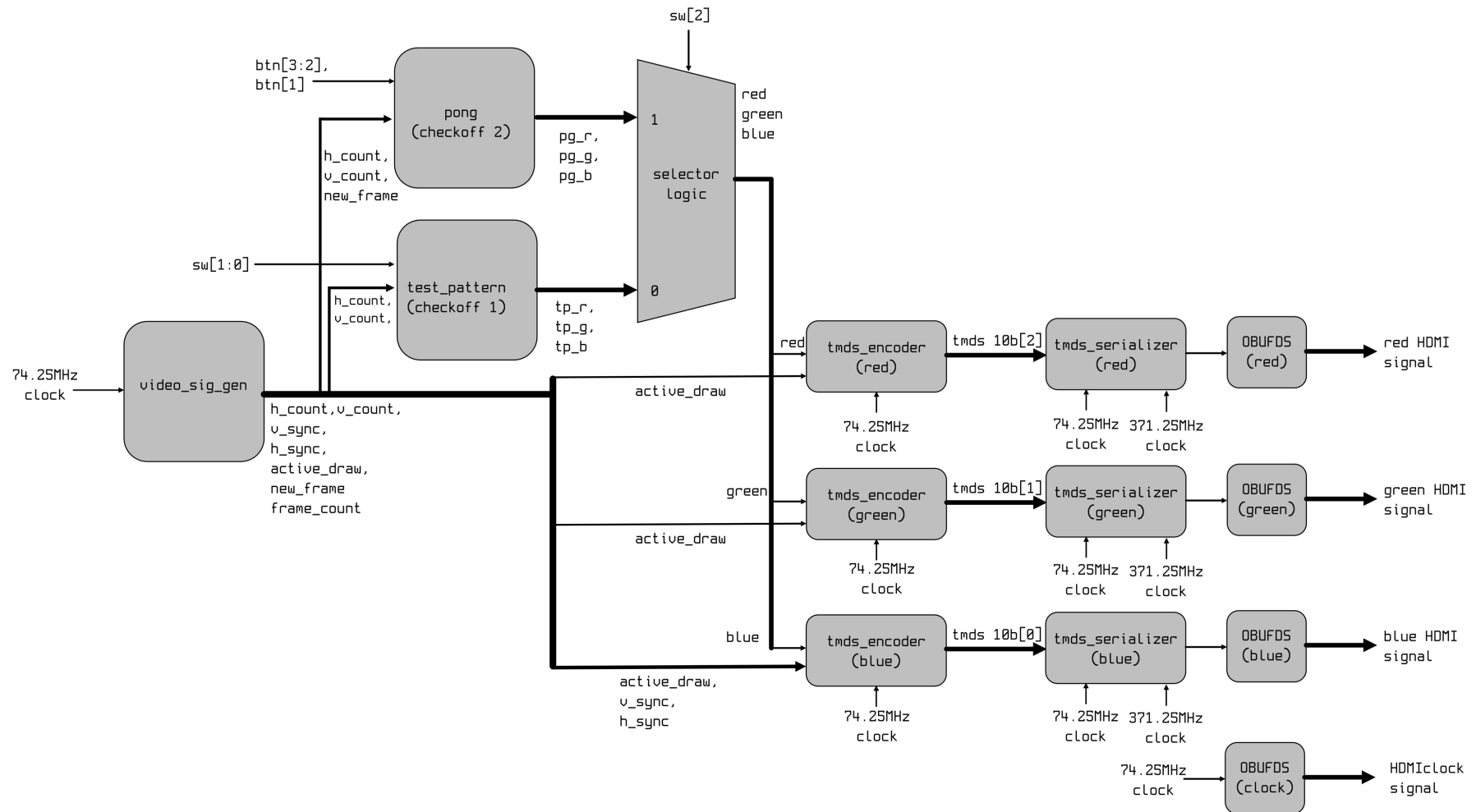
- Way Two: “Racing the Beam” or “Riding the Beam”:
 - Have everything run off of the video timing signals and compute the value of the pixel *just in time* when needed!
 - *How a lot of early video games and other things were done (and other niche applications today).*
 - *All computation (game logic and render logic) must be done on the clock cycle that it is needed. You live (and die) on the pixel clock.*



Save big computation tasks up for the blanking regions when living on the beam



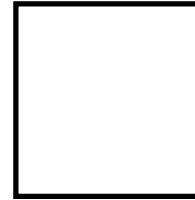
Lab 04 Setup (Racing the Beam)



Examples of pixel logic:

Whole Screen is White:

```
assign red = 8'hFF;  
assign green = 8'hFF;  
assign blue = 8'hFF;
```



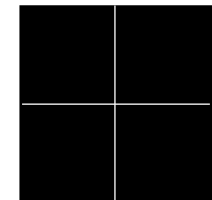
Draw green vertical line at horizontal spot 500:

```
always_comb begin  
    green = (hcount==500)? 8'hFF:8'h00;  
    red = 8'h00;  
    blue = 8'h00;  
end
```



Draw white crosshair at (500,500)

```
always_comb begin  
    if (hcount==500 || vcount==500)begin  
        green = 8'hFF;  
        red = 8'hFF;  
        blue = 8'hFF;  
    end else begin  
        green = 8'h00;  
        red = 8'h00;  
        blue = 8'h00;  
    end  
end
```

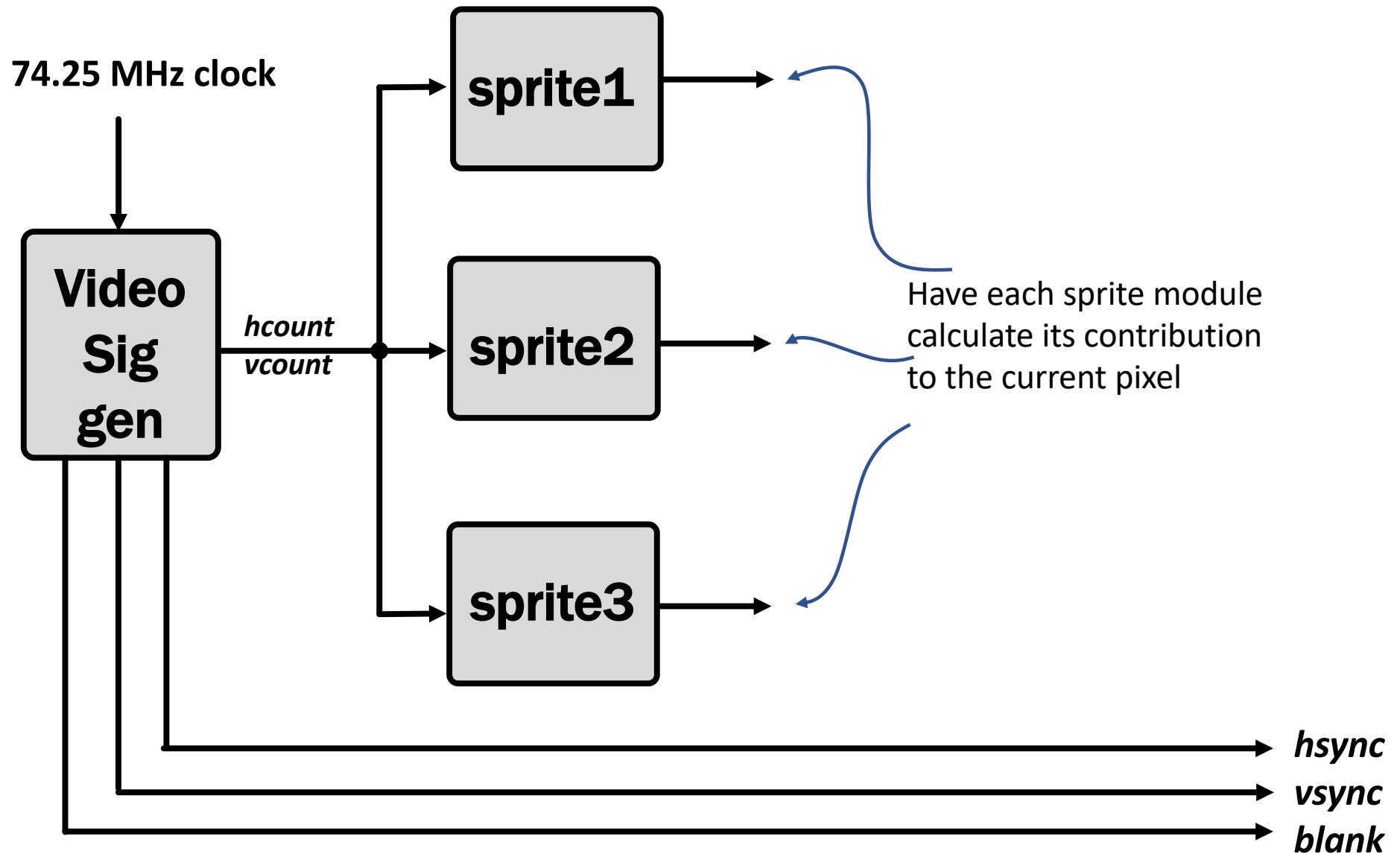


Examples of Pixel Logic

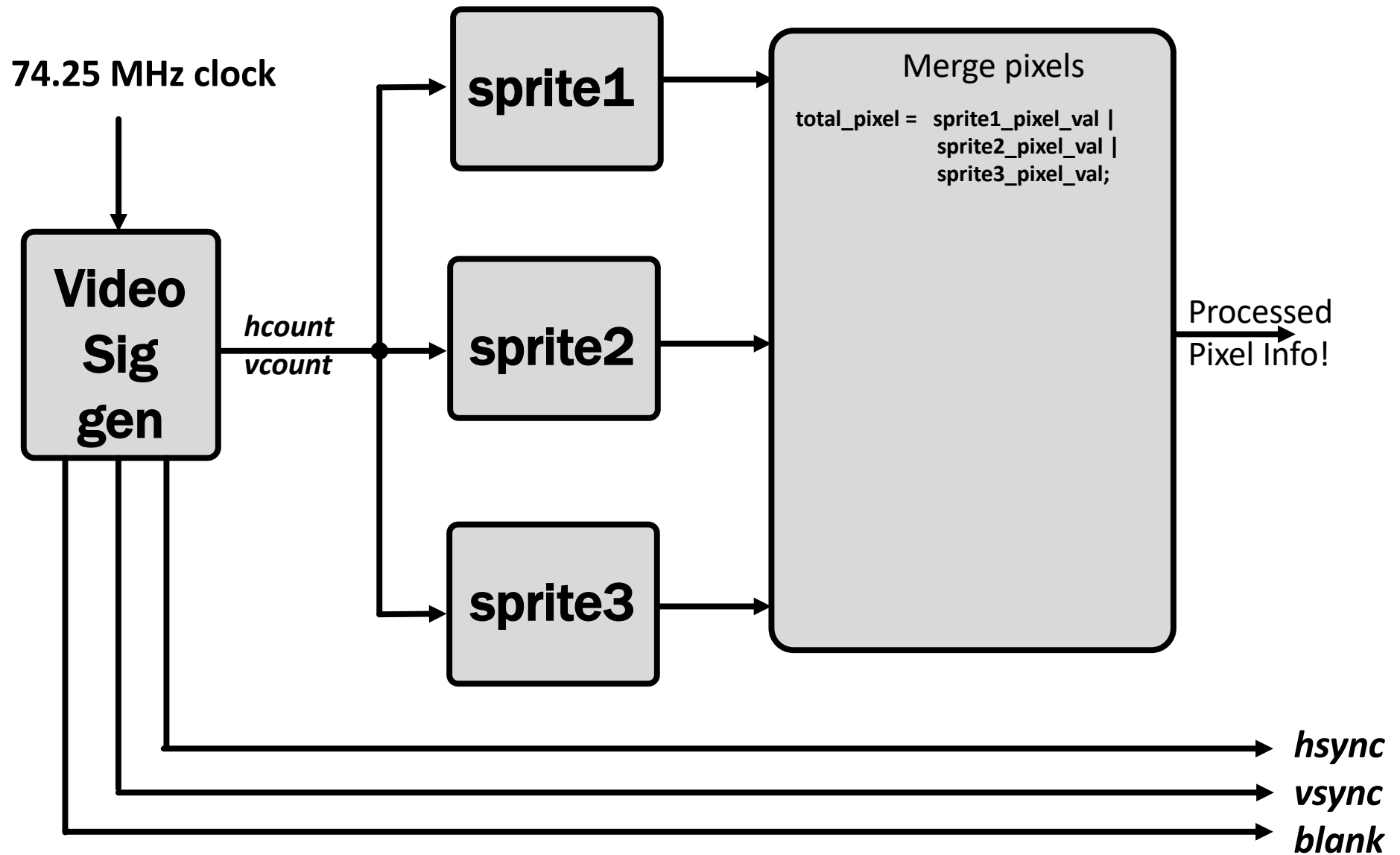
- From Lab 04:
- Draw a white pixel if within a set of rectangular bounds!
- Based solely on where one is currently in the drawing space.

```
1  module block_sprite #(
2      |         parameter    WIDTH=128,
3      |         |           HEIGHT=128,
4      |         |           COLOR=24'hFF_FF_FF
5      |     )
6      |     input wire [10:0] h_count,
7      |     input wire [9:0]  v_count,
8      |     input wire [10:0] x,
9      |     input wire [9:0]  y,
10     |     output logic [7:0] pixel_red,
11     |     output logic [7:0] pixel_green,
12     |     output logic [7:0] pixel_blue);
13
14     logic in_sprite;
15     assign in_sprite = ((h_count >= x && h_count < (x + WIDTH)) &&
16     |                   (v_count >= y && v_count < (y + HEIGHT)));
17     always_comb begin
18         if (in_sprite)begin
19             pixel_red = COLOR[23:16];
20             pixel_green = COLOR[15:8];
21             pixel_blue = COLOR[7:0];
22         end else begin
23             pixel_red = 0;
24             pixel_green = 0;
25             pixel_blue = 0;
26         end
27     end
28 endmodule
29
```

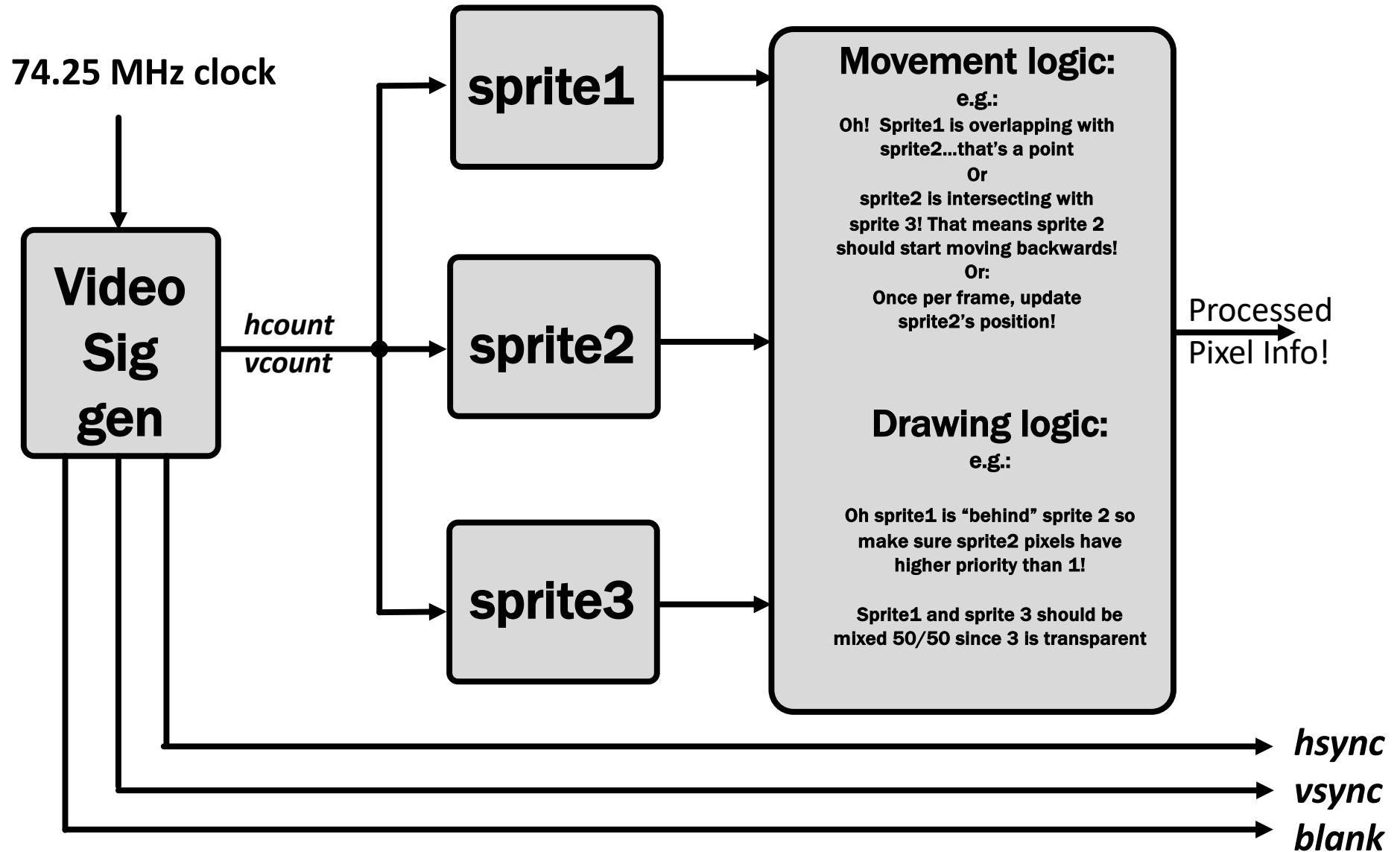
How can a drawing start to work?



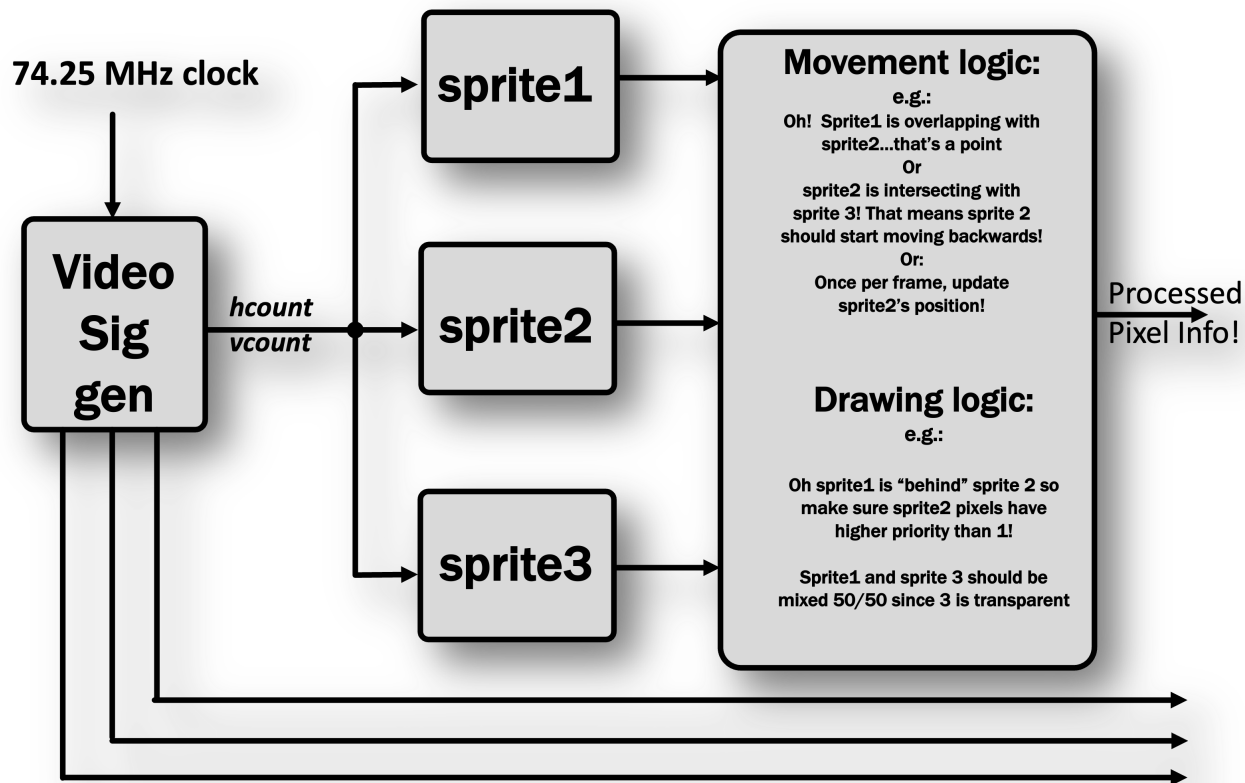
How can a drawing start to work?



How can a drawing start to work?



Sprites and Movement Might Take too Long!!!! (too much tpd!!) ☹ ☹



May need to pipeline

Though it won't really burn us until Week 5

Remember with Video

- At 720p, (74.25 MHz clock) each pixel is a visual representation of the 13.47 ns clock period!
- Having too much logic will start to cause individual pixels to blur and things.
- Kinda cool ngl

Final Projects Coming Up

- In another ~week or so, we have to start thinking about planning on starting to get going on final projects.
- First part of that is teaming and teams benefit from targeting shared goals
- On the site, we'll put up an archive of final projects