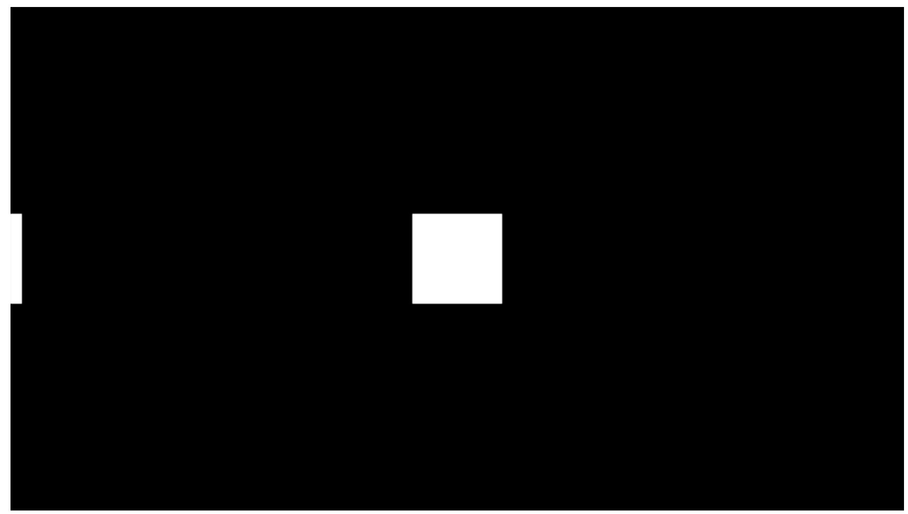
Video

6.205

Admin

- Week 03 was due last night
- Week 04 out after class: Video, babiiiiieeeeeee



Order, if’s else’s

- All three of these are the same:
- If you need to use one of these, I’
  recommend the latter two
  - In particularly long code, they force you to think about their priority, exclusivity correctly I’ve found

```
always_ff @(posedge clk)begin
    if (a)  begin
        q <= z;
    end

    if (b) begin
        q <= y;
    end
end

always_ff @(posedge clk)begin
    if (b) begin
        q <= y;
    end else if (a) begin
        q <= z;
    end
end

always_ff @(posedge clk)begin
    q <= b?y:a?z:q;
end
```

When Writing Stateful Logic…

- Try to group tasks/events that happen on the same state together…
- If you have lots of parallel tasks all on these separate if/else if/else chains that are themselves disconnected, lots of weird bugs can come out because you have to scroll back and forth a bunch follow the logic…
- Then you think a thing is happening on a certain cycle when maybe it isn't because it is getting overrode by a condition specified in some other loop somewhere.

Also Case Statements are Good

- If/elses and even parallel if’s as shown on the previous page get encoded as priority logic

```
always_ff @(posedge clk)begin
    if (state == IDLE) begin
        q <= y;
    end else if (state == FIRST) begin
        q <= z;
    end else if (state == SECOND) begin
        q <= zz;
    end else begin
        q <= zzz;
    end
end
```

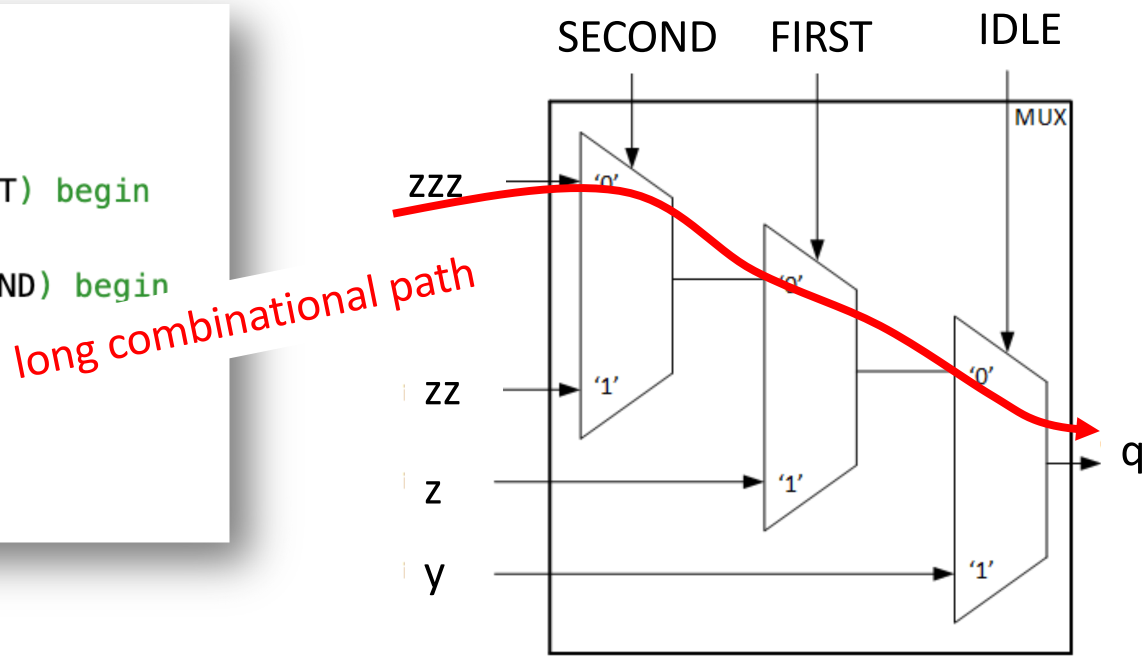


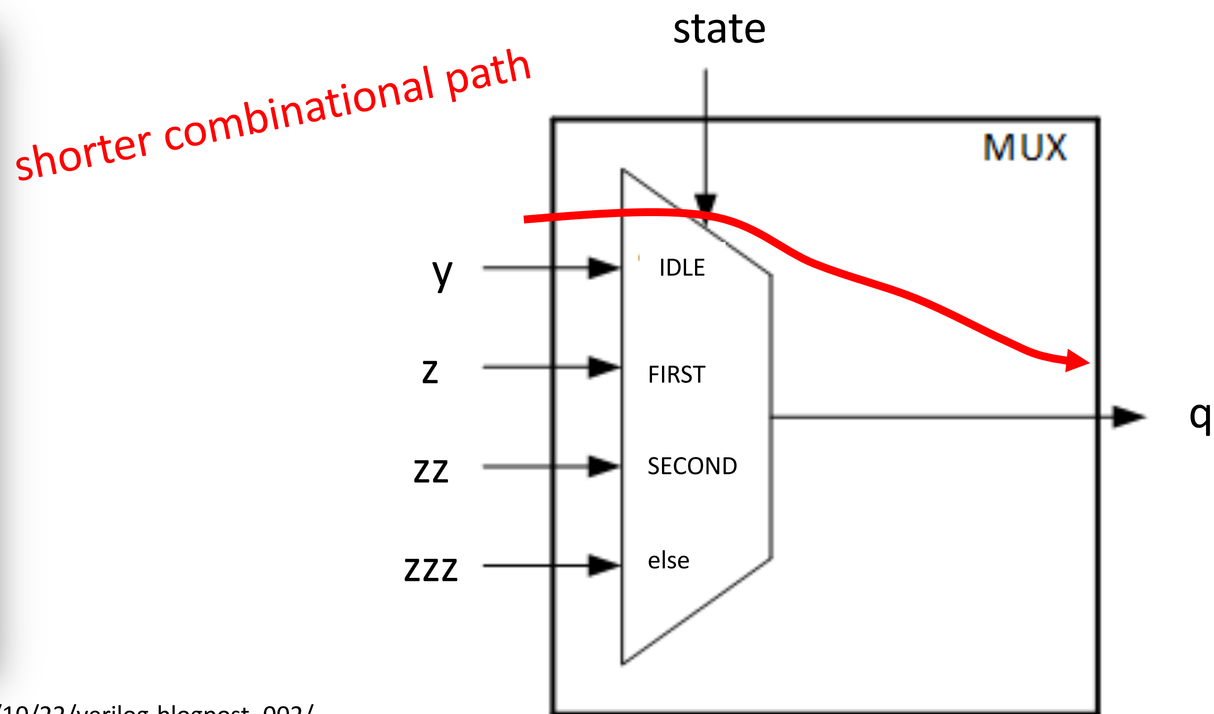
Fig 1

https://www.kevnugent.com/2020/10/22/verilog-blogpost_002/

Also Case Statements are Good

- If logic can be structured without priority, then do it! Can yield simpler underlying logic.

```
always_ff @(posedge clk)begin
    case(state)
        IDLE: begin
            q <= y;
        end
        FIRST: begin
            q <= z;
        end
        SECOND: begin
            q <= zz;
        end
        default: begin
            q <= zzz;
        end
    endcase
end
```



https://www.kevnugent.com/2020/10/22/verilog-blogpost_002/

Priority-Encoding

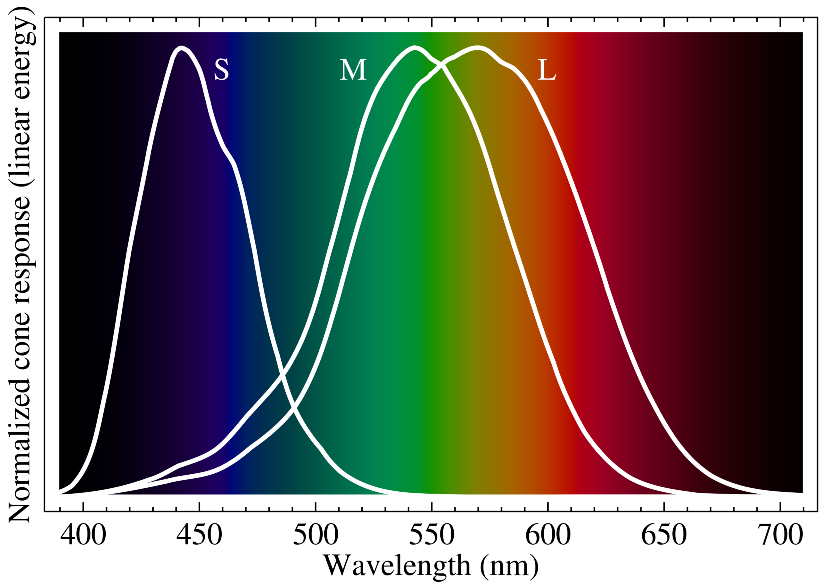
- Priority-encoding is another one of those luxuries from software land, like indexable arrays, representing things with numbers, classes, etc…
- It feels good because it is familiar and it "worked in python"
- But unless you absolutely need it (and you often will!), it can come at a cost.

Video

Displays are for 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

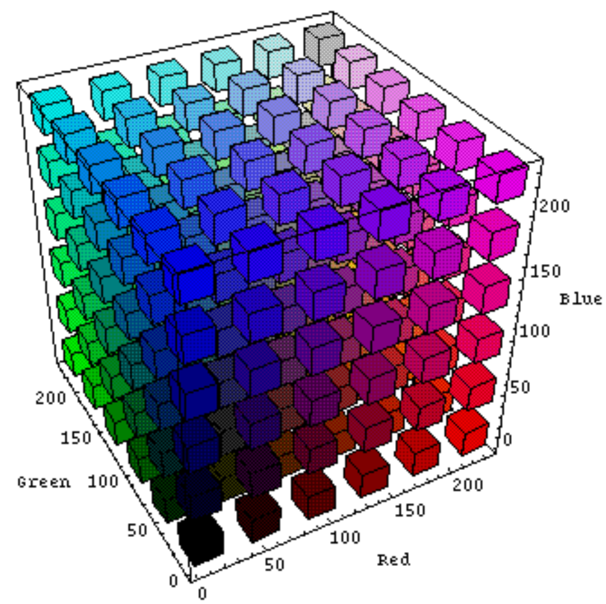
- A human eye therefore has three independent inputs regarding visual EM radiation
- Called **”trichromatic”**



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

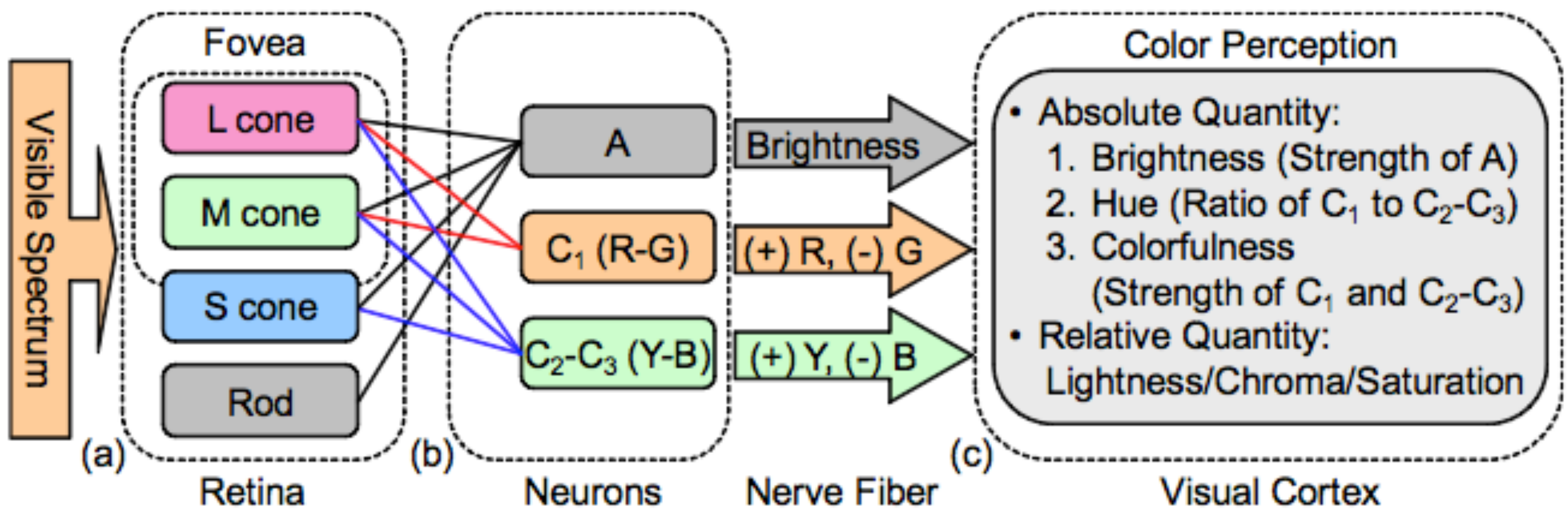
One form of RGB space (not the only way to display it)



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

Opponent Process Color Theory

- It actually isn't as simple as trichromatic vision



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

Worst Case Scenario

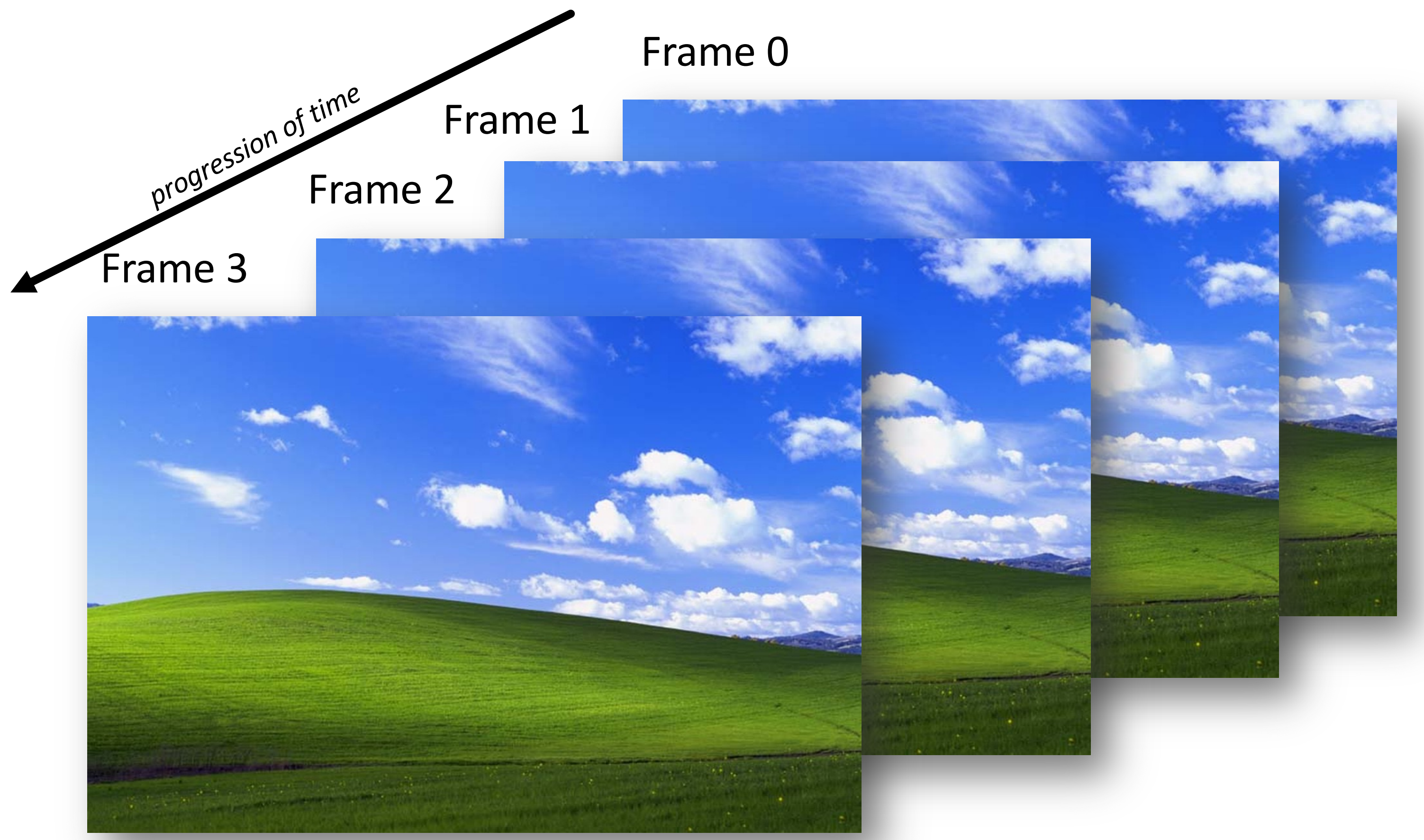
- If a person has all color receptors working…
- because of noise limitations in our naturally-evolved encoding scheme that communicates from the cone cells up to the brain…
- we can perceive about 7-10 million unique colors depending on your research source…
- How many bits do we need to encode all 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 RC time constants of our eye's to "fake" motion.



How to Transmit 5/6-dimensional data?

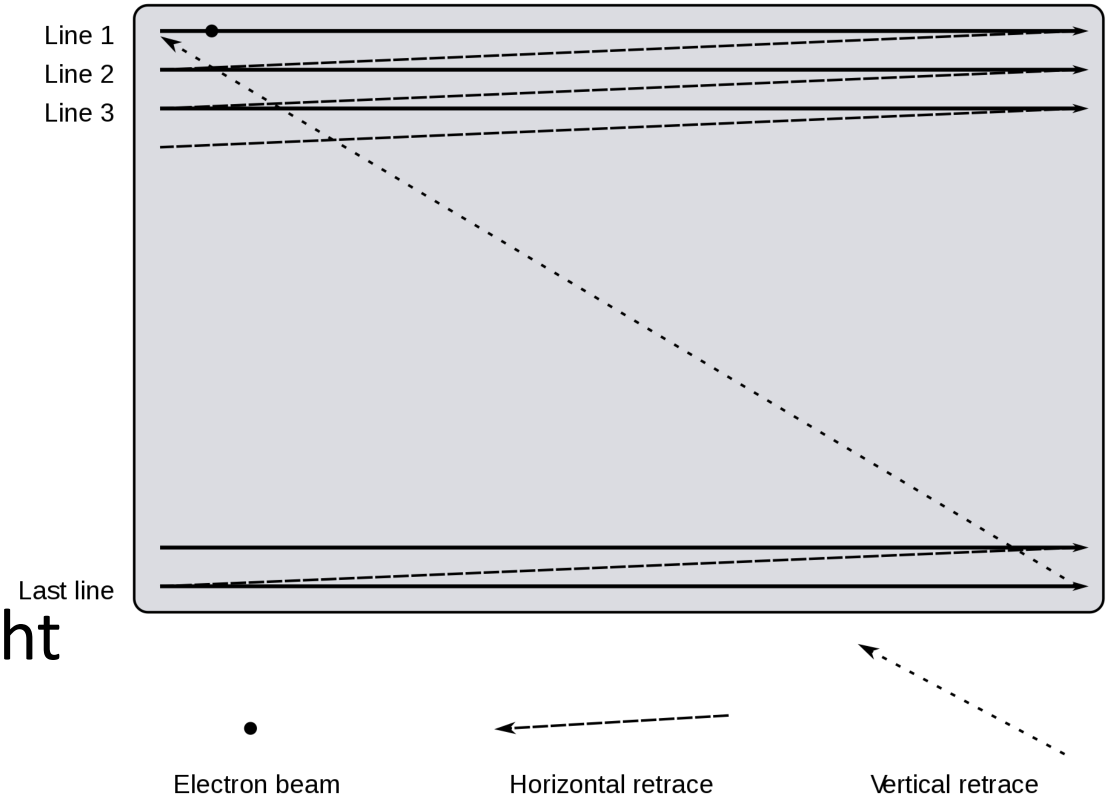
- Ideally need to convey enough 5D values quickly enough to render images fast enough that they show up as one…
- AND we also need to do the above fast enough so that fresh images appear quickly enough

How to Draw: The Raster Scan

- 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:

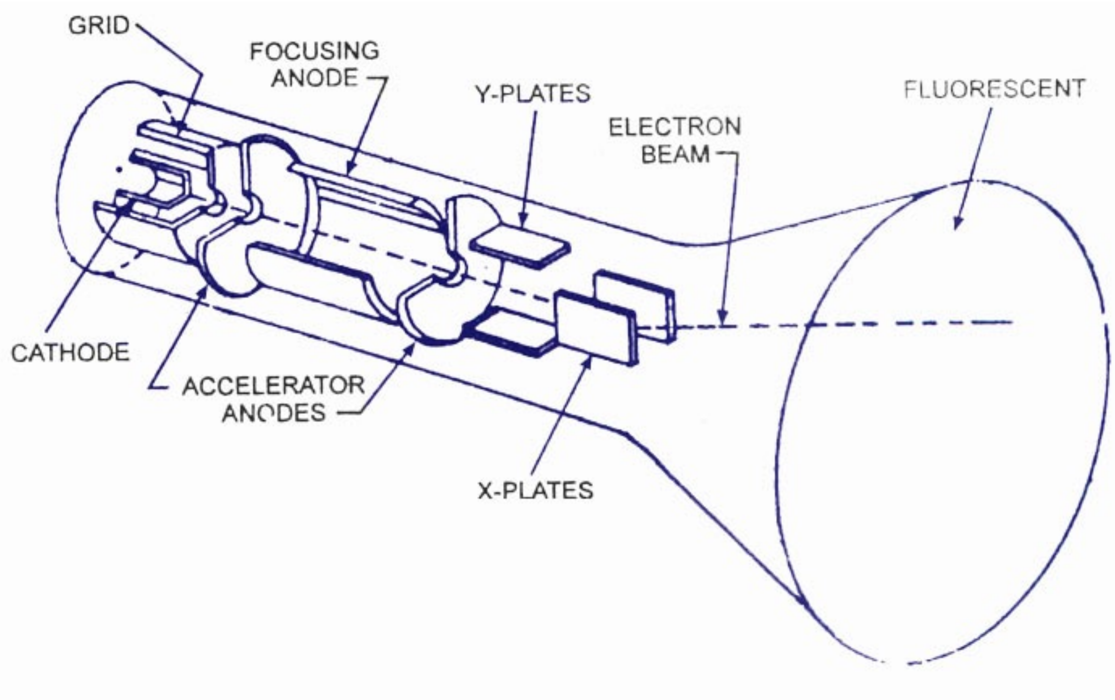
Rastrum, used for drawing musical staff

- 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)

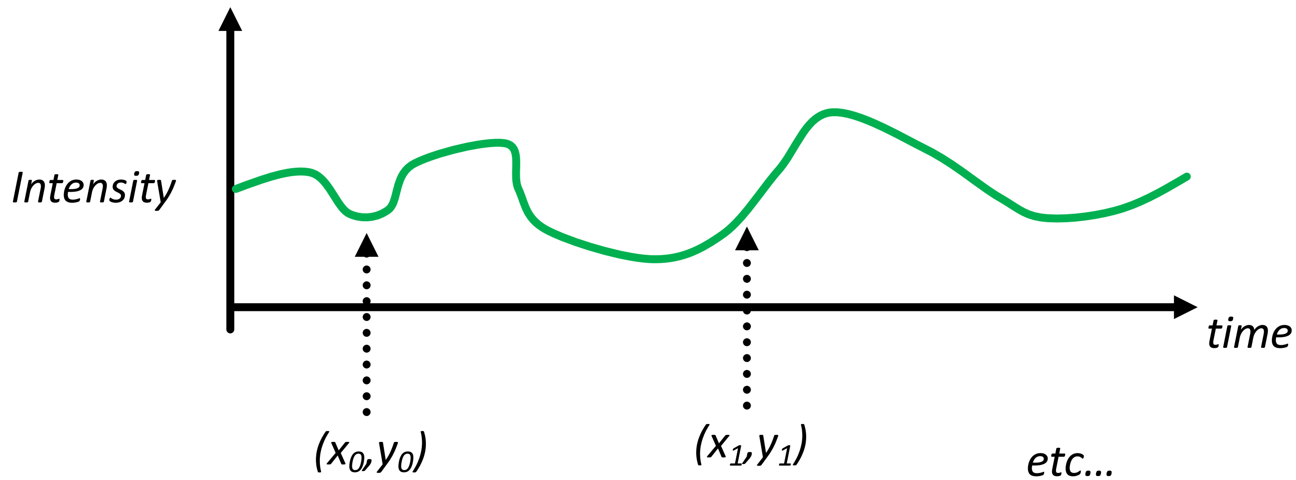
- Electron beam of varying intensity would be quickly rastered on a fluorescent screen making image



Cathode Ray Tube

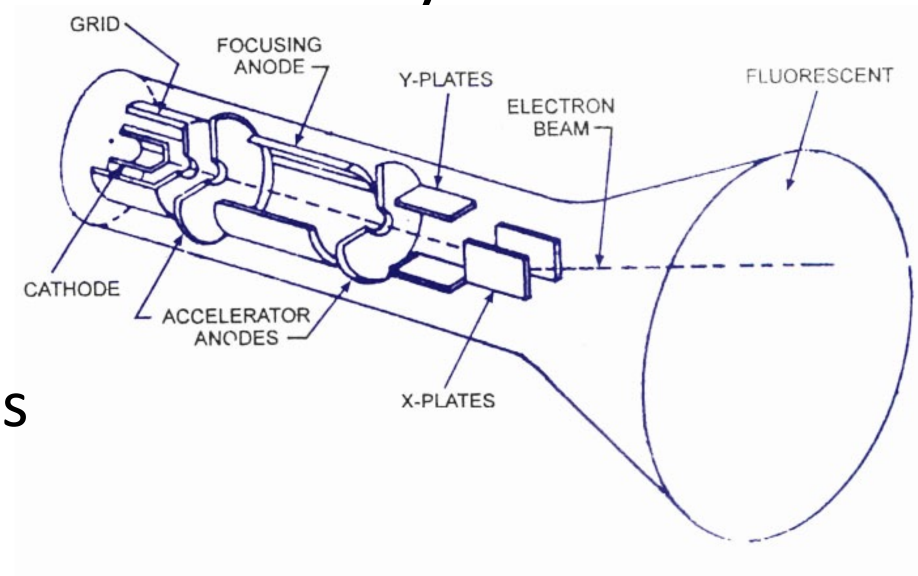
Raster Pattern of Drawing

- Allows time ←→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

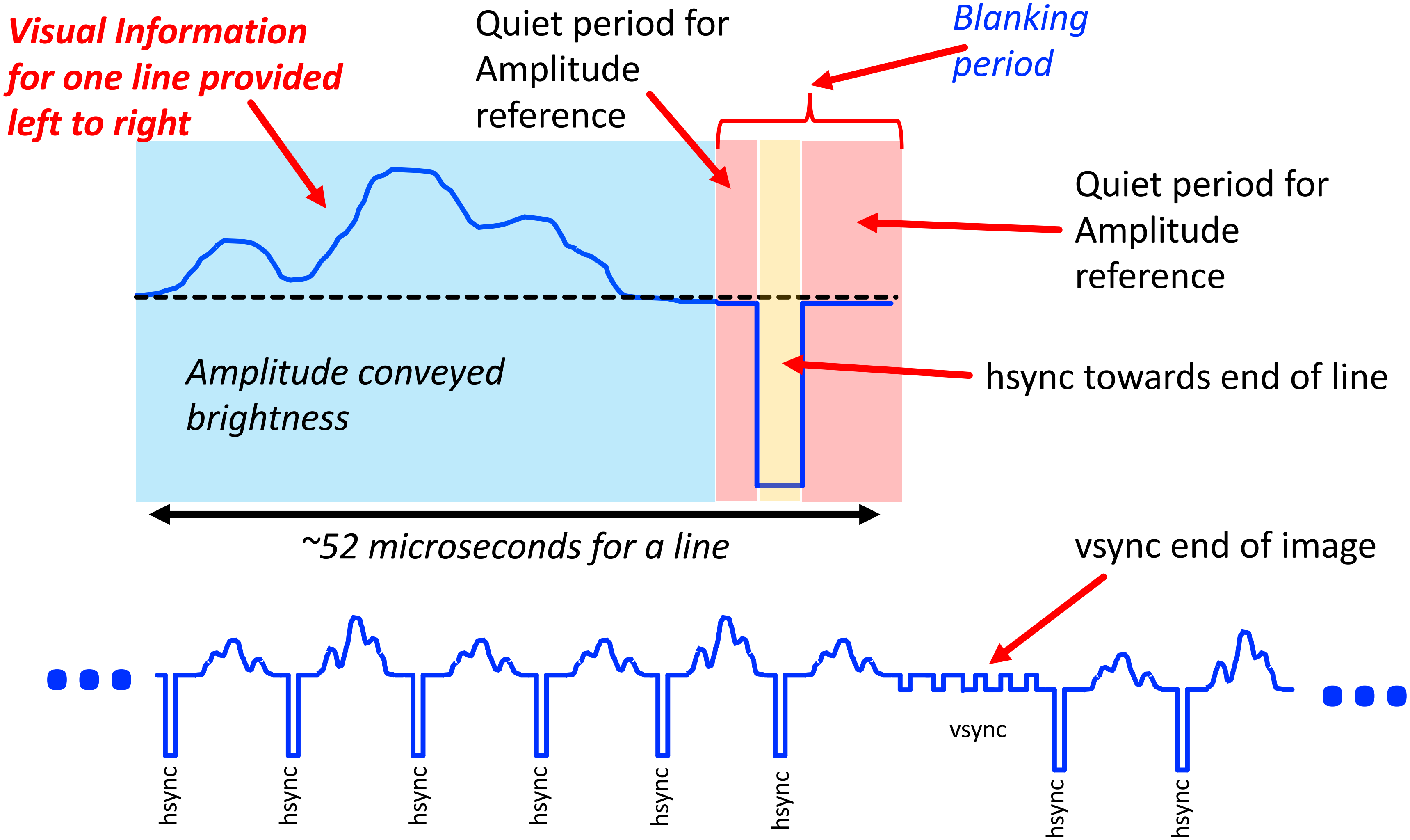


Cathode Ray Tube

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

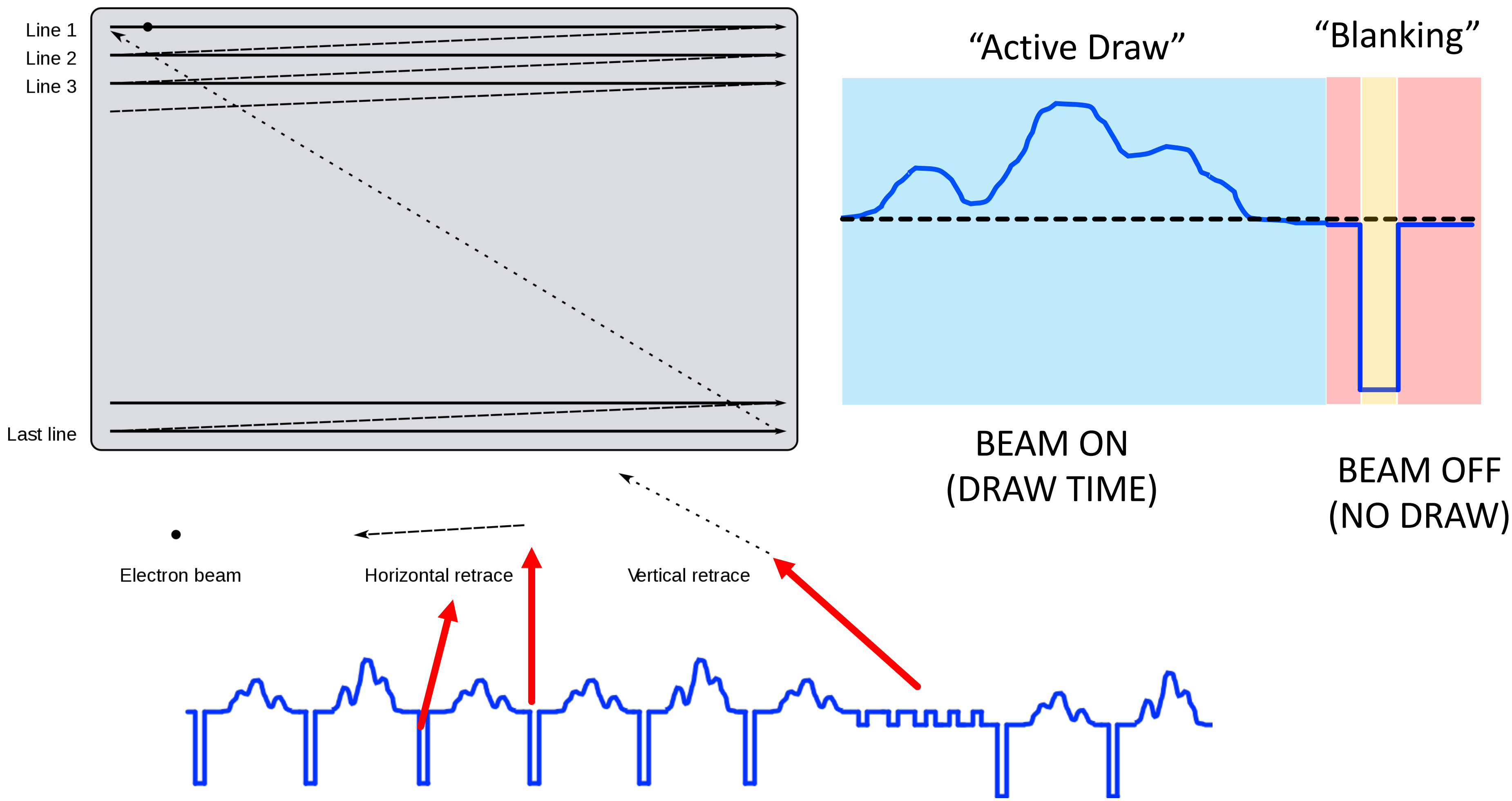
Black and White Video signal

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



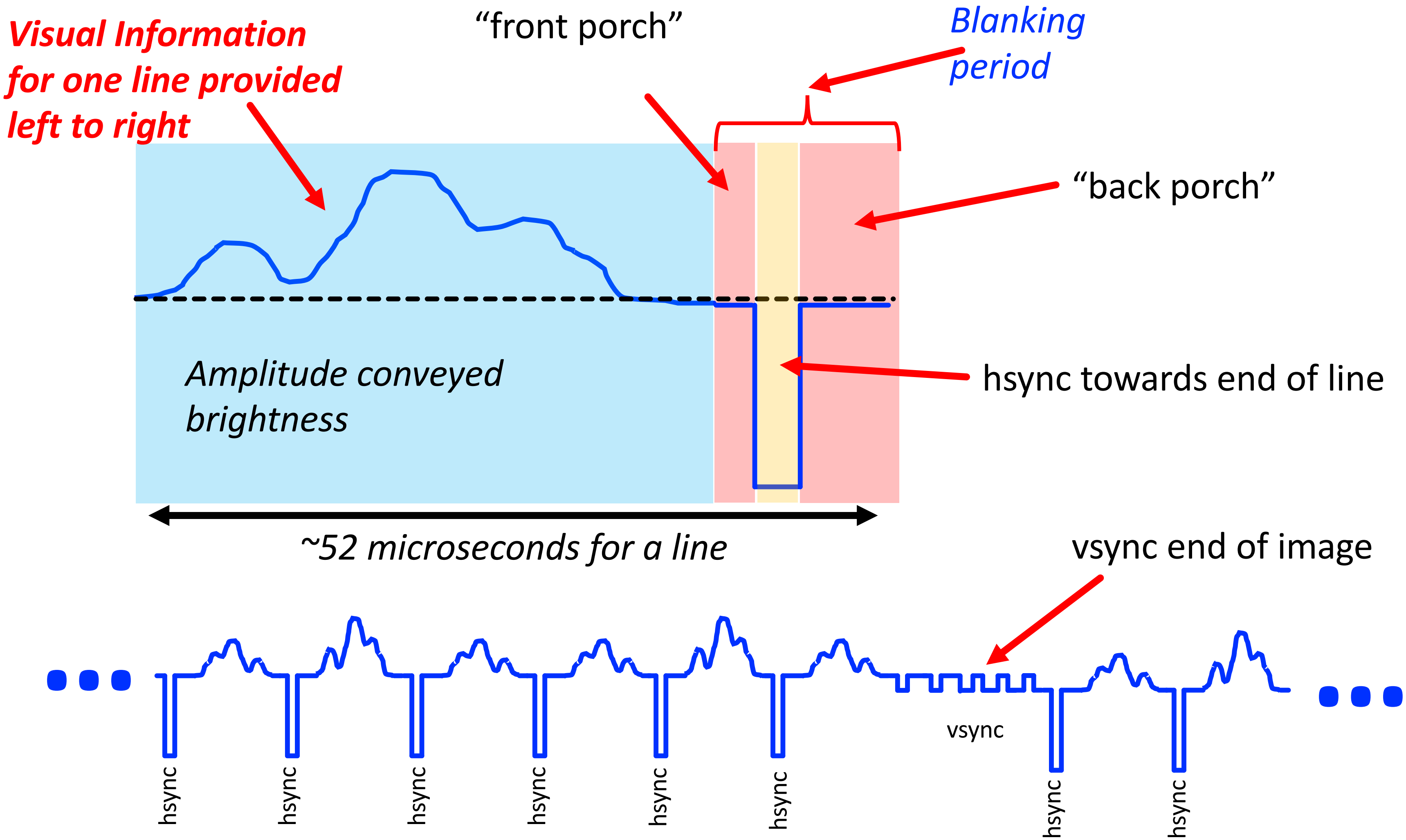
Controls in Action

- Signal completely controls beam location and intensity!



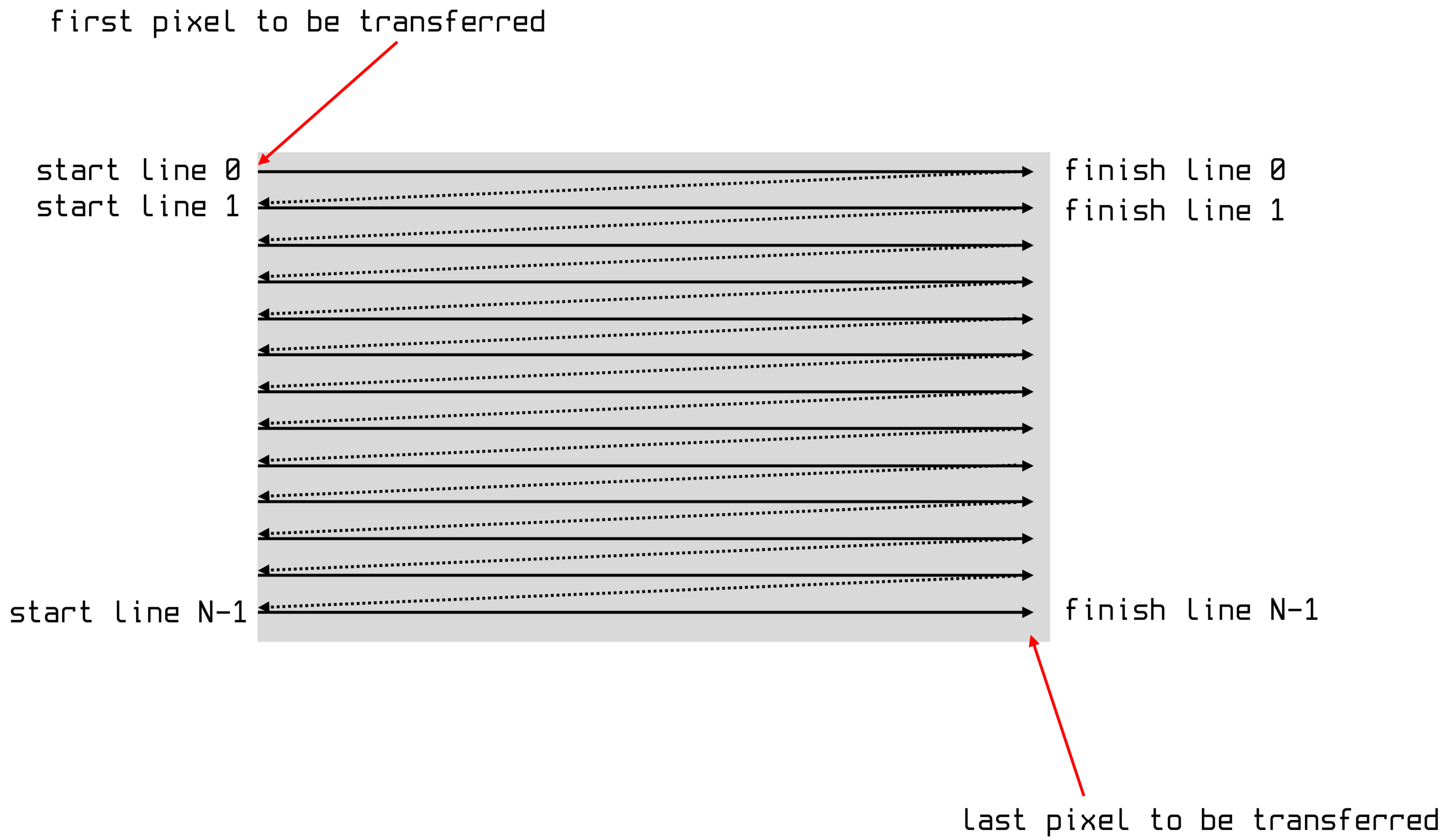
Black and White Video signal

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



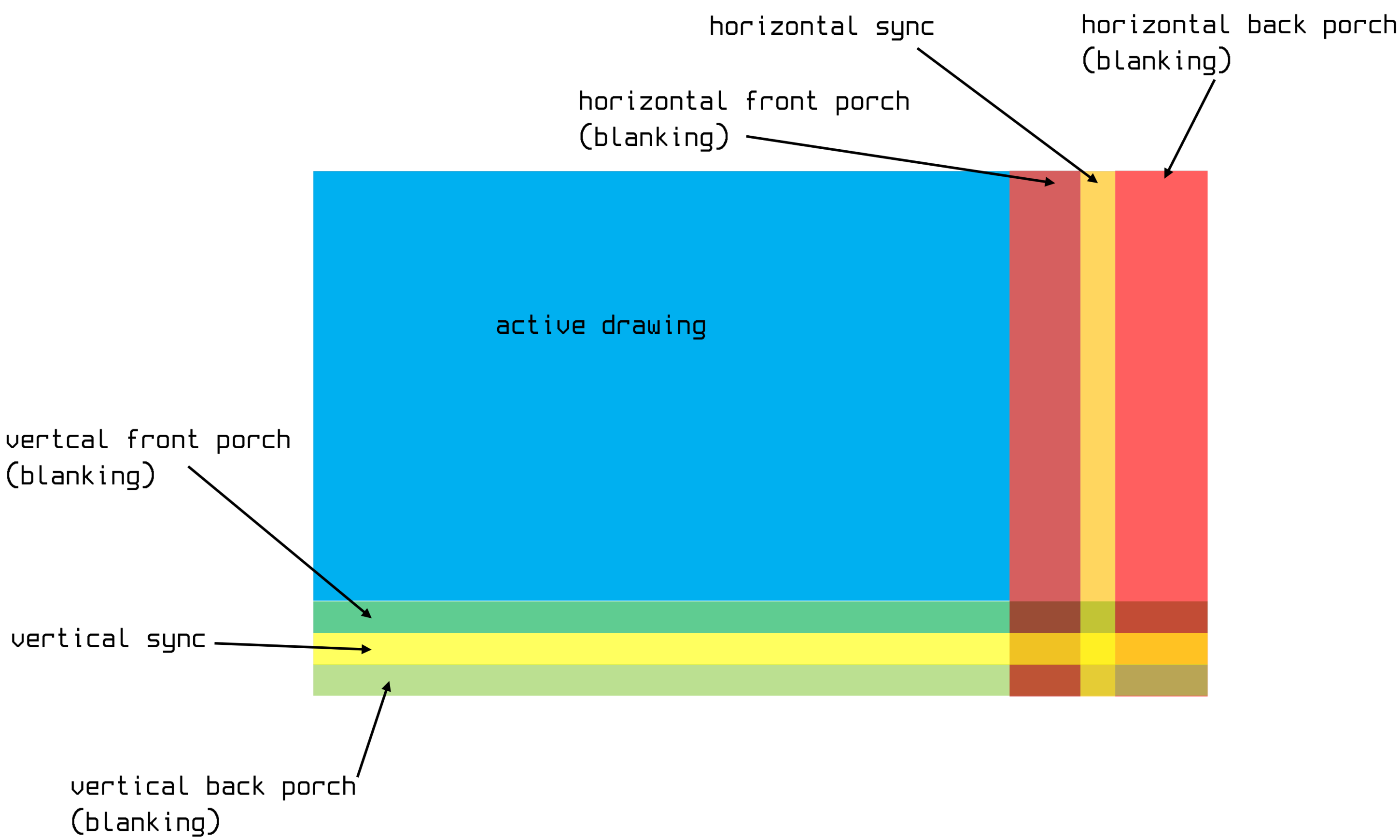
Frame

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



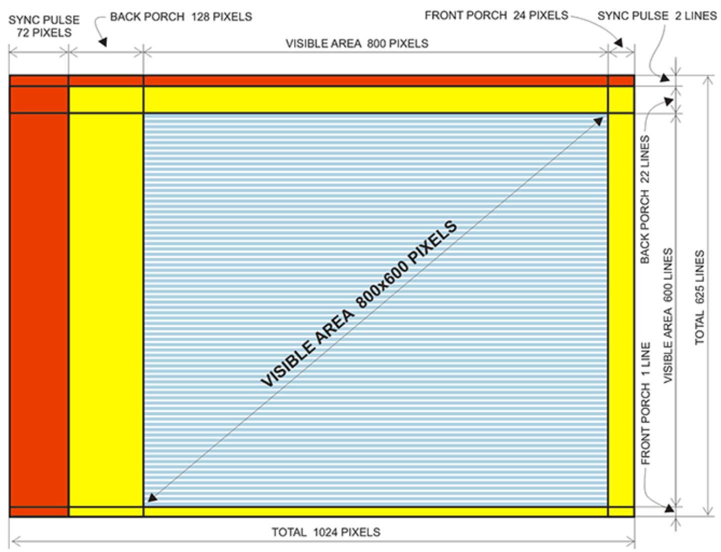
Frame

- 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.

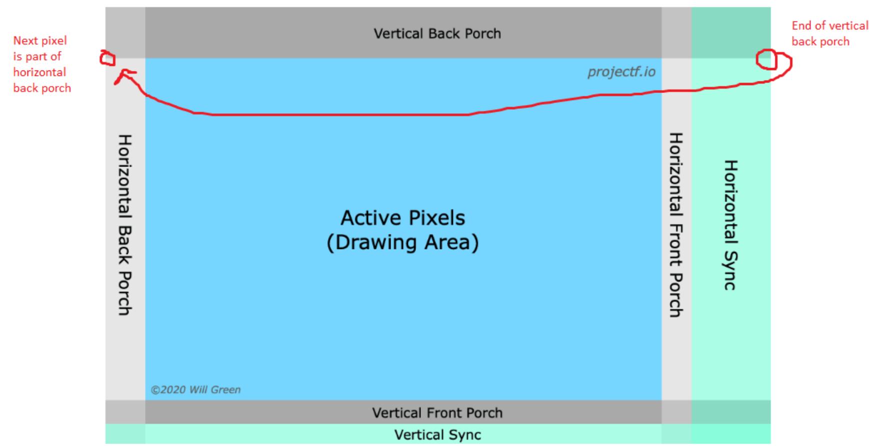


“Location” of Regions of Frame

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



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

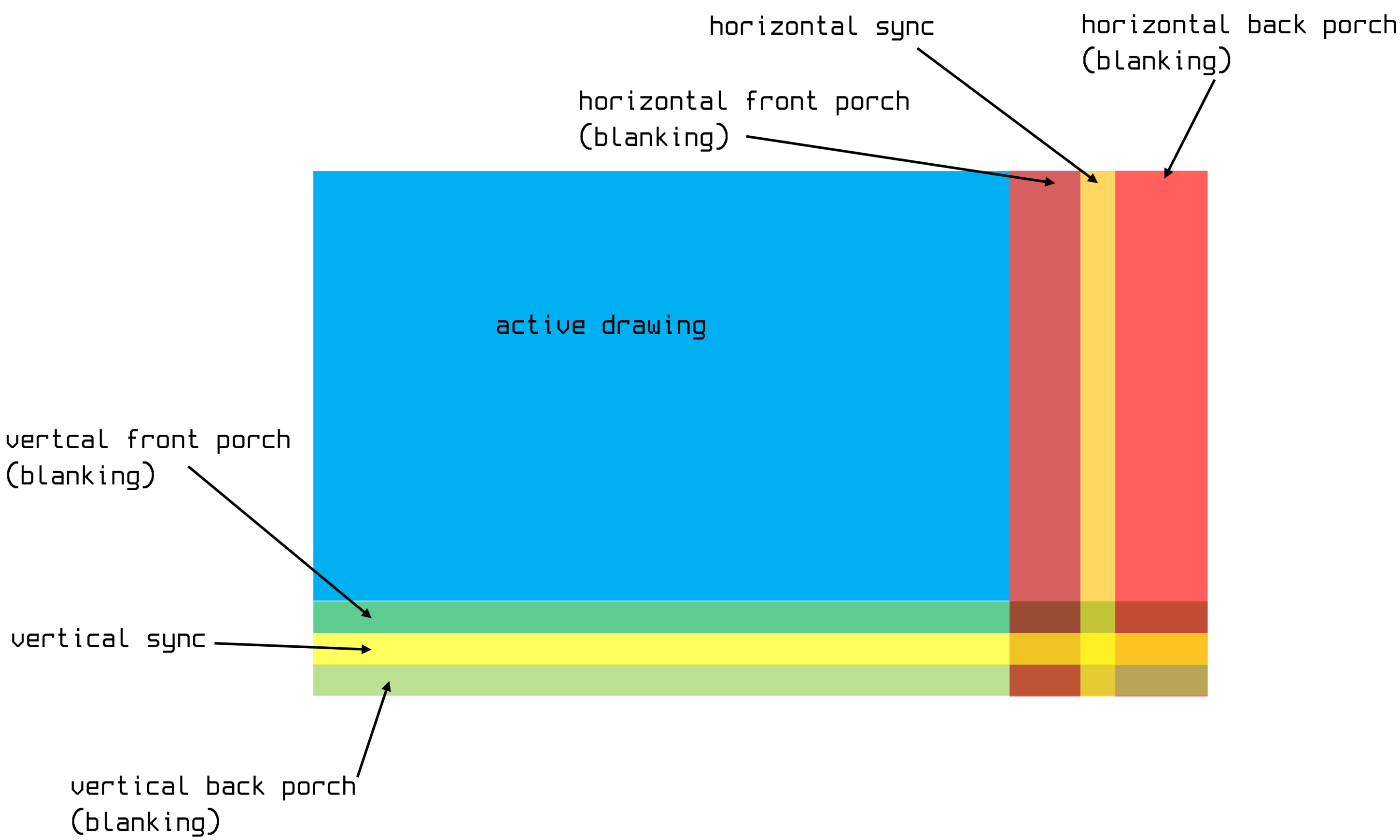


https://electronics.stackexchange.com/questions/614207/correct-order-of-monitor-display-timing

- 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.



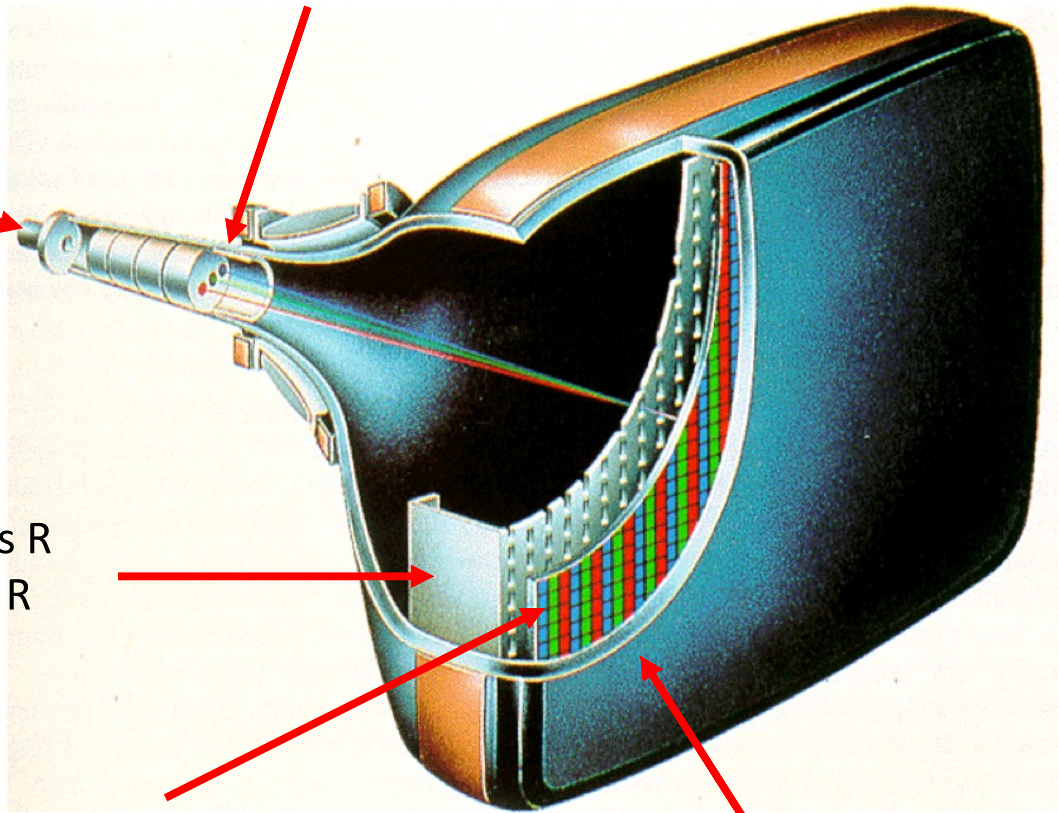
Original Video

- 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



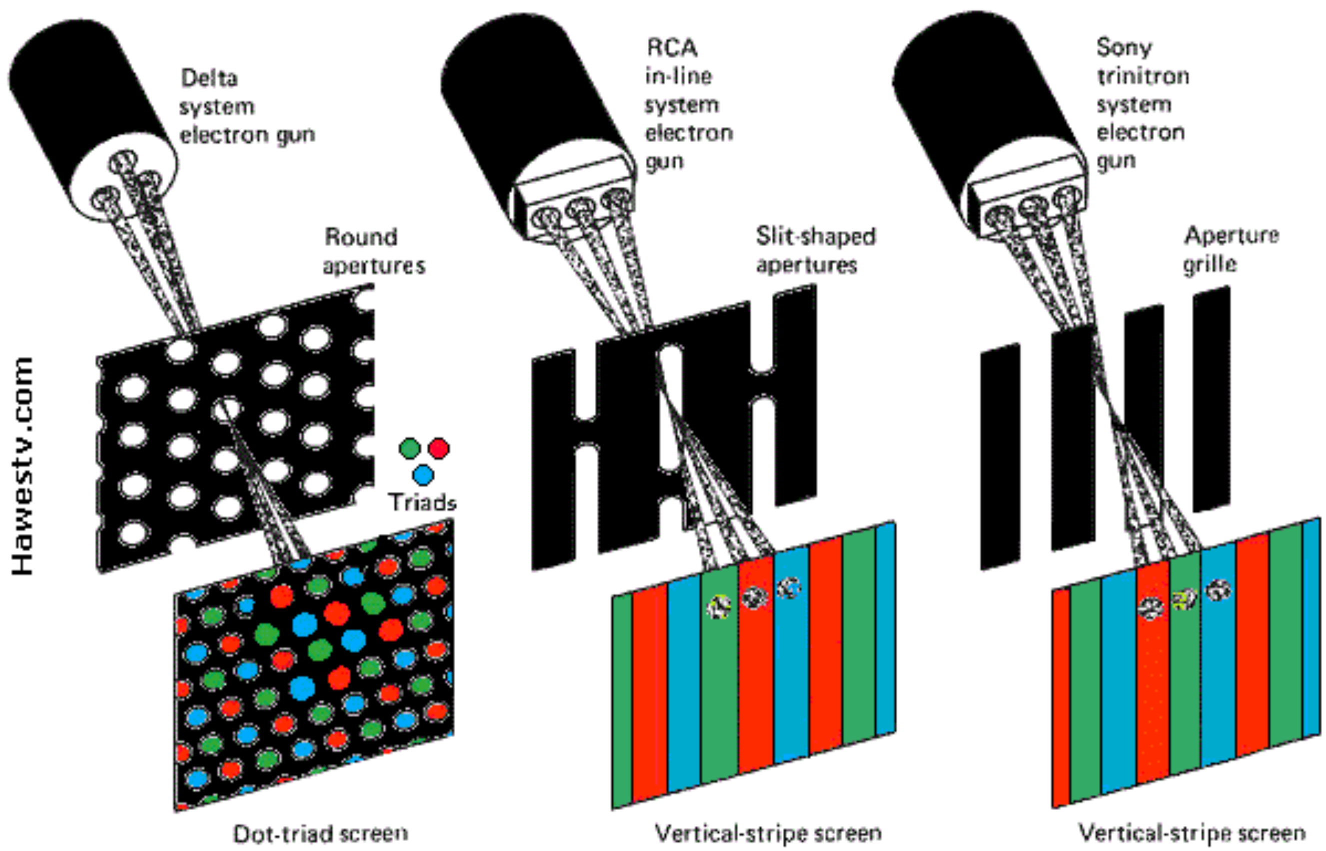
Shadow mask: ensures R beam only illuminates R pixels, etc.

Three different phosphor screens

Anode

Source: PixTech

Shadow Mask



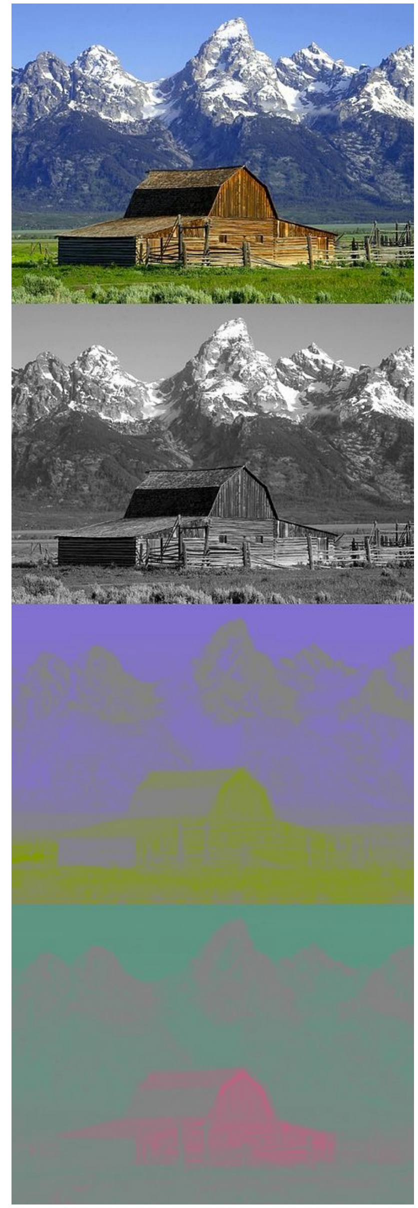
http://www.hawestv.com/etv-crts/crt-flechsig/flechsig_1st_color_crt.htm

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 ←→ 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 ←→ 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

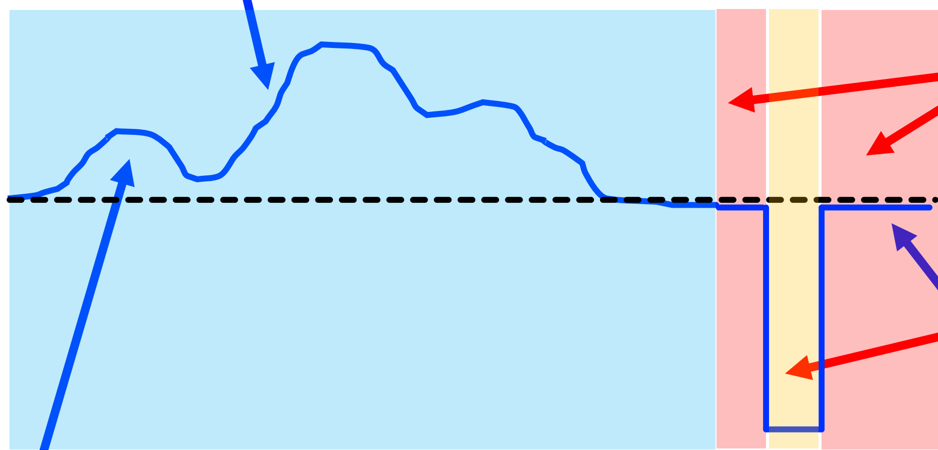
- Keep signal the same as before but add other stuff to it!

Y value (Luminance) encoded in base signal

Blanking period

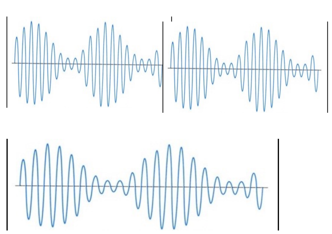
Quiet period for Y reference

Amplitude of low frequency signal conveyed brightness

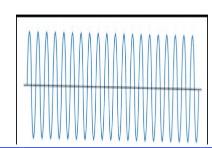


hsync end of line

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



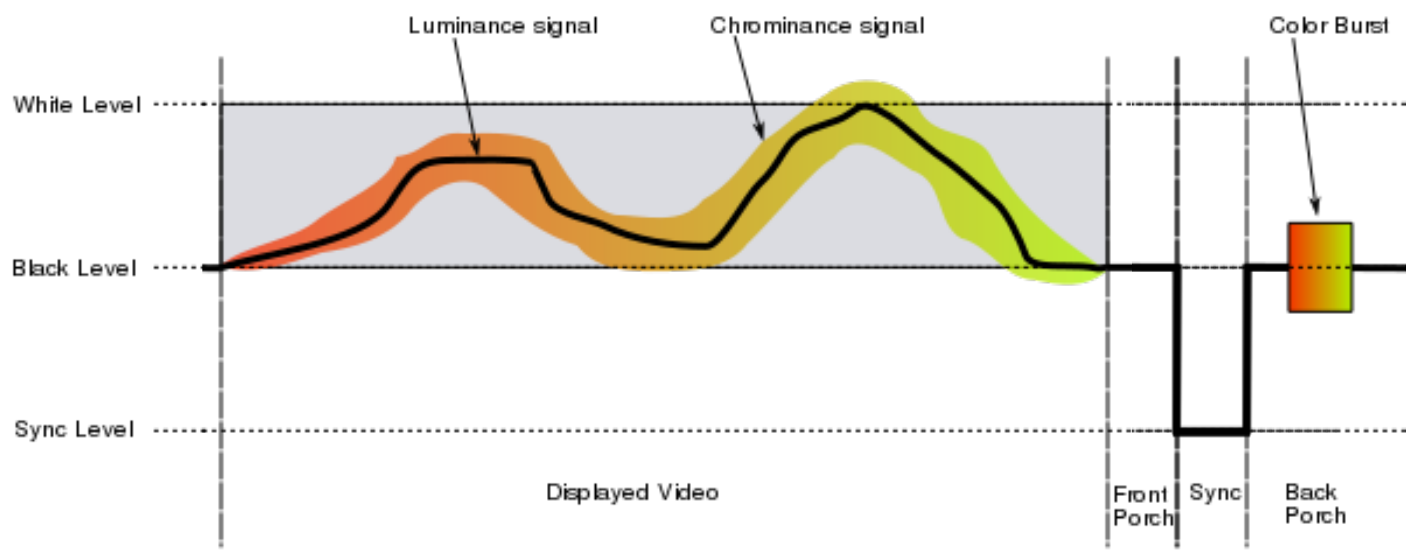
Add a Cr/Cb "color burst" to region of blanking period to calibrate for Cr/Cb Amplitude Demodulation



Composite Video Encoding:

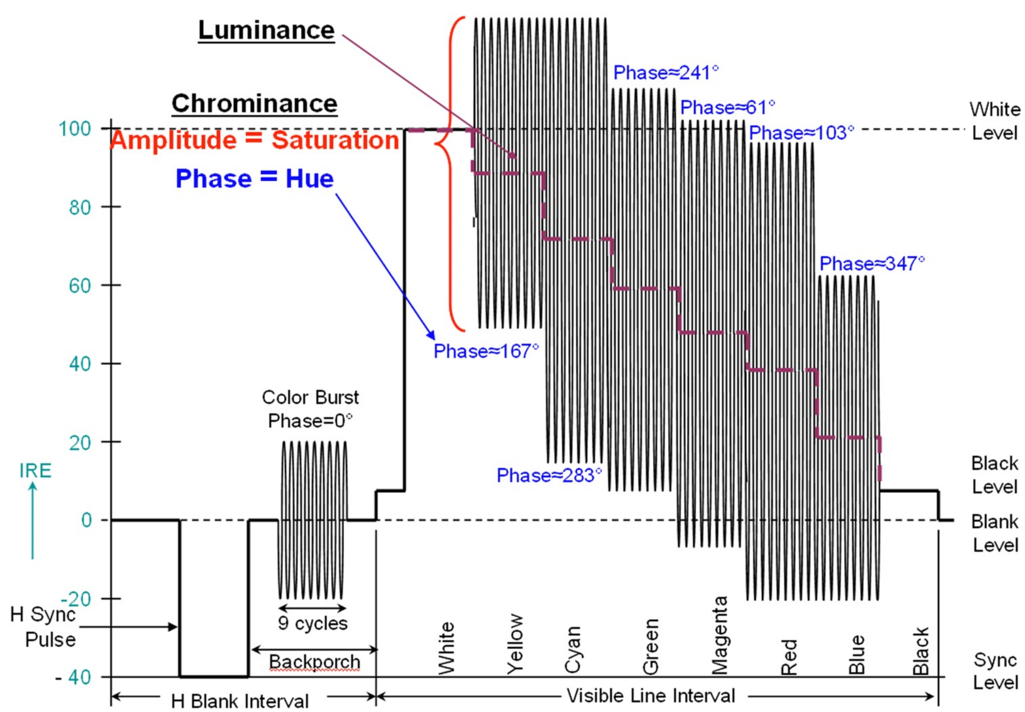
Used for most color TV transmissions and component video up until early 2000’s

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:



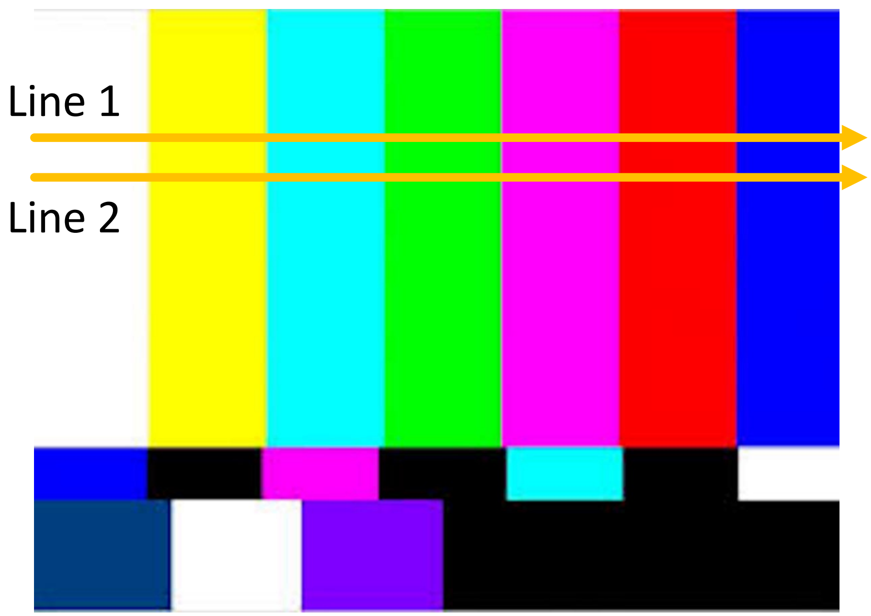
- Amplitude is **Saturation**
- Phase is **Hue**
- **Luminance** is low-freq original value

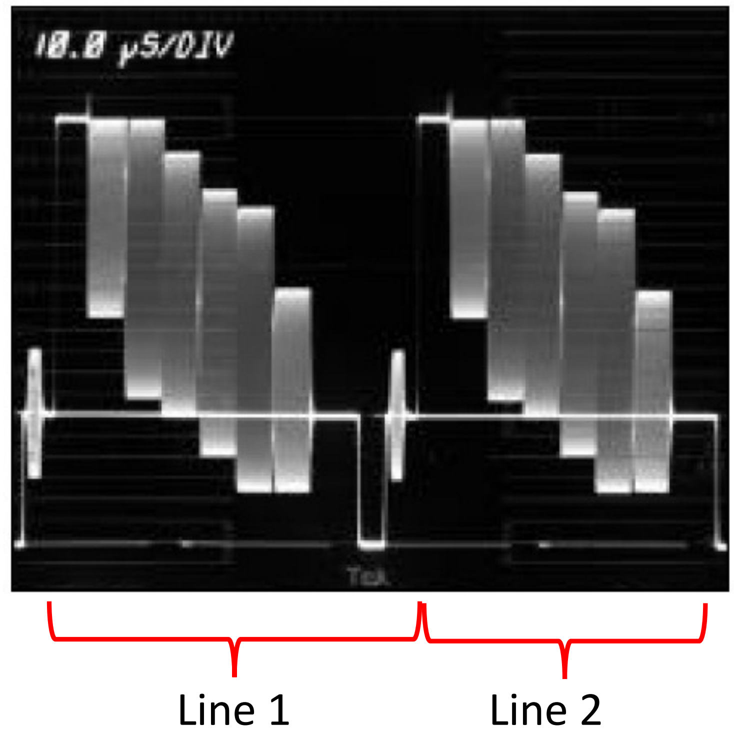
- Hue, Saturation, Luminance (HSL) is a cylindrical color space that is used a lot!

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)

Component Video Sockets on Virgin Air airplane in 2019

this

Is slightly smaller than

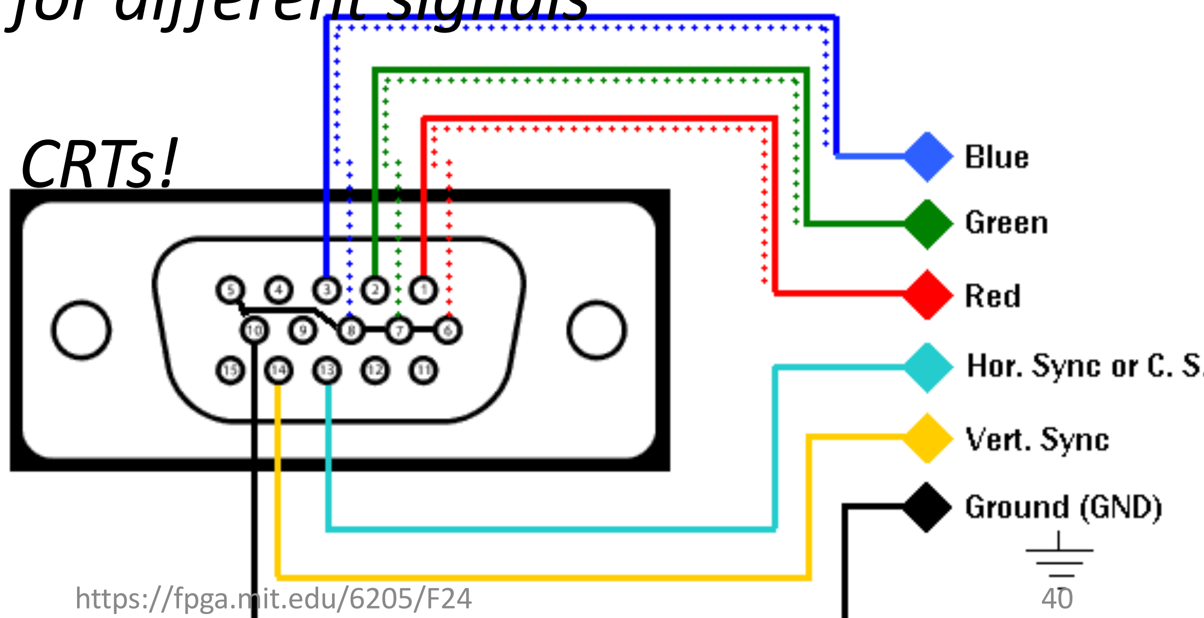
this

Poor engineering.

VGA (Video Graphics Array)

DB15 Connector

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



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)

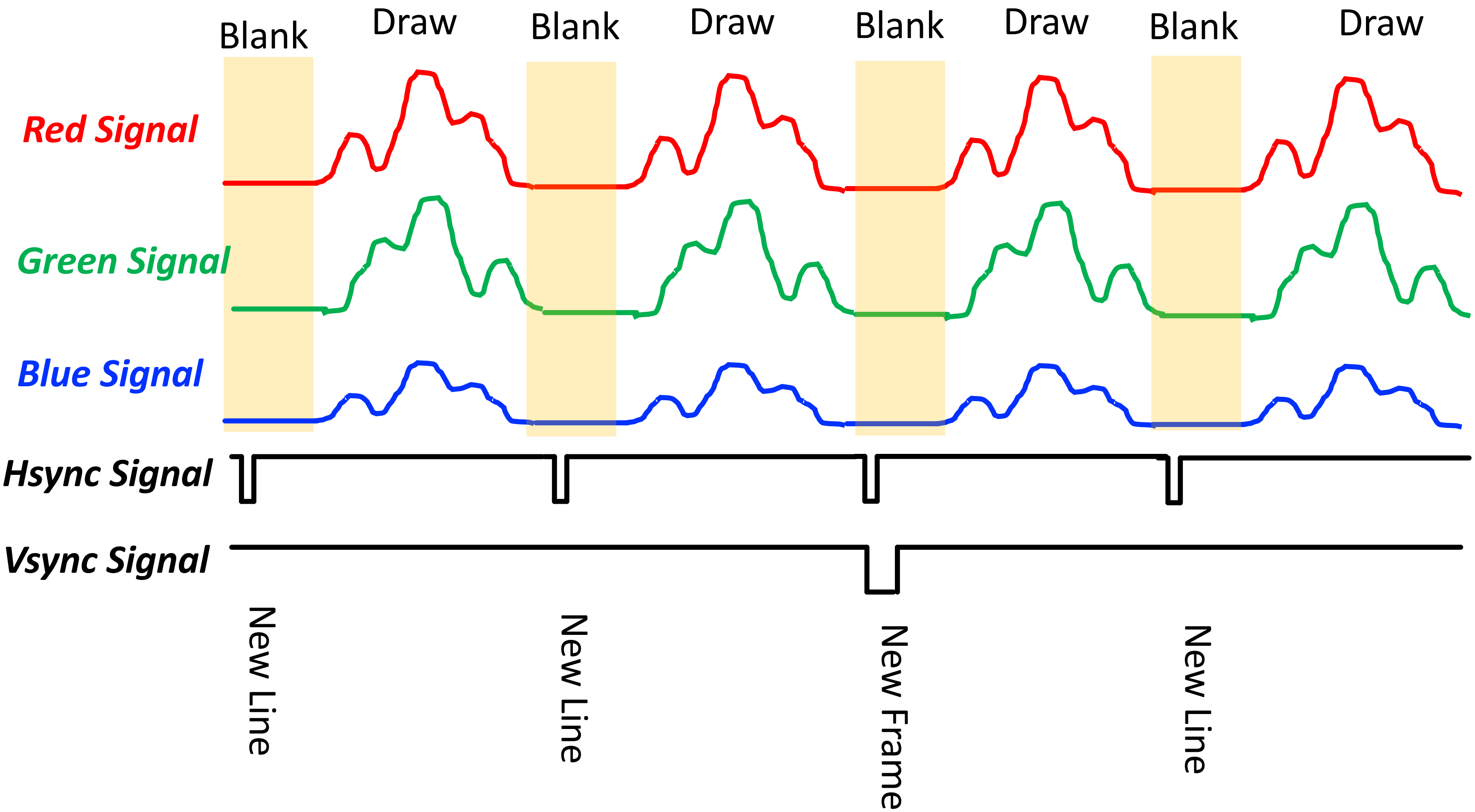


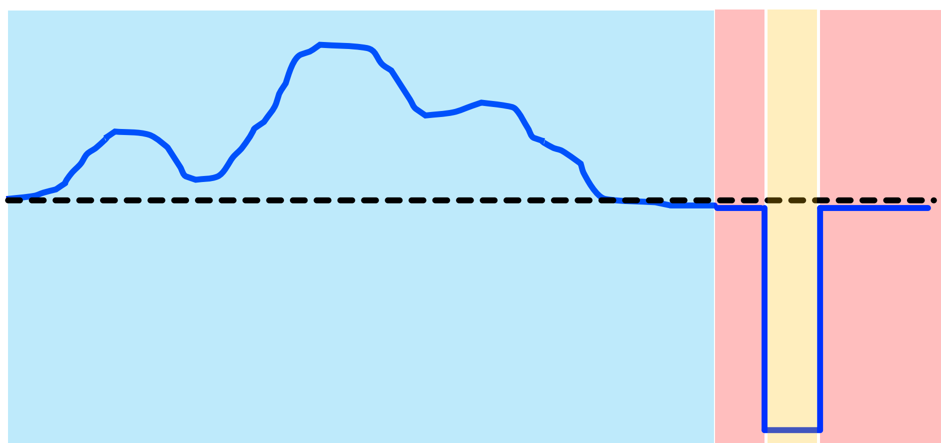
Figure out Display Resolution

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	1280x720	16:9	6.205 F24
HD TV	1920x1080	16:9	
iPhone 6 Plus	1920x1080	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" (3rd Generation)
4K HD TV	3840x2160	16:9	
8K HD TV	7680x4320	16:9	Really expensive TVs

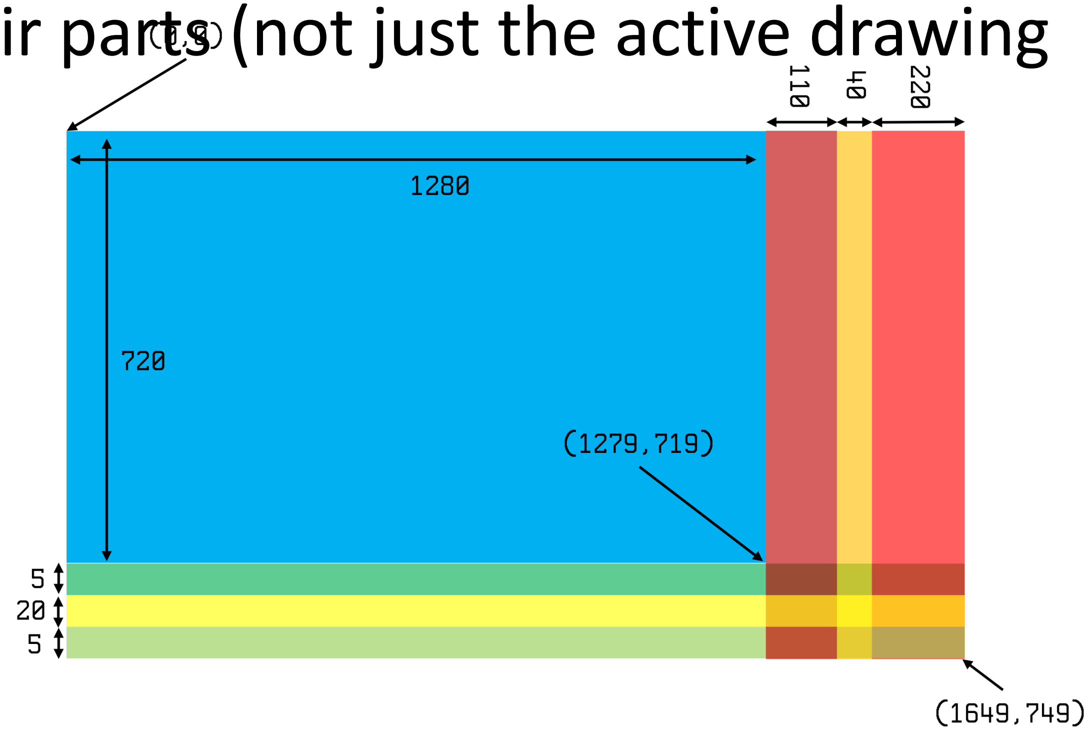
720p

- 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:



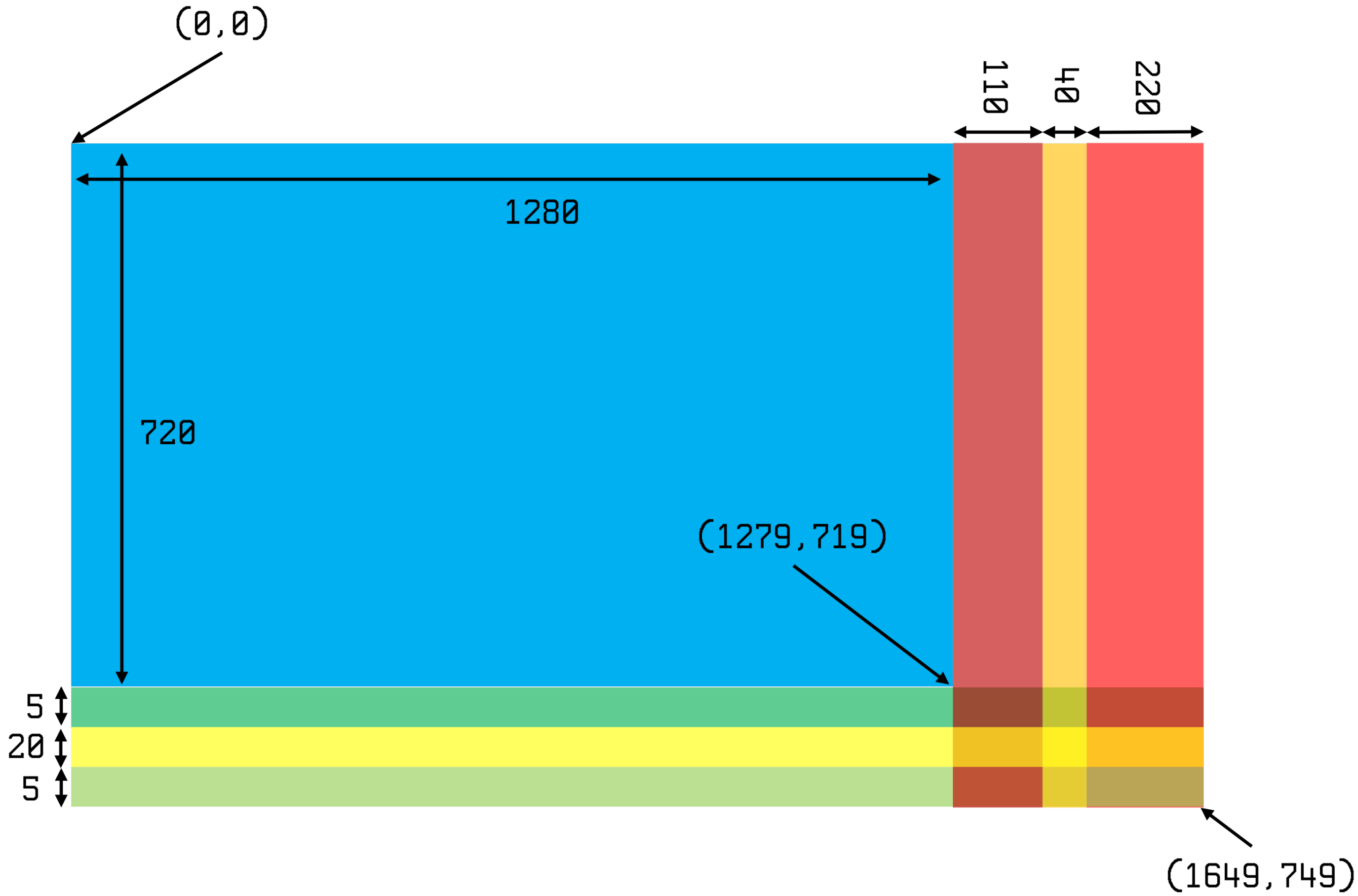
720p

- 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!)



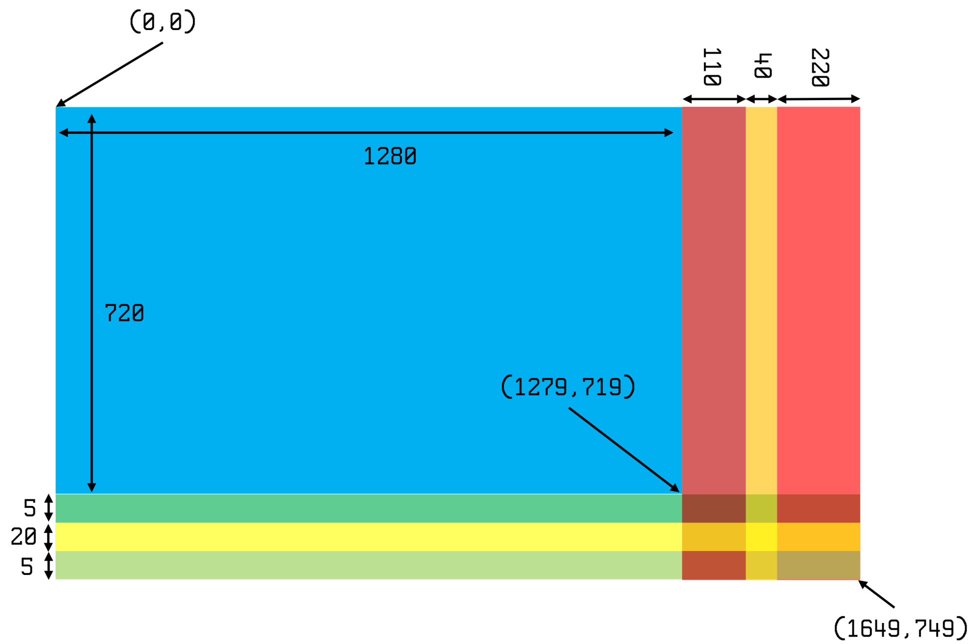
720p Timing

- 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?

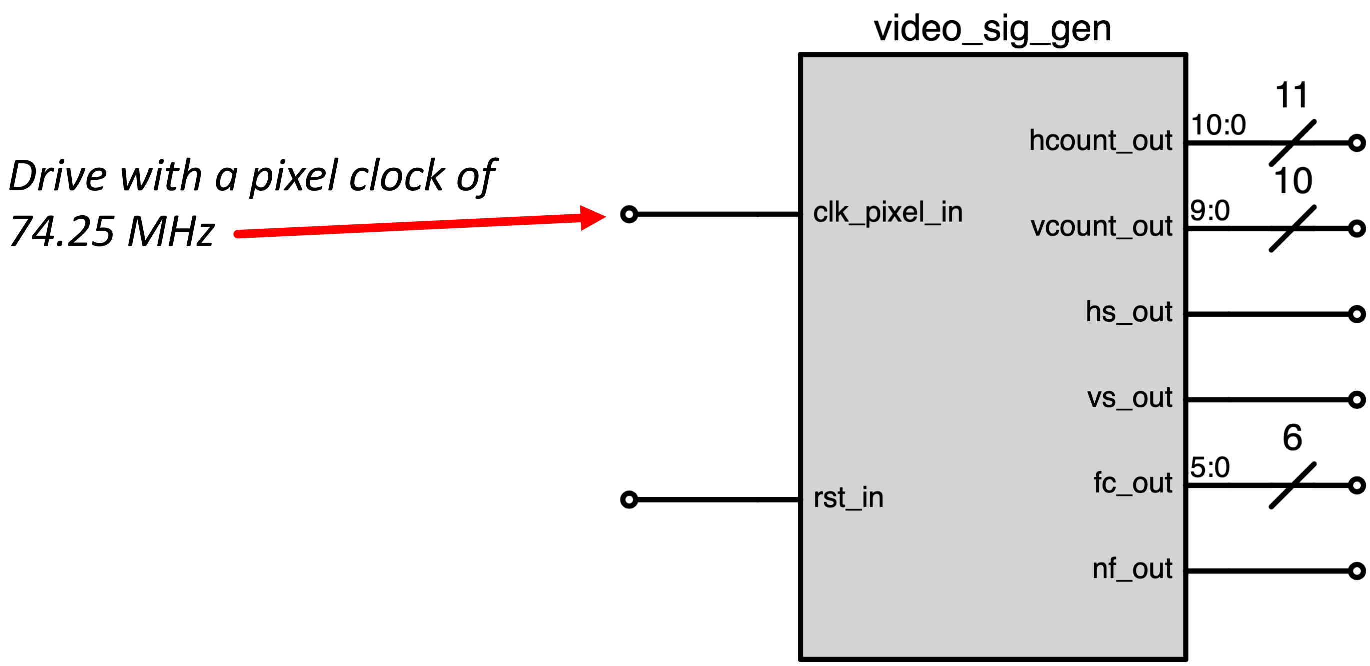
- We’ll be generating 60 fps 720p video. That means we need to deliver 60 full frames per second.
- If 1.2375 million pixels per frame
- 60 frames per second…
- We need to deliver 74.25 Million pixels per second.

- 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.

Week 04 Part 1

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

- `hcount_out`: The current horizontal count on the screen.
- `vcount_out`: The current vertical count on the screen.
- `vs_out`: The vertical sync signal, high when in the vertical sync region
- `hs_out`: The horizontal sync signal, high when in the horizontal sync region
- `ad_out`: The active drawing indicator, low when in blanking or sync period, high when actively drawing.
- `nf_out`: Single cycle indicator of a new frame (see below)
- `fc_out`: Current frame with a rolling second-long window (ranges from 0 to 59 inclusive)



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 nasty like that. All this stuff is irrelevant.

History

- Display technologies and all the associated protocols are a classic example where obsessions with backwards compatibility have 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.

Display Types

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

Back in Time

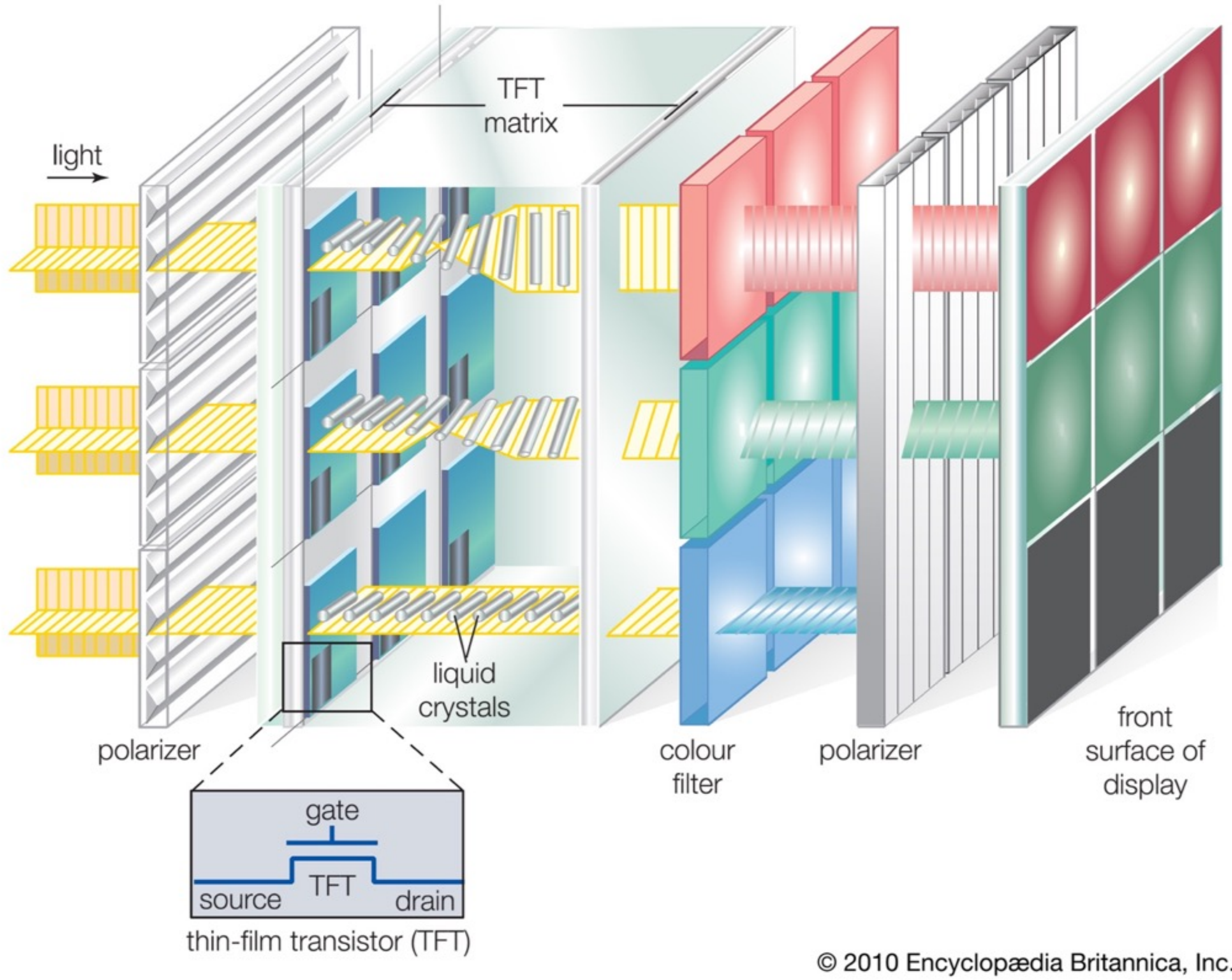
- Reflective Display
  - Electrophoretic Display (E-Ink)*
    - Ultra Low Power – displays are bi-stable, drawing power only when updating the display.
    - Viewable in sunlight – ambient light reflected from display
  - Liquid Crystal Display (LCD)
    - I'm talking old-school calculator style here

Back in Time

*Prof Joseph Jacbson, MIT

TFT LCD

Used to be Cold Cathode
Now almost always white LEDs



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

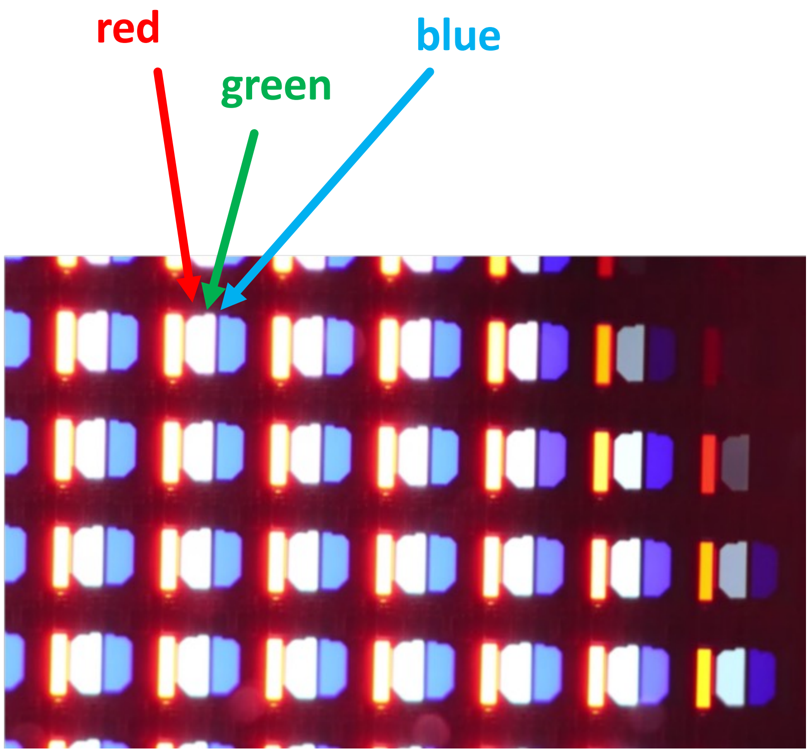
TFT (Thin-Film Transistors)

- Older Technology:
- 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

Organic Light Emitting Diodes

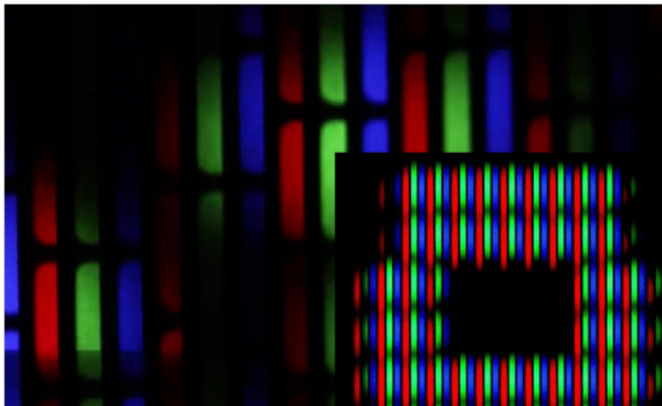
- Newest Technology
- Conceptually maybe the simplest/ideal way to do a display
    1. Gigundous array of RGB LEDs
    2. Control RGB amt. at each point
    3. Profit

1. Want black pixel? Just don't turn on LED

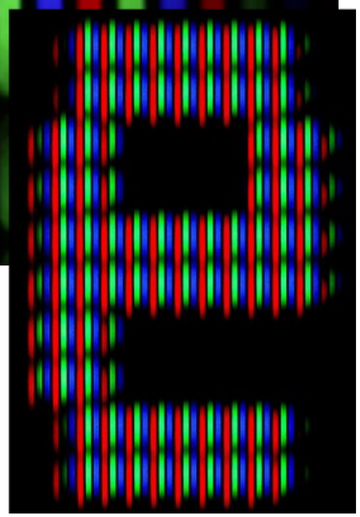


*Green saturated in this image

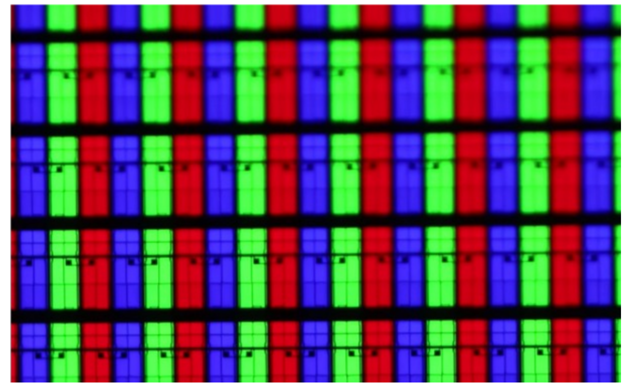
All Color Displays use RGB Pixels



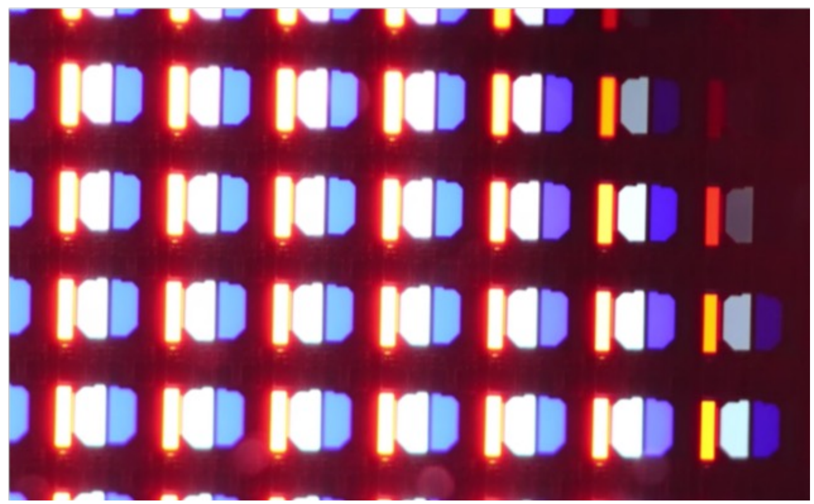
CRT pixels



CRT pixels



TFT LCD pixels



OLED pixels

Slo-Mo Guys

https://www.youtube.com/watch?v=3BJU2drrtCM

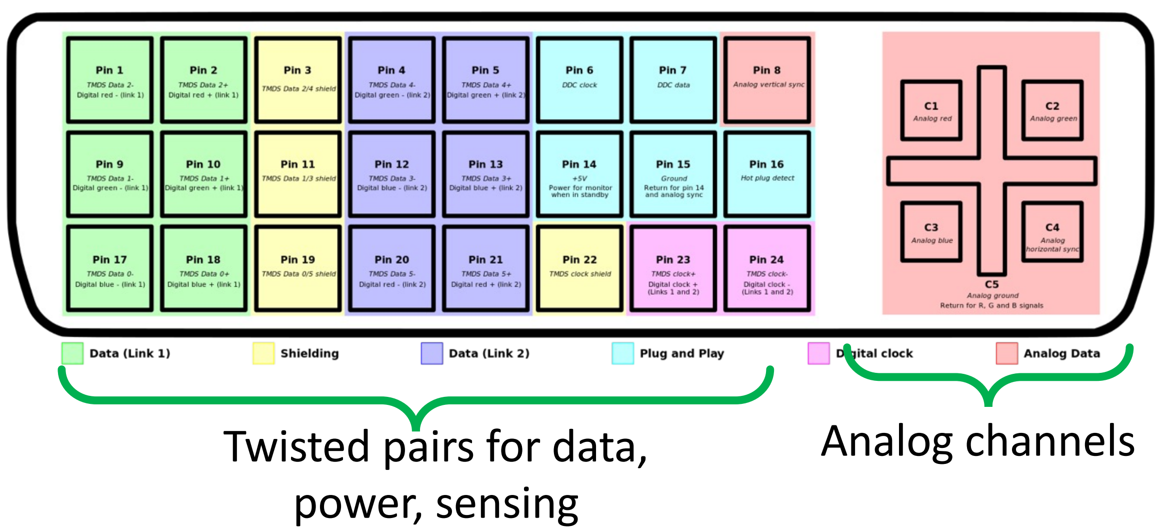
- Video Locations:
  - CRT @2:13
  - TFT LCD @ 7:58
  - OLED @ 10:50

- Whole video is a good watch

DVI (Digital Video Interface)

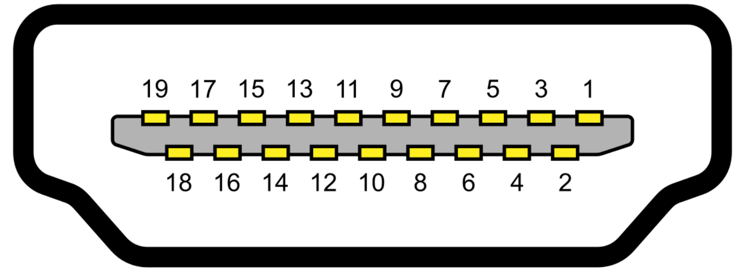
- 1998ish
- Backwards compatible with VGA to an extent (supposed to support analog)
- Sends data digitally over twisted pairs in high-level structure similar to VGA

DB15 Connector



HDMI

- It all starts with the cable and connector



- 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
  - CEC (control things)
  - Power

*each group is a differential signal pair and shield

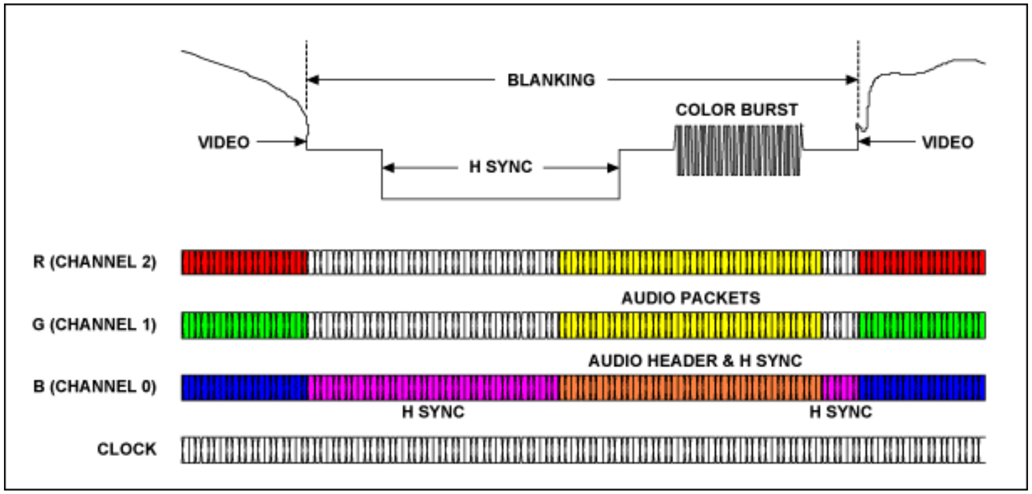
Table 1. HDMI

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

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

Color Information

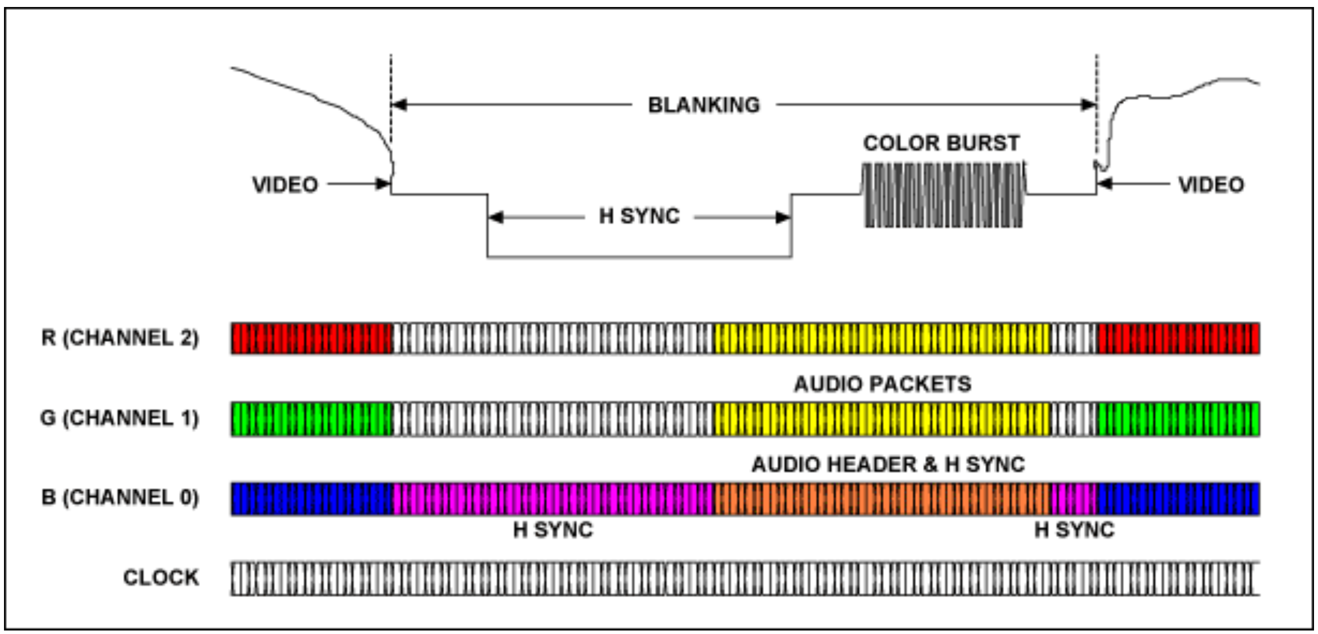
- Sent as serialized data in 10-bit frames using TMDS (week 04)
- One color per pair of wires (red, green, blue wires)
- The blue channel also carries blanking/hsync/vsync info:
  - Encodes those using four 10 bit reserved values:
  - (H = 0, V = 0): 1101010100
  - (H = 1, V = 0): 0010101011
  - (H =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



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

Audio Information

- 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



Traditional Video

HDMI

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, etc

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

HDMI Data Transfer

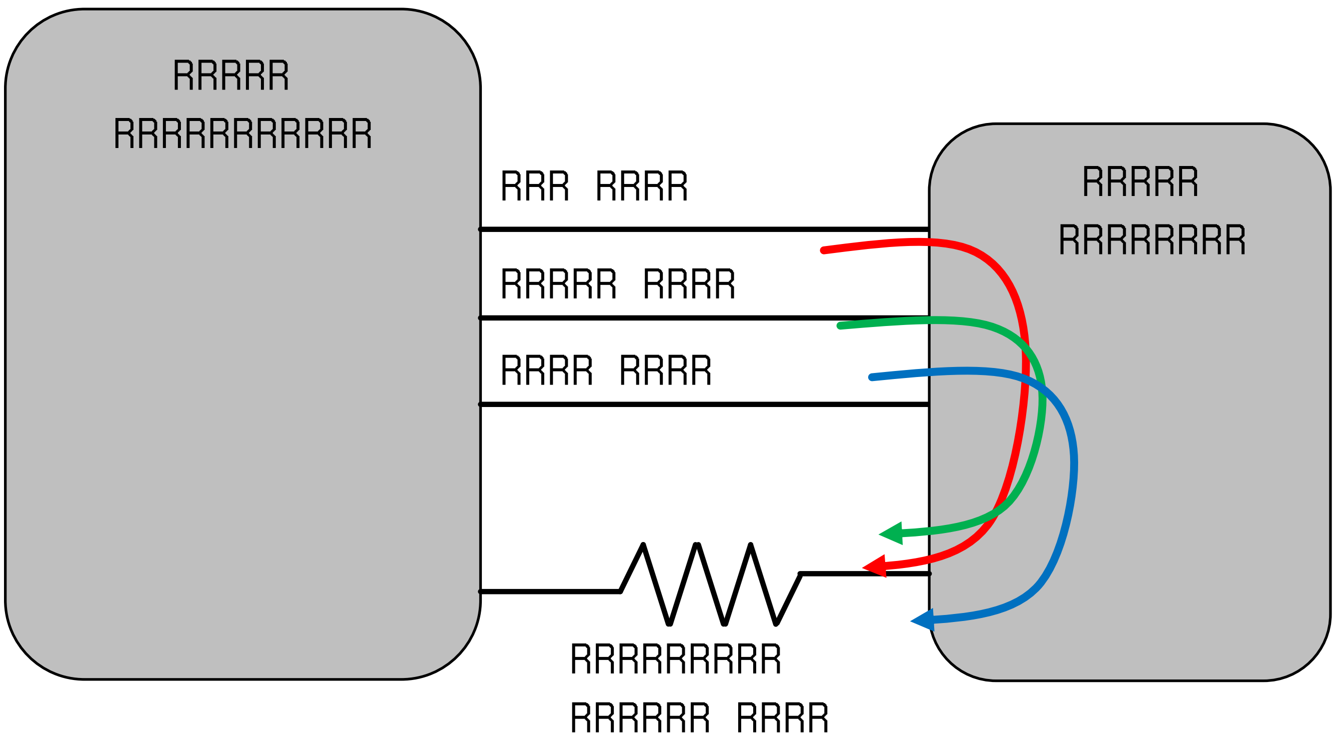
- Modern displays are built around sending the red, green, and blue signals on their own channels.
- Each channel sends that portion of a pixel's information serially.
- For 720p we're sending 74.25 million pixels per second.
- If we were to do the 8 bits of serially that means the red, green, and blue 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→1→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 ***T**ransitions **M**inimized*

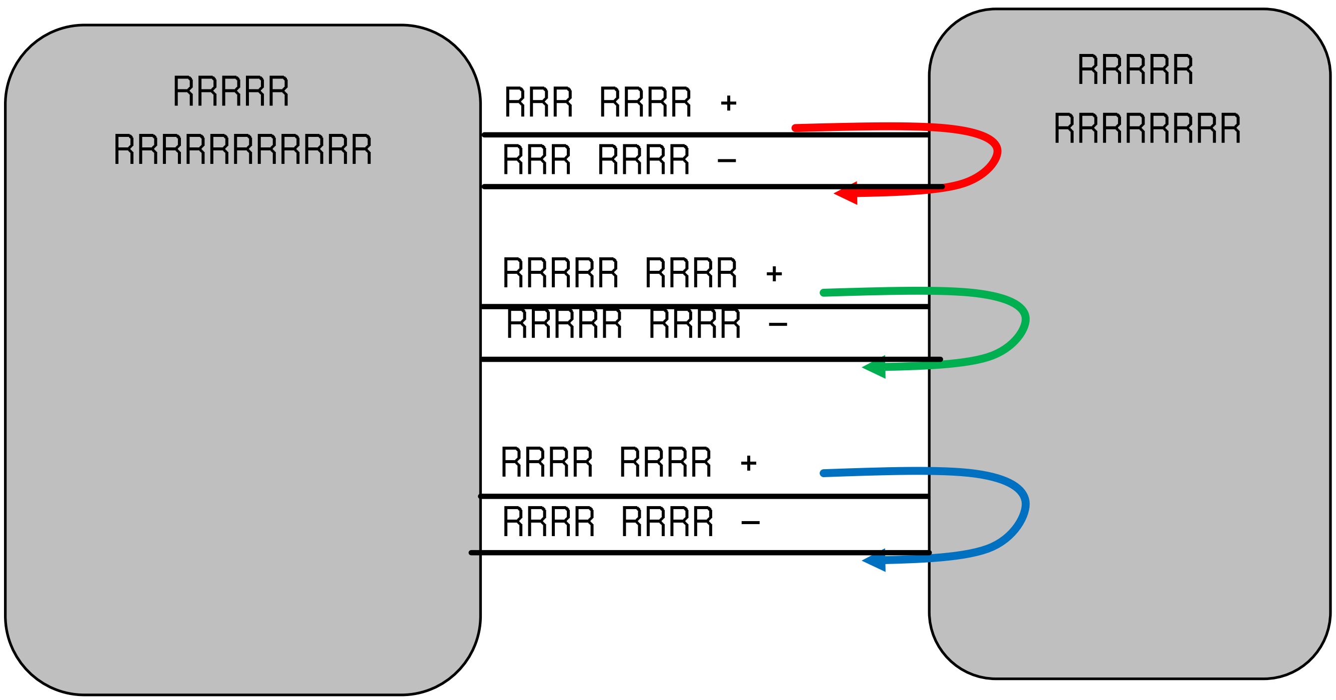
Long Distances

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

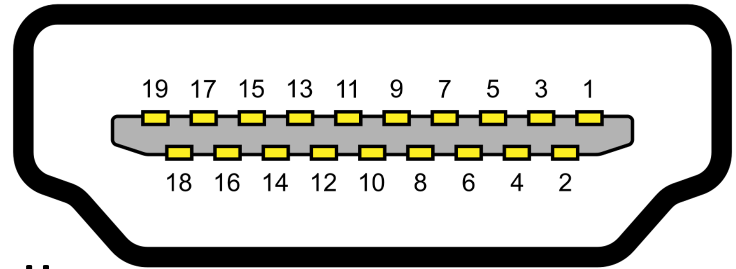


Long Distances

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



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:
  - 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. 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 do this in Lab 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"

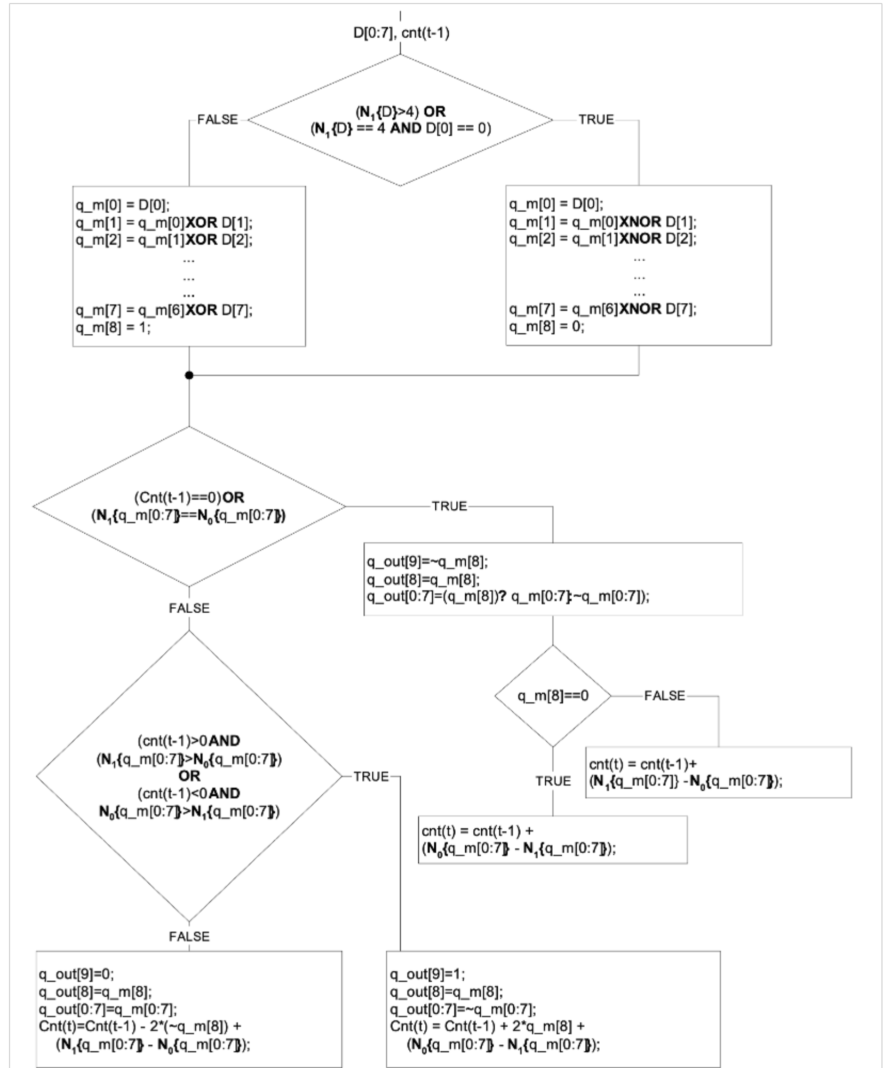
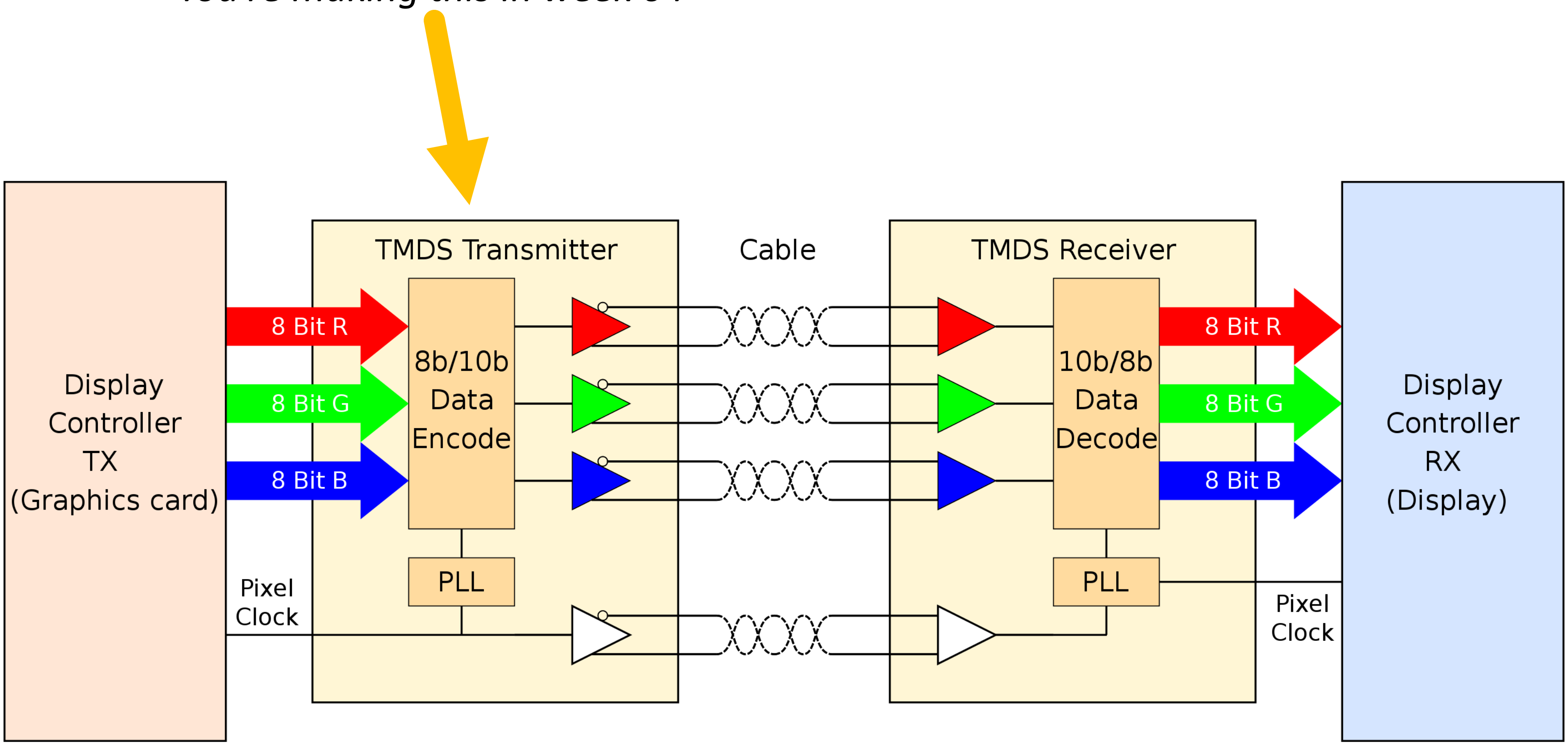


Figure 5-7 TMDS Video Data Encode Algorithm

TMDS system

You're making this in week 04



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

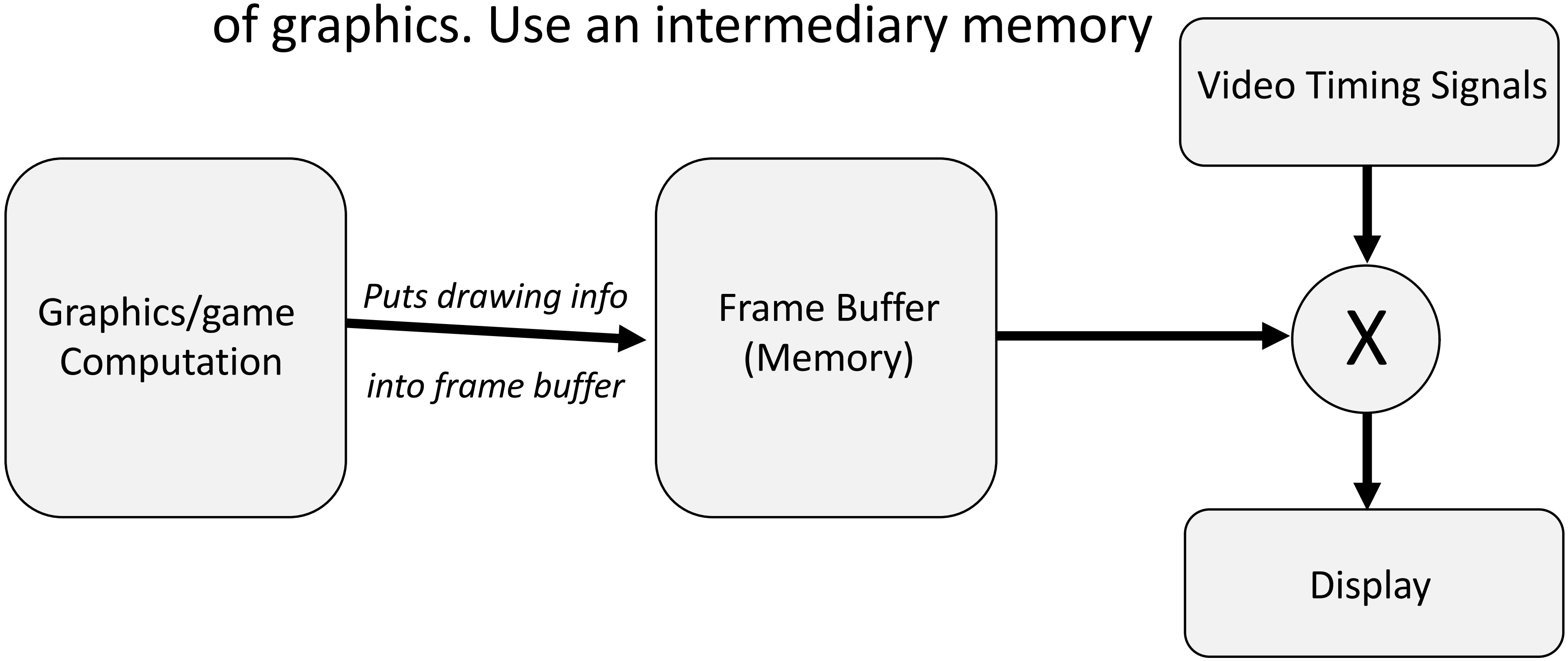
Conclusion

- 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:

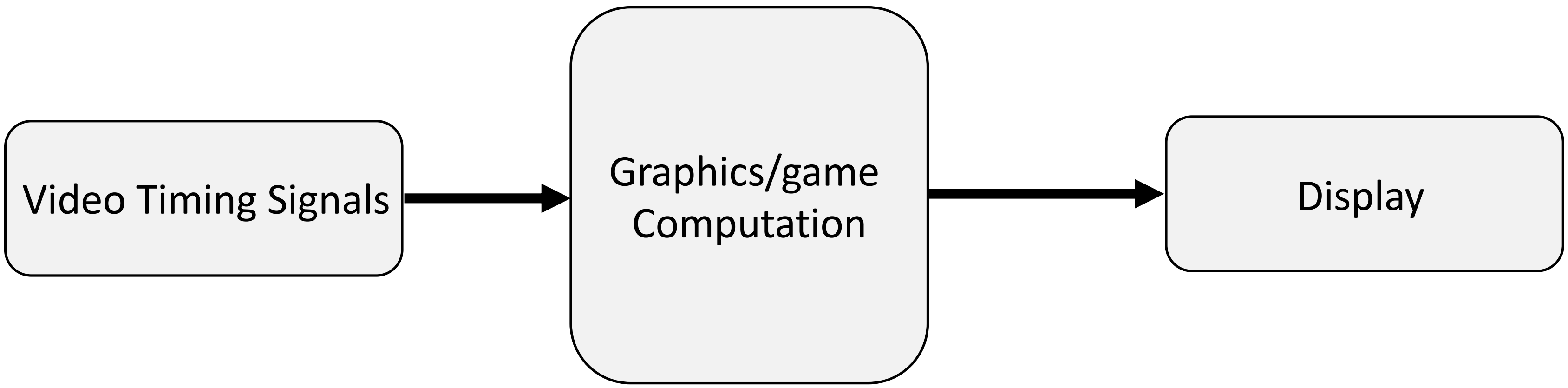
- Way One: Frame Buffer:
  - Separate computation of drawing from actual rendering of graphics. Use an intermediary memory



More modern way of doing it (need lots of memory though!)

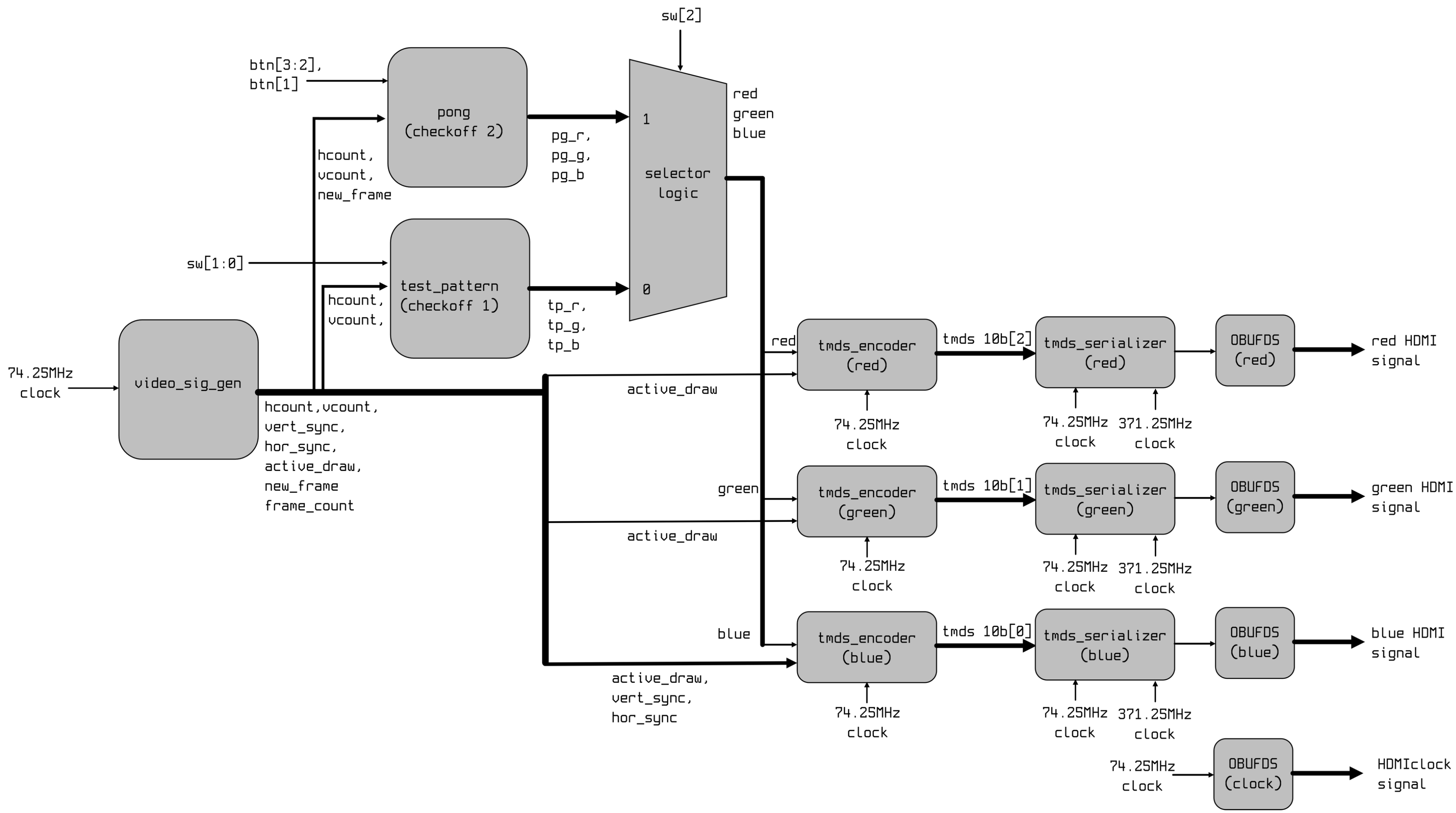
Two General Ways to Produce Video:

- Way Two: “Racing 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*

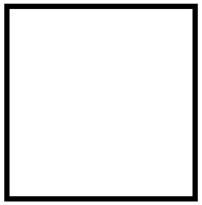
Lab 04 Setup (Racing the Beam)



Examples of pixel logic:

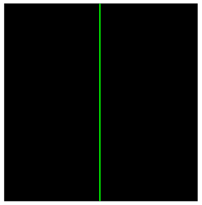
Whole Screen is White:

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



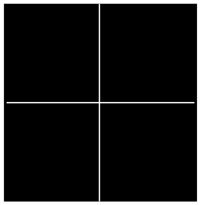
Draw green vertical line at horizontal spot 500:

```
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
```

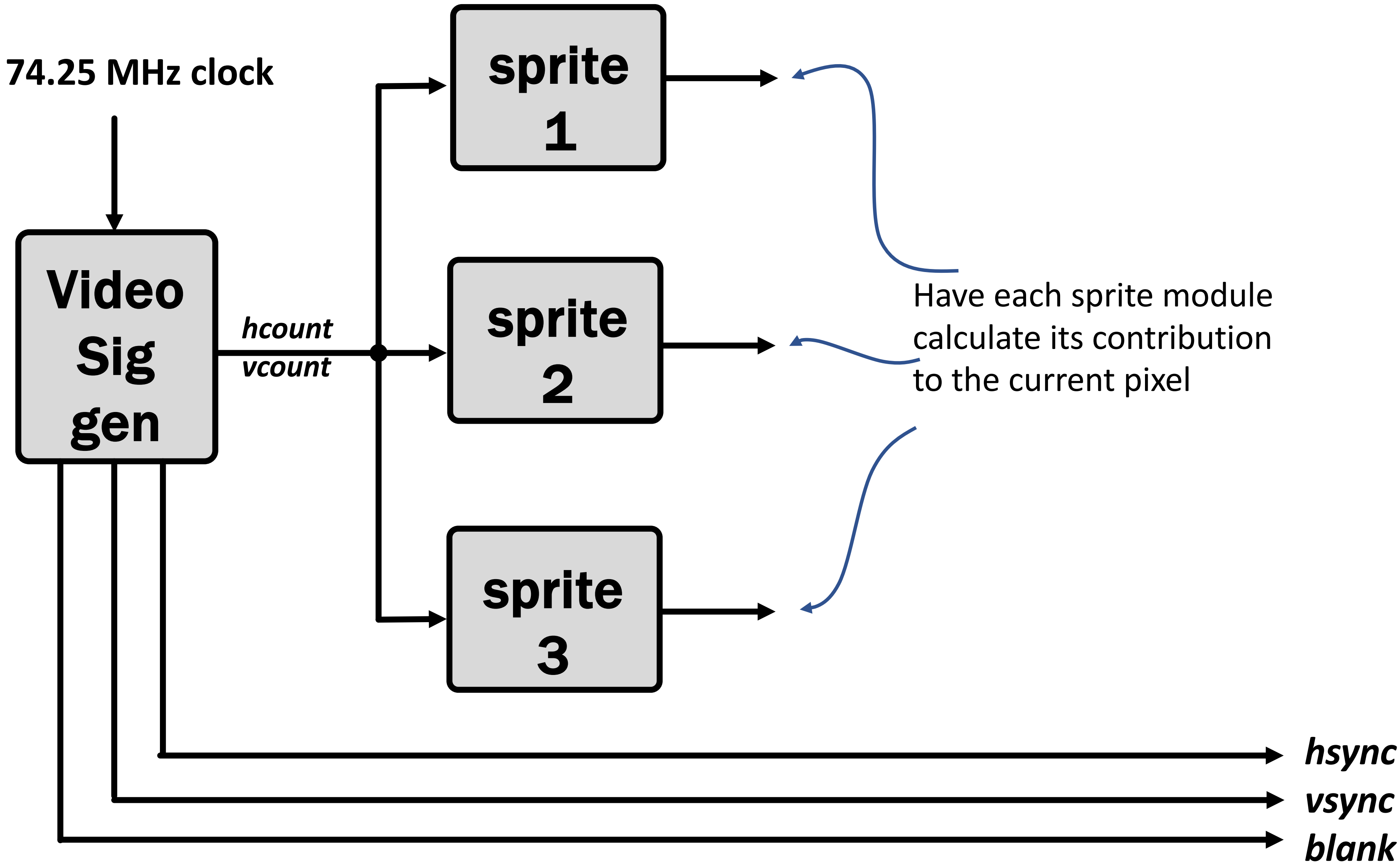


Examples of Pixel Logic

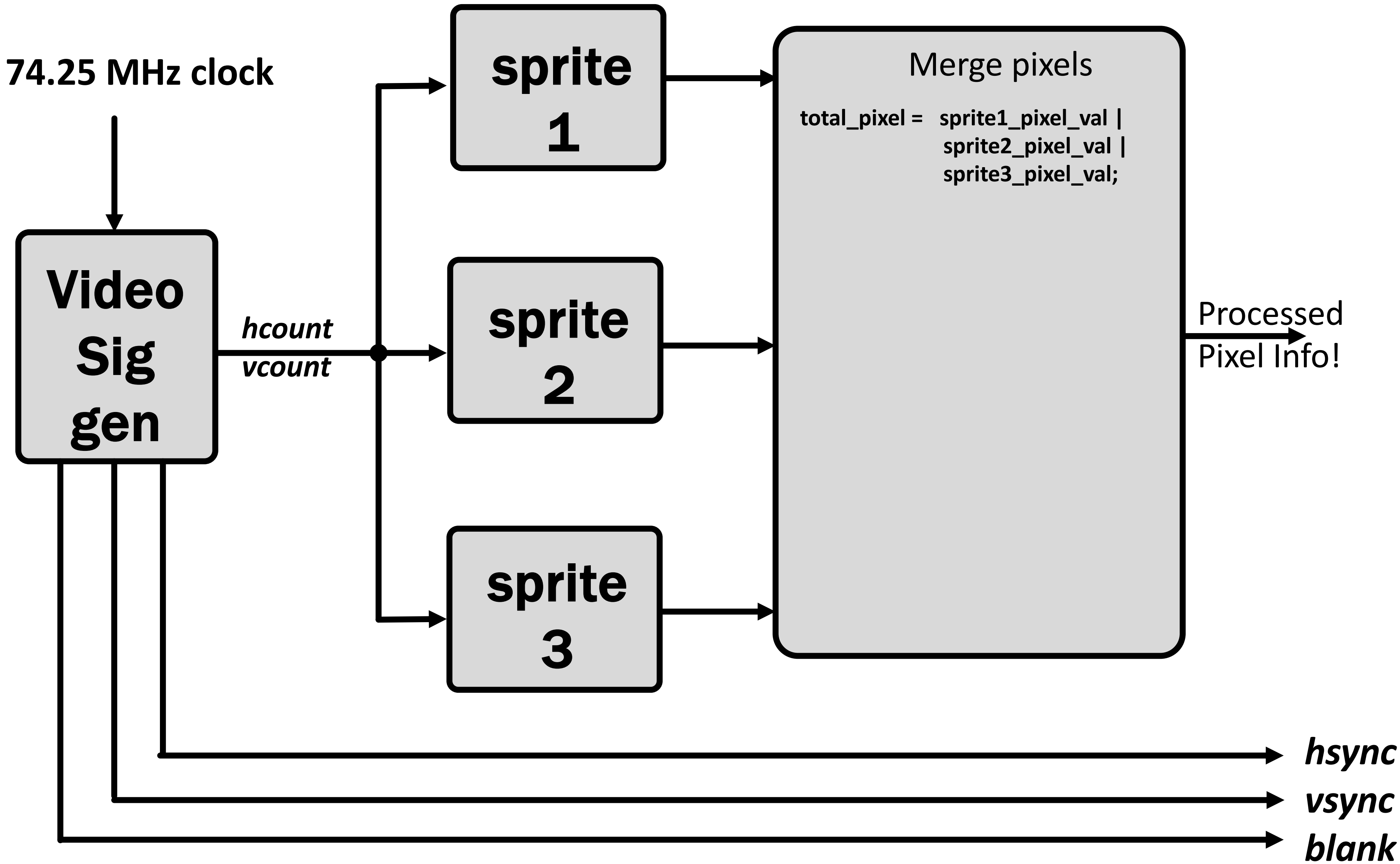
- From Lab 04:
- Draw a white pixel if within a set of rectangular bounds!

```
module block_sprite #(
  parameter WIDTH=128, HEIGHT=128, COLOR=24'hFF_FF_FF)(
  input wire [10:0] hcount_in,
  input wire [9:0] vcount_in,
  input wire [10:0] x_in,
  input wire [9:0] y_in,
  output logic [7:0] red_out,
  output logic [7:0] green_out,
  output logic [7:0] blue_out);
  logic in_sprite;
  assign in_sprite = ((hcount_in >= x_in && hcount_in < (x_in + WIDTH))
                &&(vcount_in >= y_in && vcount_in < (y_in + HEIGHT)));
  always_comb begin
    if (in_sprite)begin
      red_out = COLOR[23:16];
      green_out = COLOR[15:8];
      blue_out = COLOR[7:0];
    end else begin
      red_out = 0;
      green_out = 0;
      blue_out <= 0;
    end
  end
endmodule
```

How can a drawing start to work?



How can a drawing start to work?



How can a drawing start to work?