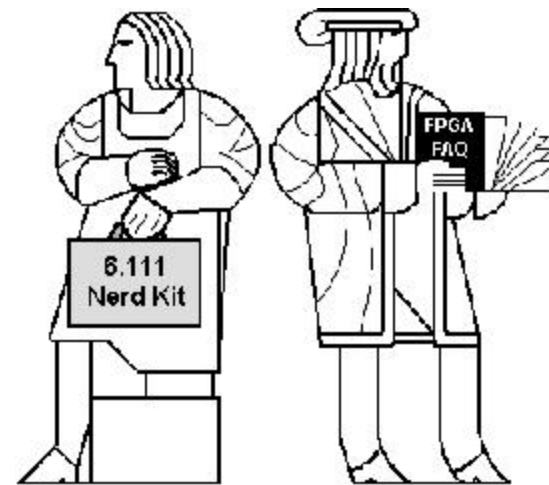


6.111 Lecture 15

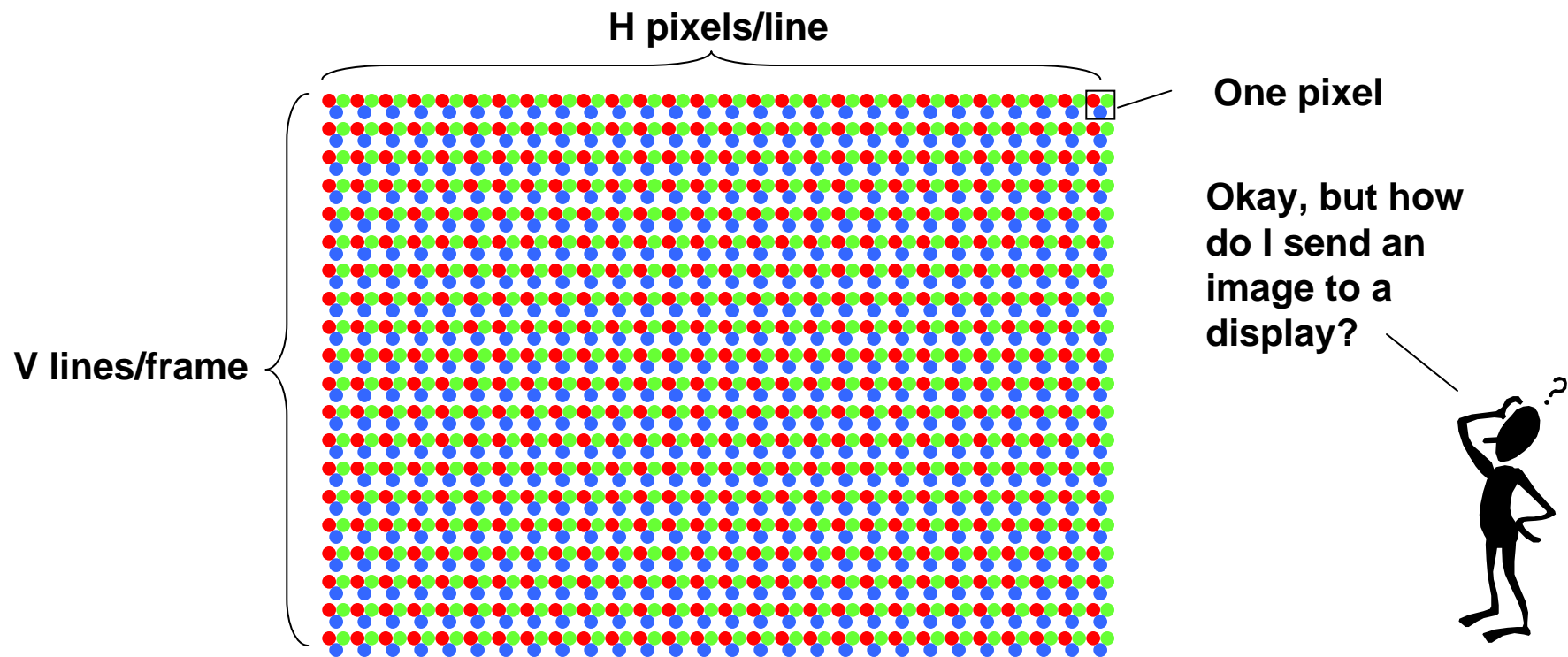
Today: Video

1. Cathode Ray Tubes
2. NTSC
3. Video Decode/Encode
4. VGA Video
5. Demo



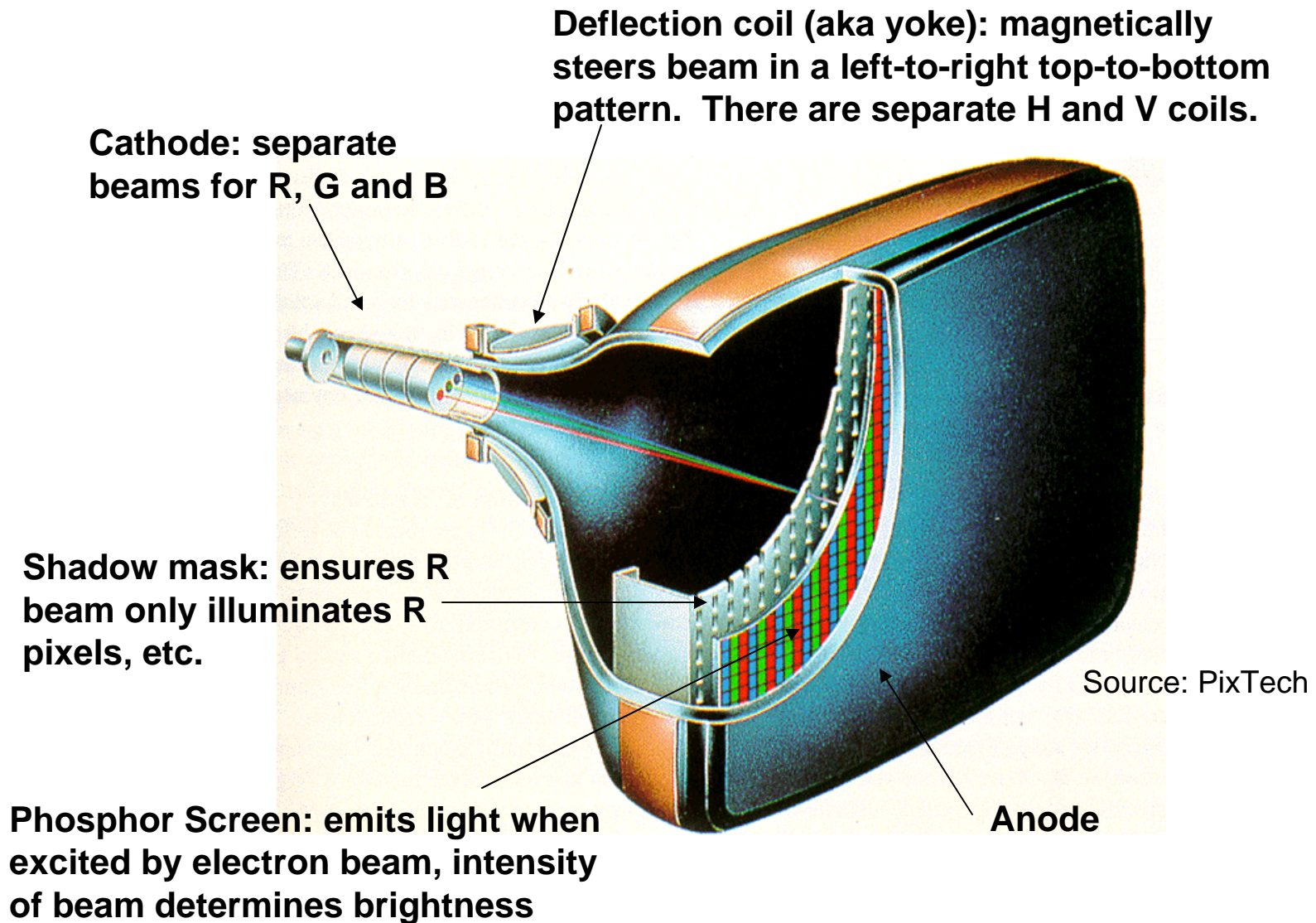
1. The CRT: Generalized Video Display

Think of a color video display as a 2D grid of picture elements (pixels). Each pixel is made up of red, green and blue (RGB) emitters. The relative intensities of RGB determine the apparent color of a particular pixel.

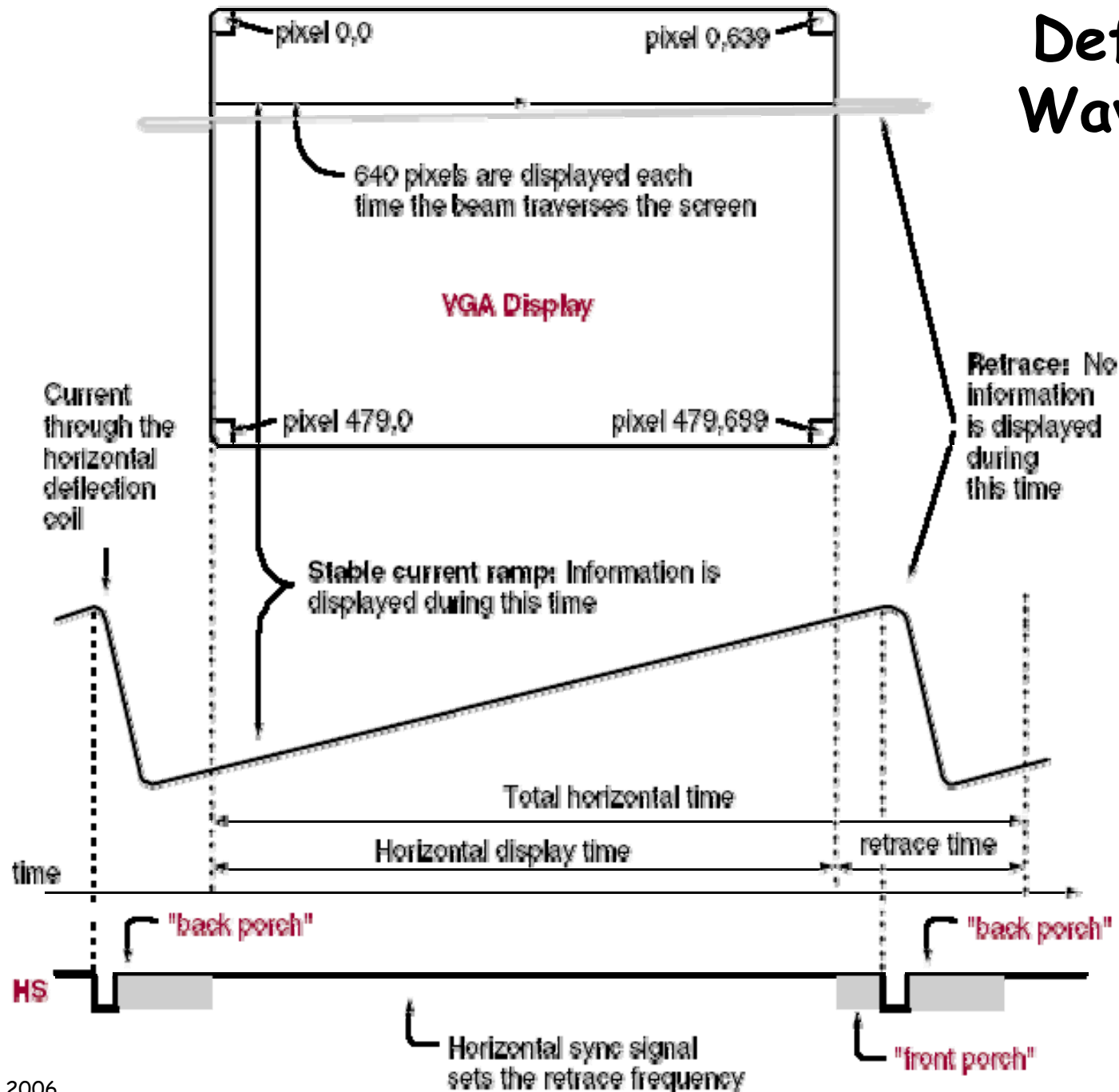


Traditionally $H/V = 4/3$ or with the advent of high-def $16/9$.
Lots of choices for H, V and display technologies (CRT, LCD, ...)

Background: Cathode Ray Tubes

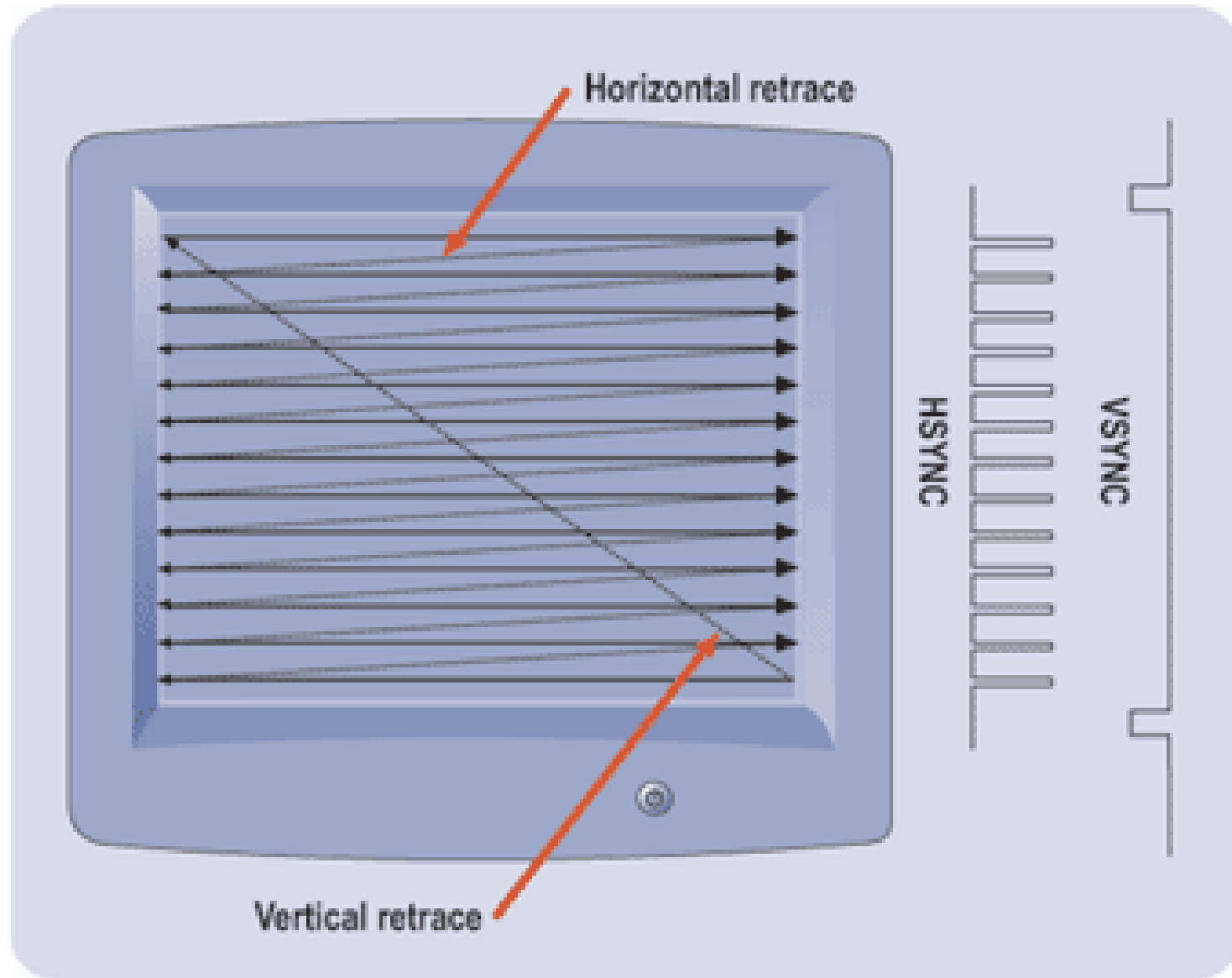


Deflection Waveforms



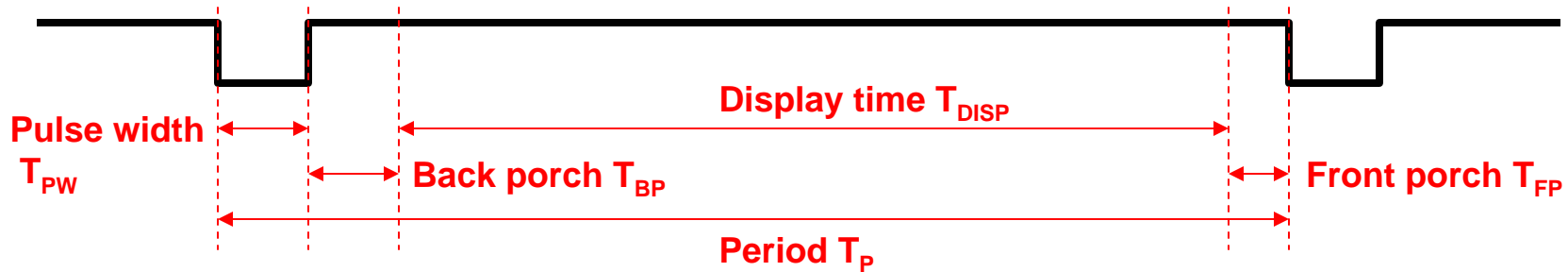
Source: Xilinx Spartan-3 Starter Kit Board User Guide

Sync Signals (HS and VS)



Sync Signal Timing

The most common ways to send an image to a video display (even displays that don't use deflection coils, eg, LCDs) require you to generate two sync signals: one for the horizontal dimension (HS) and one for the vertical dimension (VS).



<i>Format</i>		<i>CLK</i>	<i>P</i>	<i>PW</i>	<i>BP</i>	<i>DISP</i>	<i>FP</i>
VGA	HS (pixels)	25Mhz	794	95	47	640	13
	VS (lines)	--	528	2	33	480	13
XGA	HS (pixels)	65Mhz	1344	136	160	1024	24
	VS (lines)	--	806	6	23	768	9

Interlace

Non-interlaced (aka progressive) scanning:

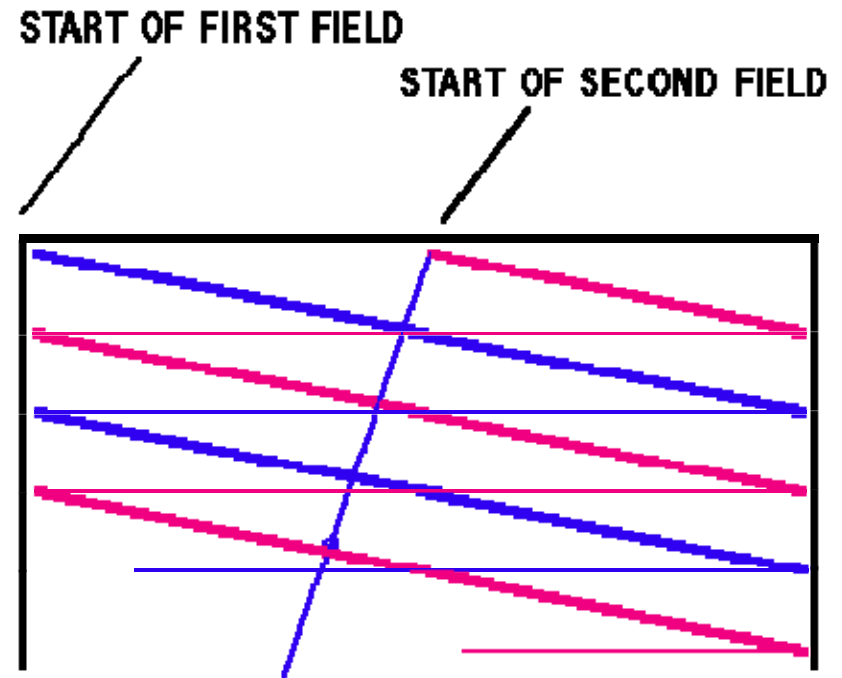
- VS period is a multiple of HS period
- Frame rate \geq 60Hz to avoid flicker

Interlaced scanning:

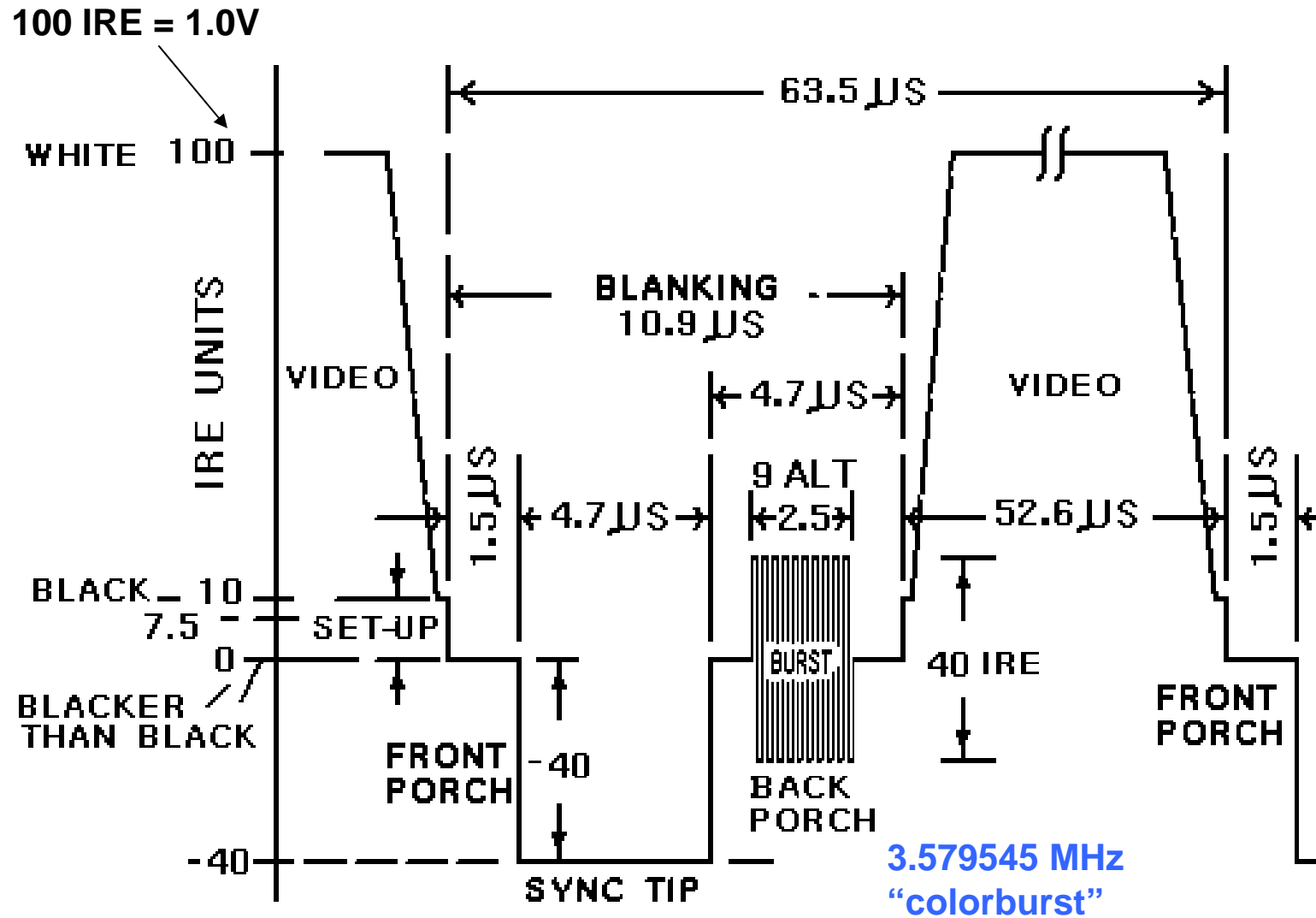
VS period is *not* a multiple of HS period, so successive vertical scan are offset relative to horizontal scan, so vertical position of scan lines varies from frame to frame.

NTSC example:

- 525 total scan lines (480 displayed)
- 2 fields of 262.5 scan lines (240 displayed).
- Field rate is 60Hz, frame rate = 30Hz



2. NTSC*: Composite Video Encoding

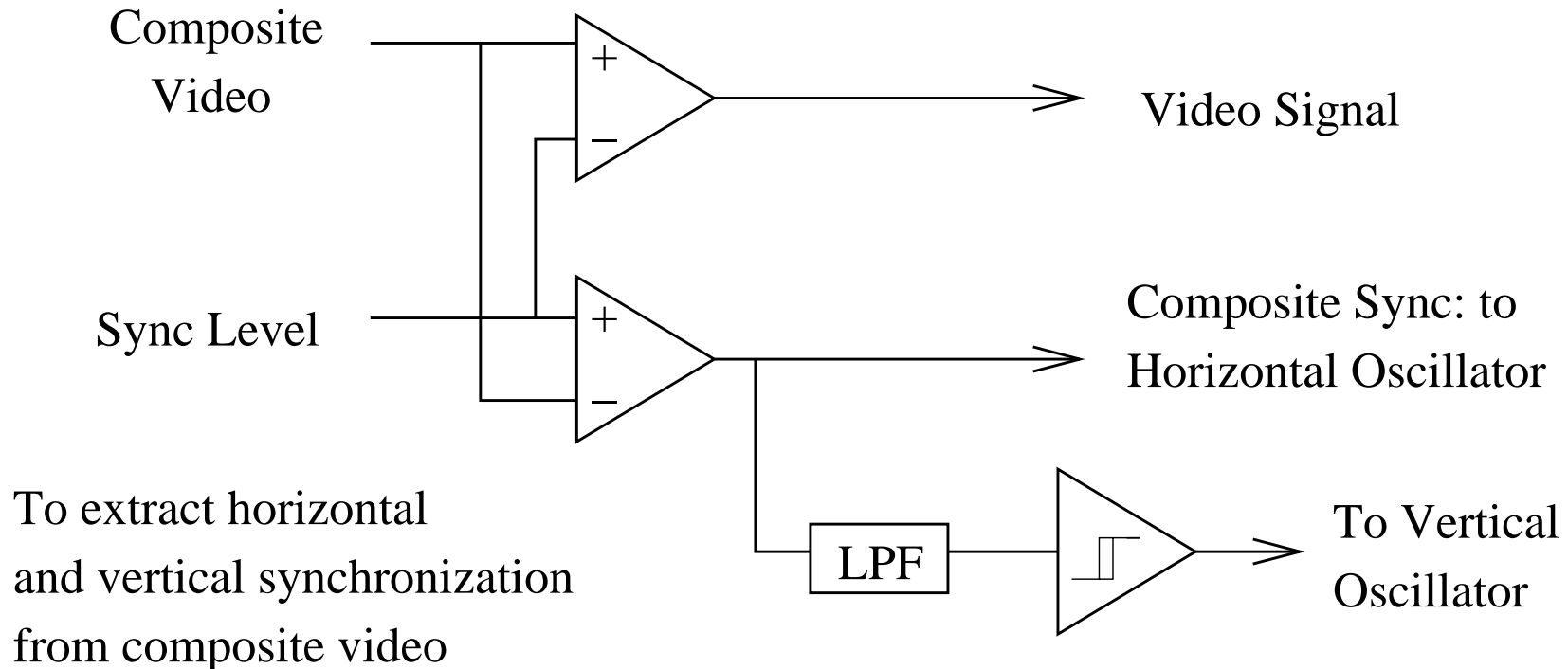


*National Television System Committee: 1940

Source: <http://www.ntsc-tv.com>

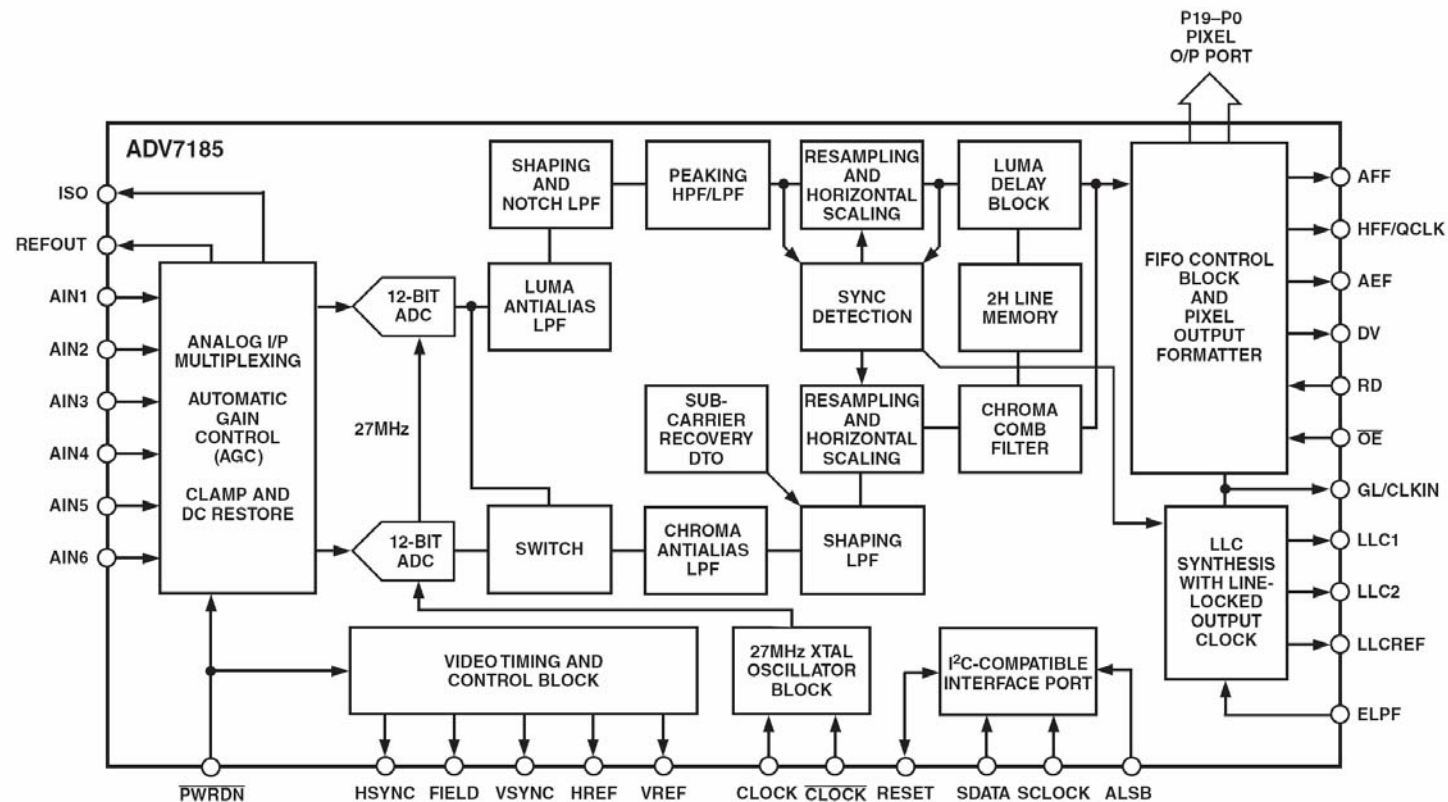
3. Video Capture: Signal Recovery

- Composite video has picture data and both syncs.
 - Picture data (video) is above the sync level.
 - Simple comparators extract video and composite sync.
- Composite sync is fed directly to the horizontal oscillator.
- A low-pass filter is used to separate the vertical sync.
 - The edges of the low-passed vertical sync are squared up by a Schmidt trigger.



Labkit: ADV7185 NTSC Decoder

- Decodes NTSC and PAL video (composite or S-video)
- Produces CCIR656 (10-bit) or CCIR601 (8-bit) digital data



Labkit: ADV7185 NTSC Decoder

- Decodes NTSC and PAL video (composite or S-video)
- Produces CCIR656 (10-bit) or CCIR601 (8-bit) digital data

BLANKING PERIOD			TIMING REFERENCE CODE				720 PIXELS YUV 4 : 2 : 2 DATA										TIMING REFERENCE CODE				BLANKING PERIOD		
...	80	10	FF	00	00	SAV	C _{B0}	Y ₀	C _{R0}	Y ₁	C _{B2}	Y ₂	...	C _{R718}	Y ₇₁₉	FF	00	00	EAV	80	10	...	

NAME	EXPLANATION
SAV	start of active video range
C _{Bn}	U (B - Y) colour difference component, pixel number n = 0, 2, 4 to 718
Y _n	Y (luminance) component, pixel number n = 0, 1, 2, 3 to 719
C _{Rn}	V (R - Y) colour difference component, pixel number n = 0, 2, 4 to 718
EAV	end of active video range

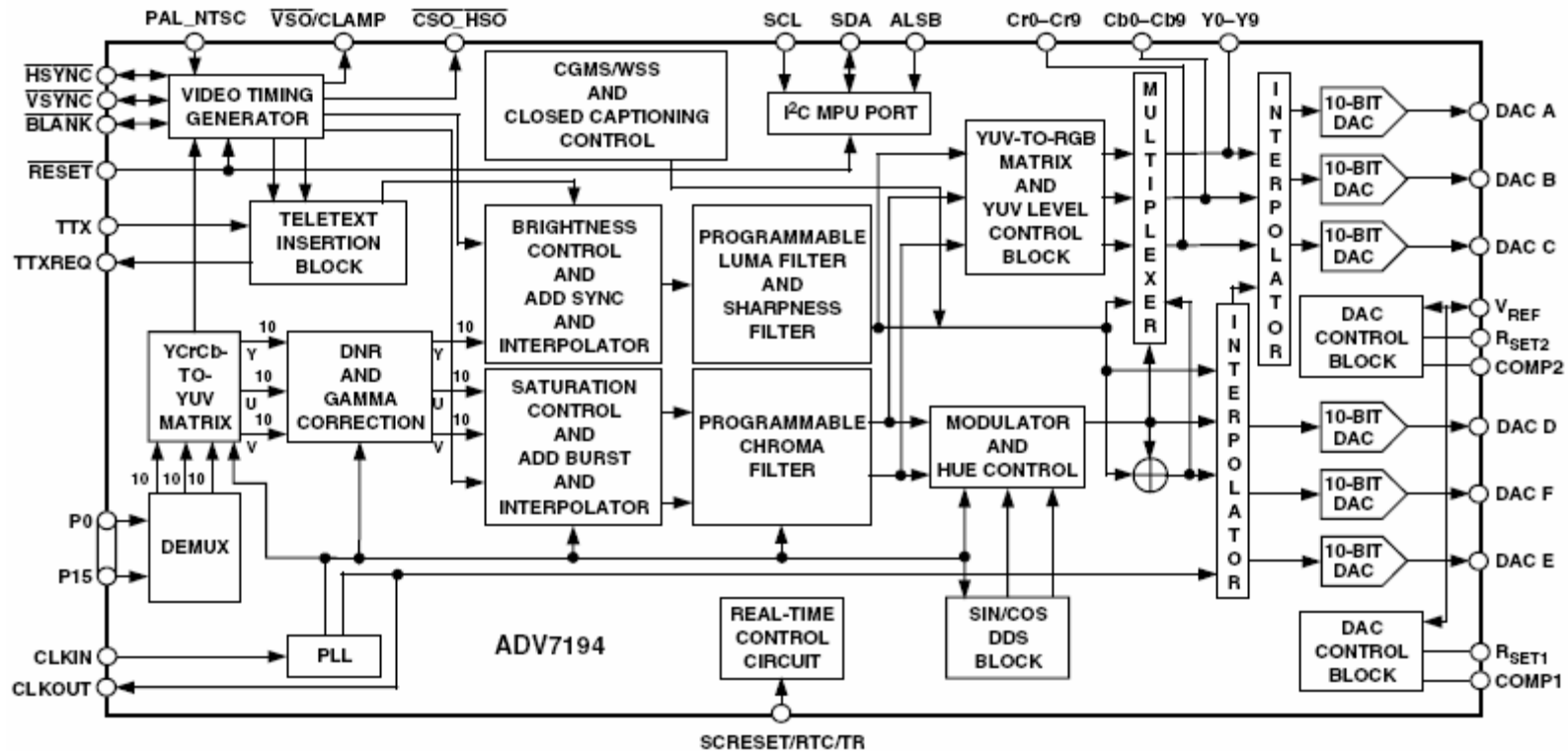
- YUV: Y=brightness, U & V = color ; C_B → U, C_R → V

$$R = Y + 1.402 V$$

$$G = Y - 0.344 U - 0.714 V$$

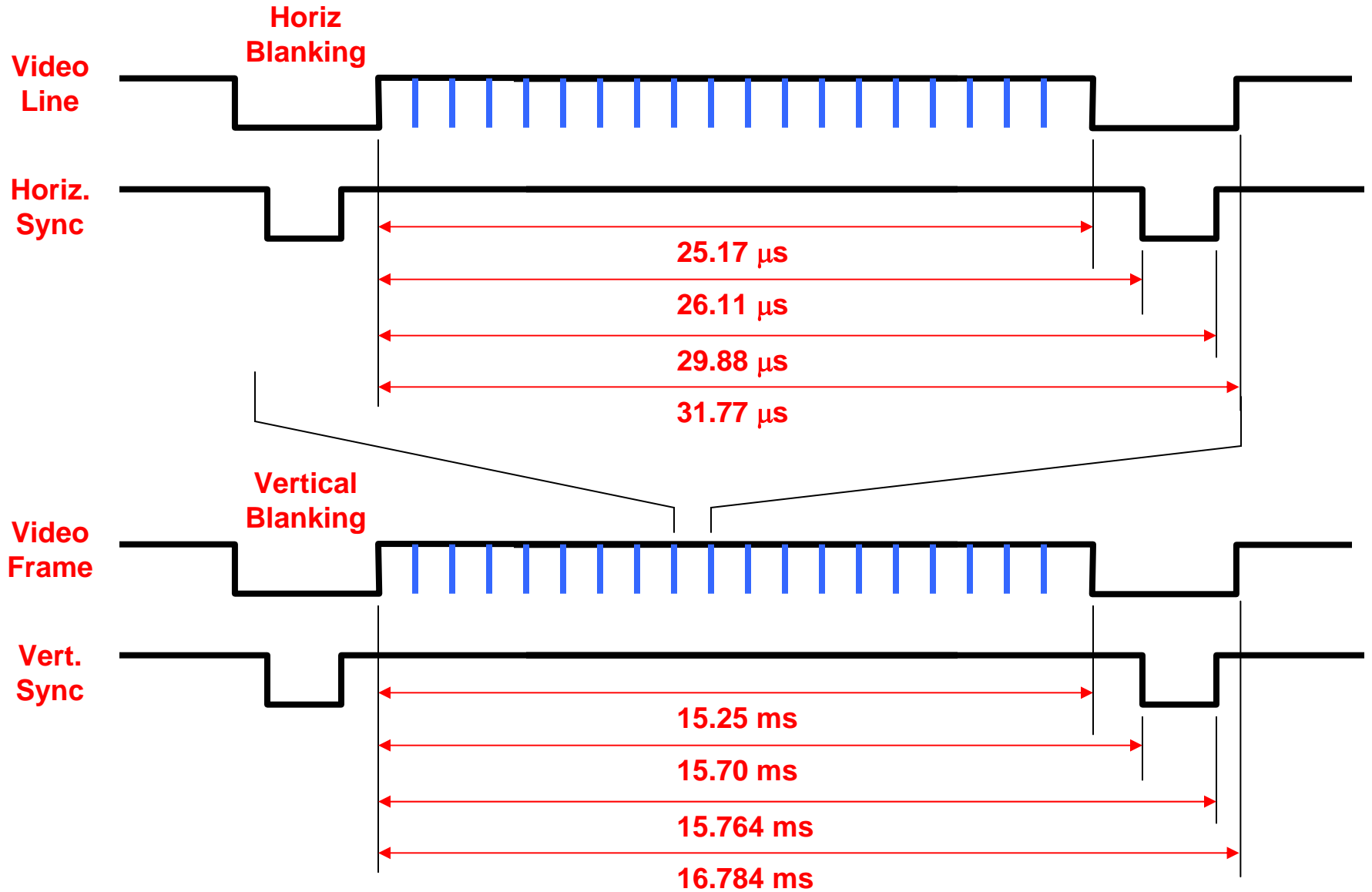
$$B = Y + 1.772 U$$

Labkit: AD7194 Digital Video Encoder



CCIR 601/656 4:2:2 digital video data → analog baseband TV signal

4. VGA Video



Labkit: ADV7125 Triple DAC (VGA)

- Two Challenges:

- (1) **Generate Sync Signals**

- Sync signal generation requires precise timing
- Labkit comes with **27 MHz clock**
- Use **phase-locked-loops (PLL)** to create higher frequencies
- Xilinx FPGA's have a "Digital Clock Manager" (**DCM**)

```
DCM pixel_clock (.CLKIN(clock_27mhz), .CLKFX(pixel_clock);
```

```
// synthesis attribute CLKFX_DIVIDE of pixel_clock is 10
```

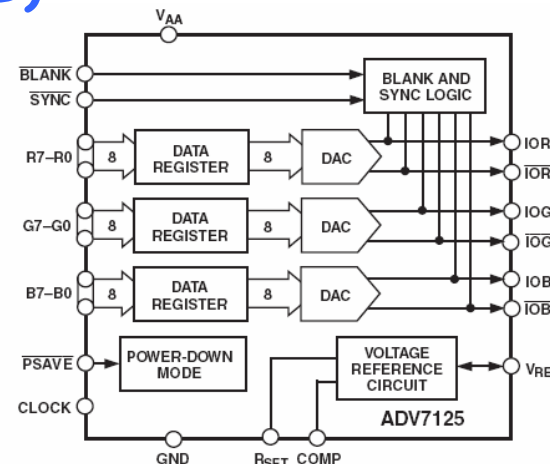
```
// synthesis attribute CLKFX_MULTIPLY of pixel_clock is 24
```

$$27 \text{ MHz} * 24 / 10 = 64.8 \text{ MHz}$$

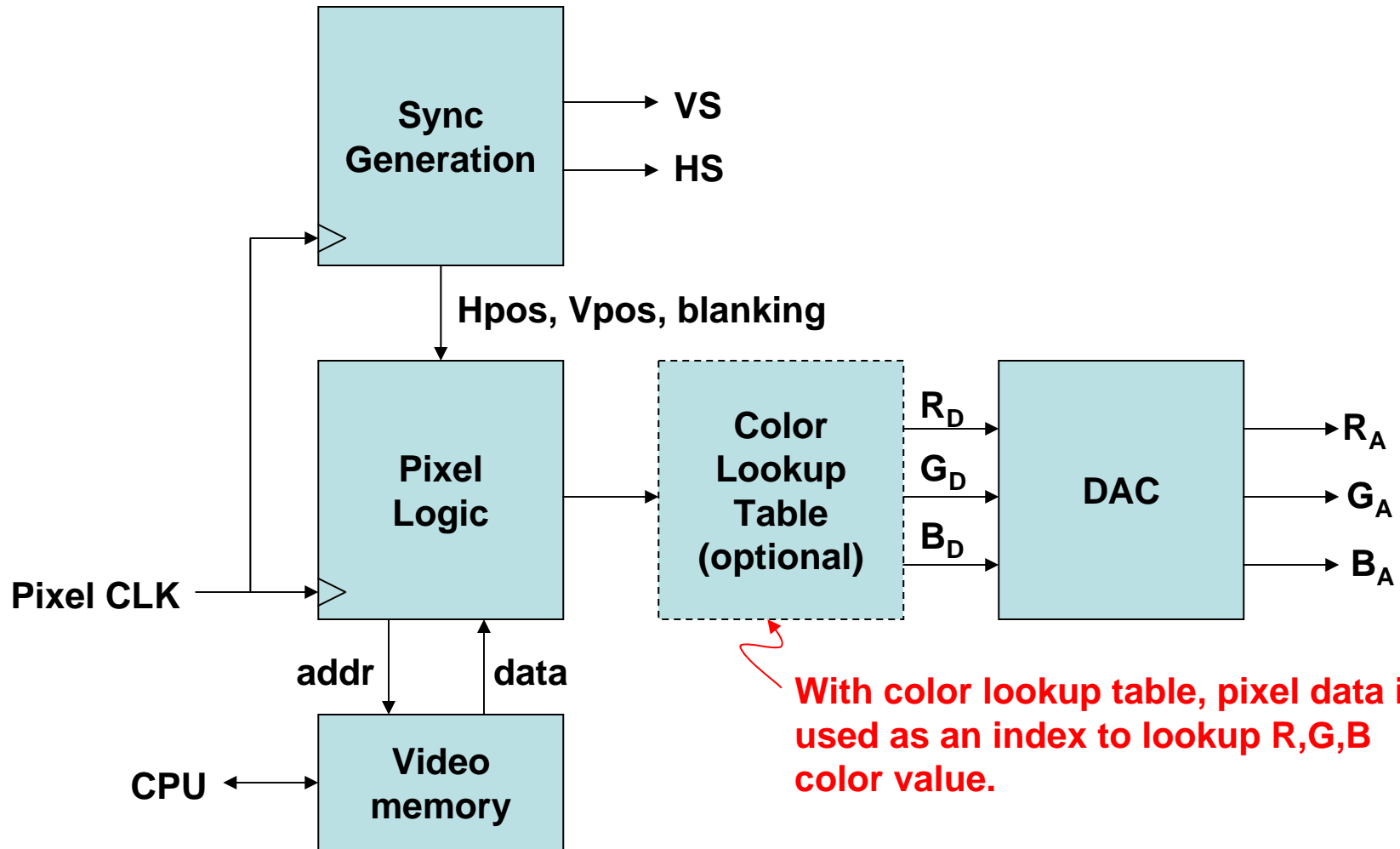
→Used in Lab5!

- (2) **Generate Video Pixel Data (RGB)**

- Use ADV7125 Triple DAC
- Send 24 bits of R,G,B data at pixel clock rate to chip
- Create pixels either in real time
- Or using dual port RAM
- Or from character maps
- Or ...?



Generating VGA-style Video

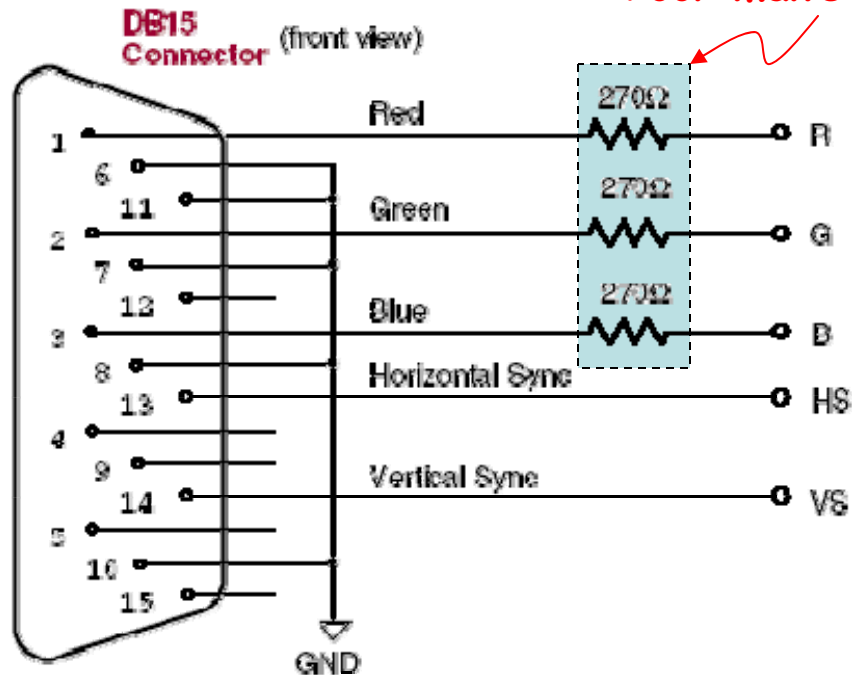


With color lookup table, pixel data is used as an index to lookup R,G,B color value.

Without color lookup table, pixel data is used directly as R,G,B value (aka "true color")

Simple VGA Interface for FPGA

Poor man's Video DAC



Your circuitry should produce TTL-level signals (3.3V high level)

HS, VS are active-low signals.

R, G, B are active-high.

Shown: a simple "8-color" scheme

The R, G and B signals are terminated with 75 Ohms to ground inside of the VGA monitor. So when you drive your 3.3V signal through the 270 Ohm series resistor, it shows up at the monitor as 0.7V - exactly what the VGA spec calls for.

$$0.7V = \left(\frac{75}{75 + 270}\right)(3.3V)$$

Verilog: XVGA Display (1024x768)

```
module xvga(clk,hcount,vcount,hsync,vsync);
    input clk;                // 64.8 Mhz
    output [10:0] hcount;
    output [9:0] vcount;
    output hsync, vsync;
    output [2:0] rgb;

    reg hsync,vsync,hblank,vblank,blank;
    reg [10:0] hcount;        // pixel number on current line
    reg [9:0] vcount;        // line number

    wire hsyncon,hsyncoff,hreset,hblankon; // next slide for generation
    wire vsyncon,vsyncoff,vreset,vblankon; // of timing signals

    wire next_hb = hreset ? 0 : hblankon ? 1 : hblank; // sync & blank
    wire next_vb = vreset ? 0 : vblankon ? 1 : vblank;

    always @(posedge clk) begin
        hcount <= hreset ? 0 : hcount + 1;
        hblank <= next_hb;
        hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync; // active low

        vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
        vblank <= next_vb;
        vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync; // active low
    end
end
```

XVGA (1024x768) Sync Timing

```
// assume 65 Mhz pixel clock

// horizontal: 1344 pixels total
// display 1024 pixels per line
assign hblankon = (hcount == 1023); // turn on blanking
assign hsynccon = (hcount == 1047); // turn on sync pulse
assign hsyncoff = (hcount == 1183); // turn off sync pulse
assign hreset = (hcount == 1343); // end of line (reset counter)

// vertical: 806 lines total
// display 768 lines
assign vblankon = hreset & (vcount == 767); // turn on blanking
assign vsyncon = hreset & (vcount == 776); // turn on sync pulse
assign vsyncoff = hreset & (vcount == 782); // turn off sync pulse
assign vreset = hreset & (vcount == 805); // end of frame
```

5. Demo

- **Frame**

```
always @(posedge clk) begin
    if (vblank | (hblank & ~hreset)) rgb <= 0;
    else
        rgb <= (hcount==0 | hcount==639 |
                vcount==0 | vcount==479) ? 7 : 0;
end
```

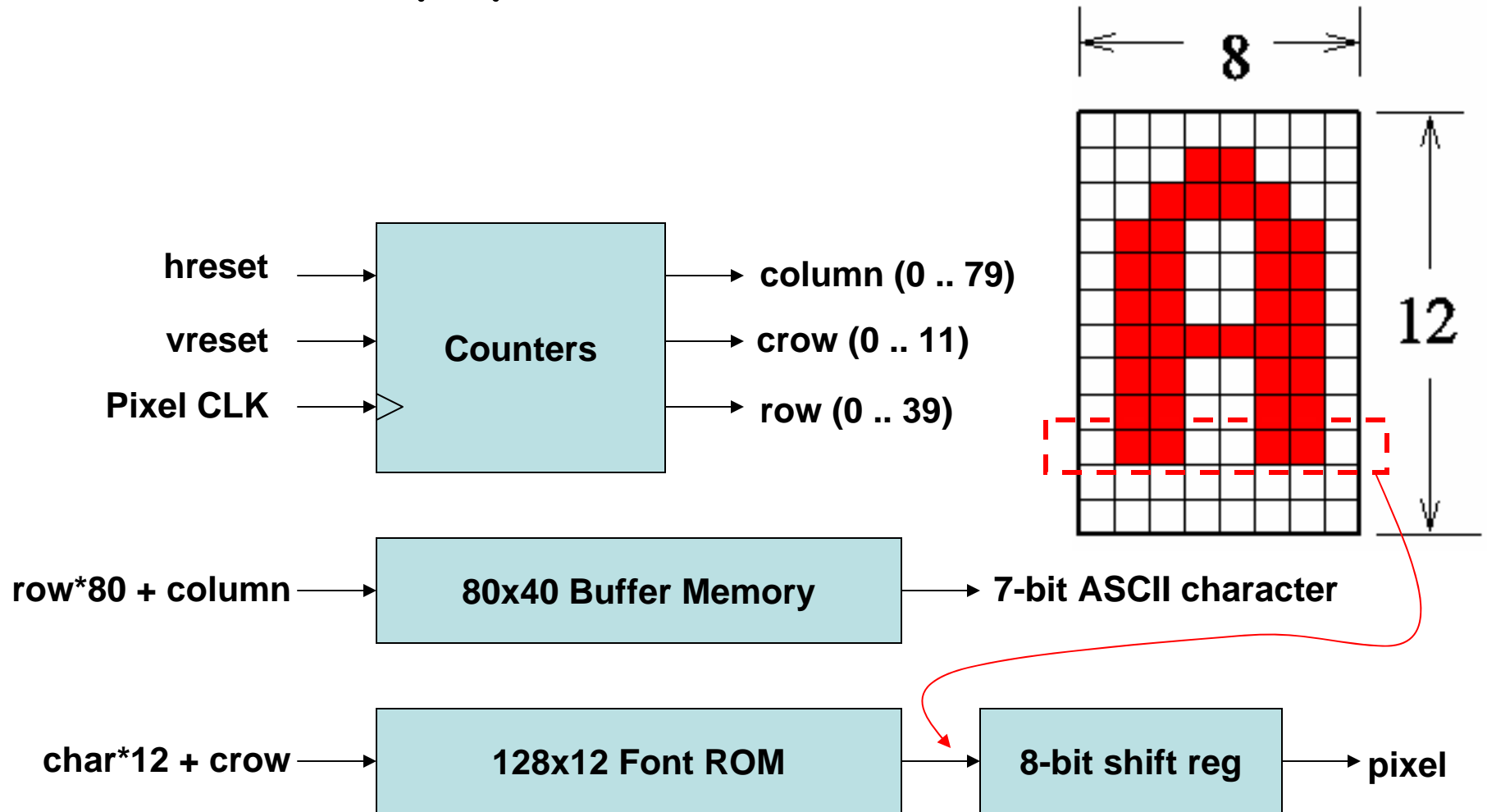
- **Color bars**

```
always @(posedge clk) begin
    if (vblank | (hblank & ~hreset)) rgb <= 0;
    else
        rgb <= hcount[8:6];
end
```

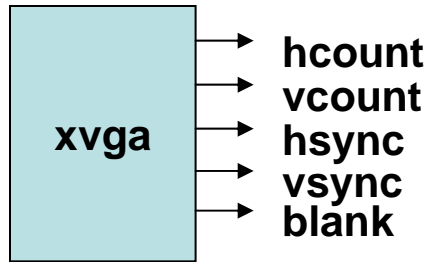
<i>RGB</i>	<i>Color</i>
000	black
001	blue
010	green
011	cyan
100	red
101	magenta
110	yellow
111	white

Demo (cont'd.)

Character Display (80 columns x 40 rows, 8x12 glyph)

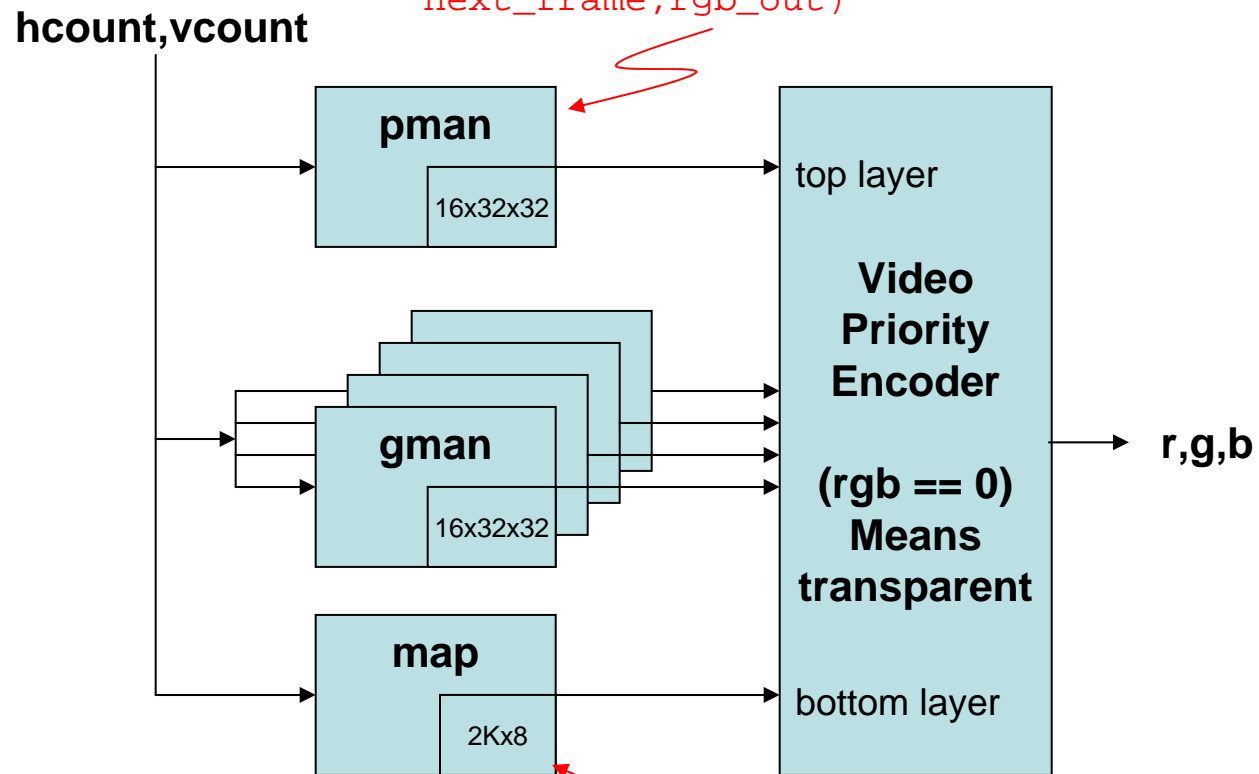


Demo (Pacman: video)



Sprite: rectangular region of pixels, position and color set by game logic. 32x32 pixel mono image from BRAM, up to 16 frames displayed in loop for animation:

```
sprite( clk, reset, hcount, vcount, xpos, ypos, color, next_frame, rgb_out )
```



4 board maps, each 512x8
each map is 16x24 tiles (376 tiles)

Each tile has 8 bits: 4 for move direction (==0 for a wall), pills