



Projecto e Controlo em Lógica Digital

Refs:

Cyclone II device Handbook, Altera corp.
Quartus II Handbook, Altera corp.
DE2 documentation
Verilog HDL, S. Palnitkar, Prentice Hall

- Introdução
 - Fpgas
 - Laboratório
 - Verilog
 - Lógica combinatória
 - Lógica sequencial
- Ferramentas
 - Simulação
 - no QuartusII
 - em Verilog
 - Osciloscópio
 - Analisador lógico
 - Interno
 - Externo

- Máquinas de estados
- Clocks
 - PLL

- Advanced verilog
- Memories
- I/O
- Analog devices
- Audio
- Video
- MicroProcessors

The 3rd lab



The VGA test screen
(It is not a picture!!)

Remember to split the program in blocks!!
(Will be usefull for your next project)

Tools – Mega Wizard

The image shows the Quartus II software interface with the MegaWizard Plug-In Manager dialog box open. The dialog box is titled "MegaWizard Plug-In Manager [page 1]" and contains the following text:

The MegaWizard Plug-In Manager helps you create or modify design files that contain custom variations of megafunctions.

Which action do you want to perform?

- ☒ Create a new custom megafunction variation
- ☐ Edit an existing custom megafunction variation
- ☐ Copy an existing custom megafunction variation

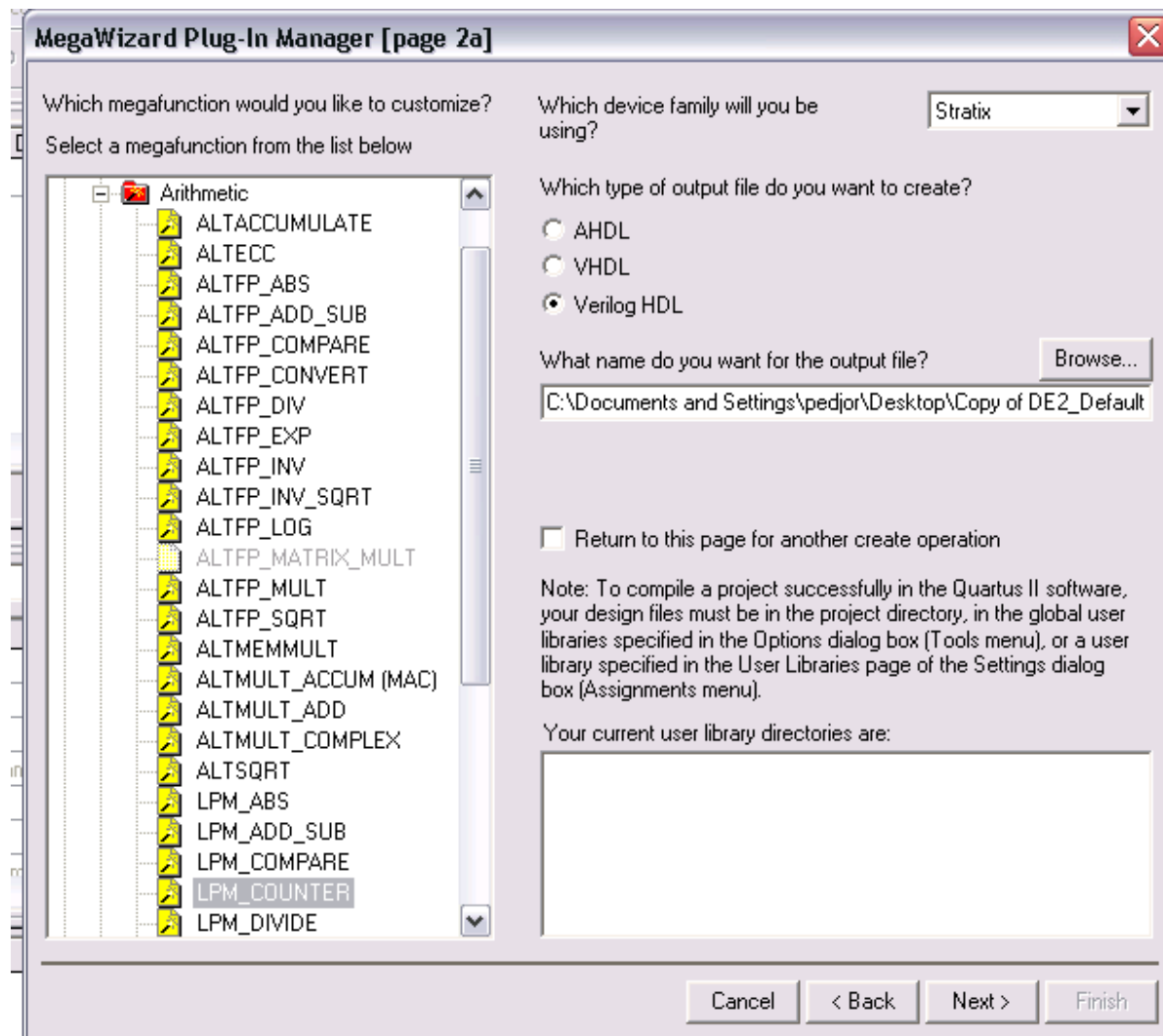
Copyright (C) 1991-2009 Altera Corporation

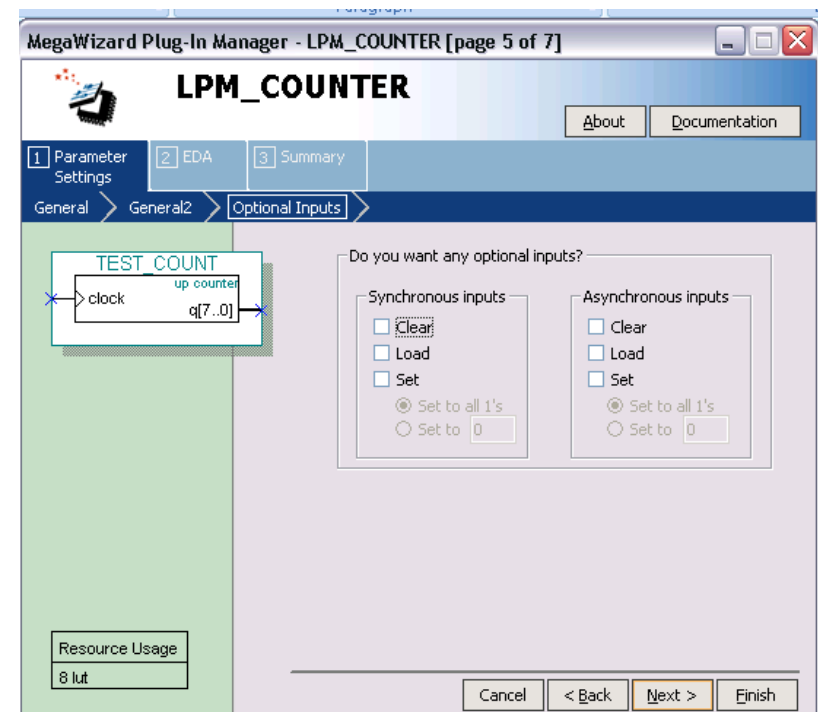
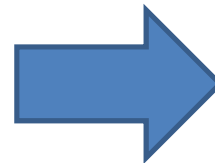
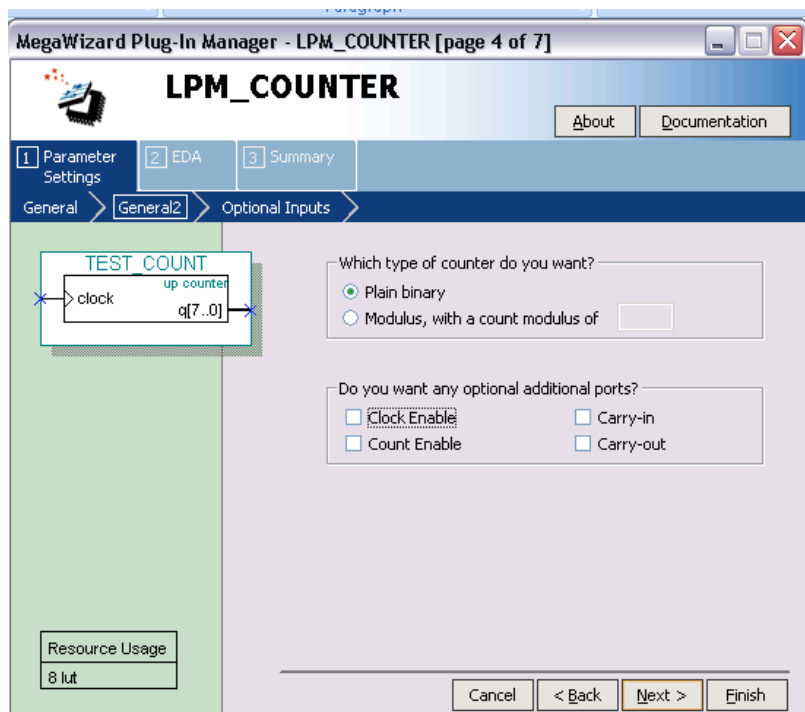
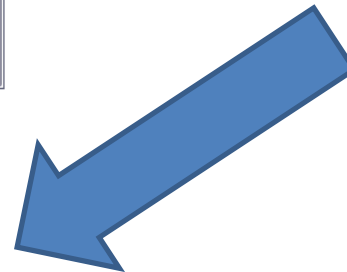
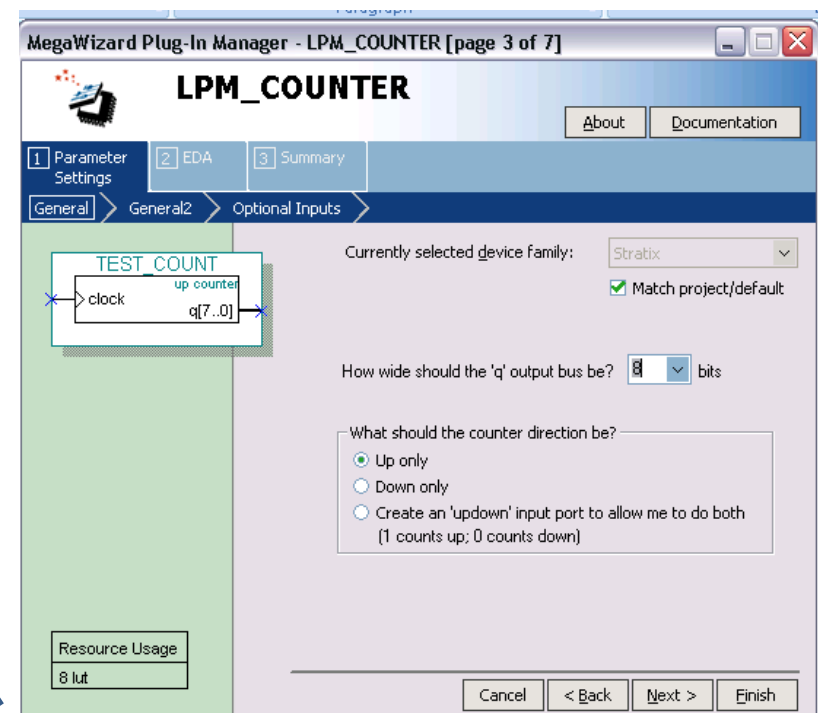
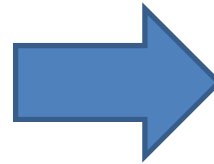
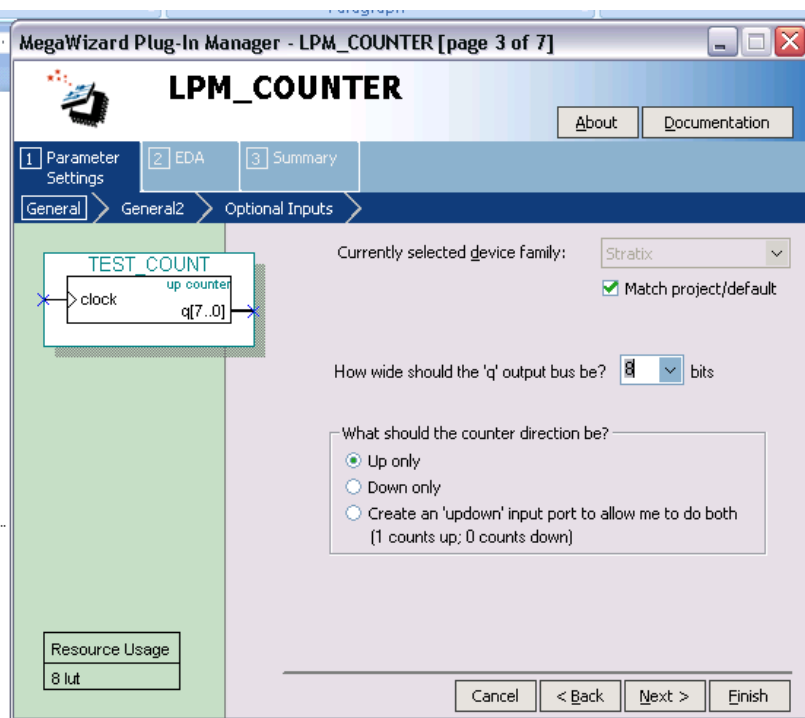
Buttons: Cancel, < Back, Next >, Finish

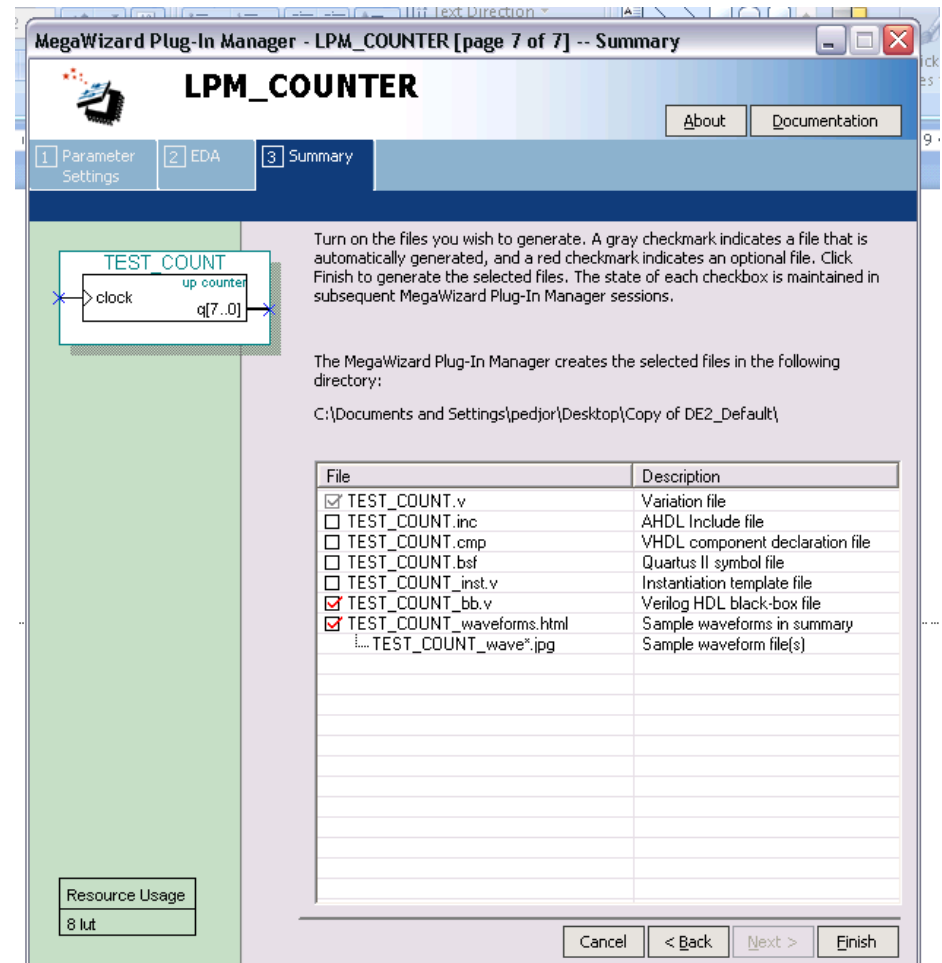
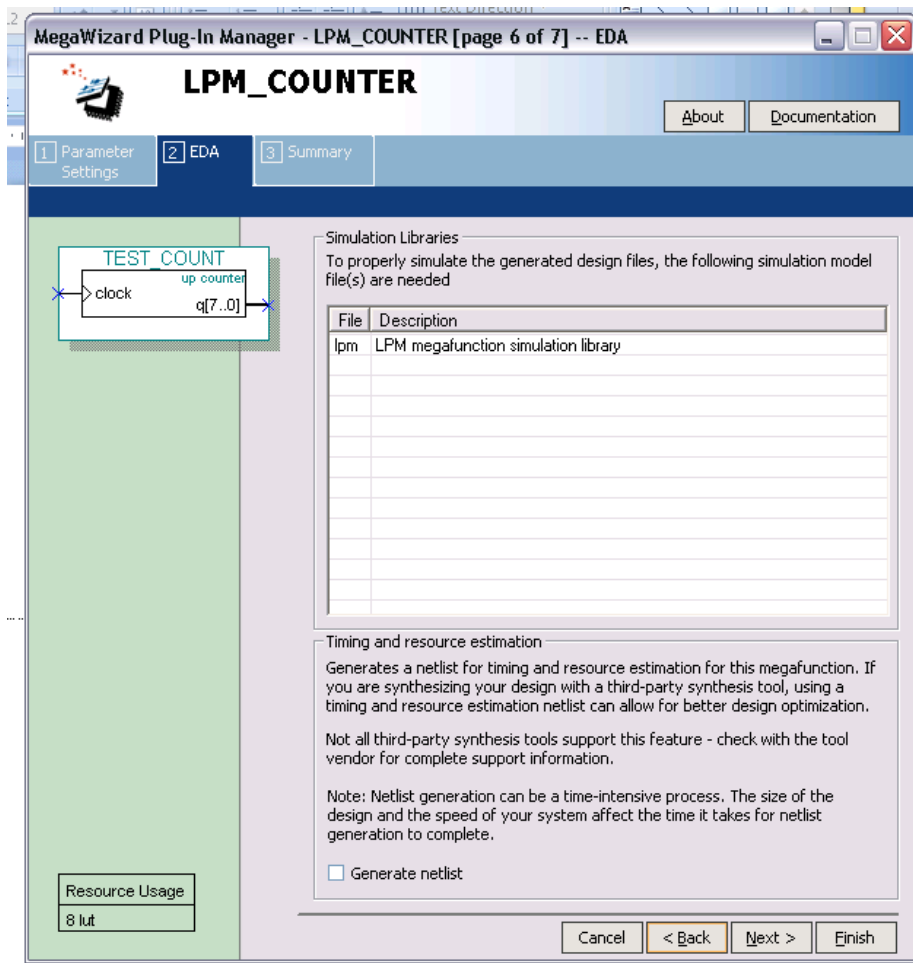
Background buttons: View New Quartus II Information, Documentation

Bottom status bar: Starts the MegaWizard Plug-In Manager

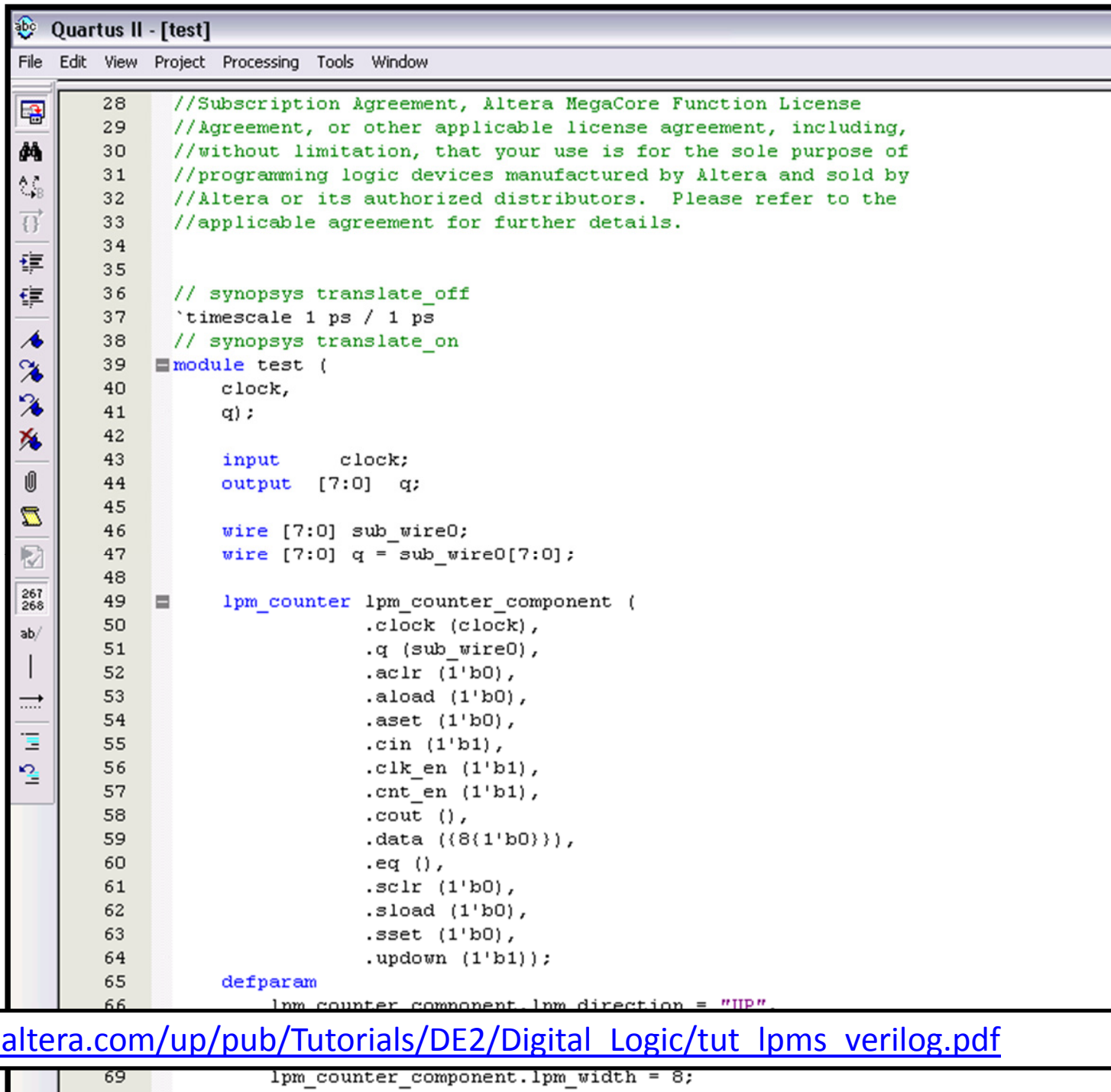
The background shows the Quartus II main window with the Tools menu open, listing various tools including Run EDA Simulation Tool, Run EDA Timing Analysis Tool, Launch EDA Simulation Library Compiler, Launch Design Space Explorer, TimeQuest Timing Analyzer, Advisors, Chip Planner (Floorplan and Chip Editor), Design Partition Planner, Netlist Viewers, SignalTap II Logic Analyzer, In-System Memory Content Editor, Logic Analyzer Interface Editor, In-System Sources and Probes Editor, SignalProbe Pins..., Programmer, MegaWizard Plug-In Manager..., SOPC Builder, Tcl Scripts..., Customize..., Options..., and License Setup... The Project Navigator on the left shows the Compilation Hierarchy, and the Tasks pane shows the Compile Design task selected.







Mega Wizard – What you get



The screenshot shows the Quartus II software interface with a Verilog test module. The window title is "Quartus II - [test]". The menu bar includes File, Edit, View, Project, Processing, Tools, and Window. The left sidebar contains various icons for project management and simulation. The main text area displays the following Verilog code:

```
28 //Subscription Agreement, Altera MegaCore Function License
29 //Agreement, or other applicable license agreement, including,
30 //without limitation, that your use is for the sole purpose of
31 //programming logic devices manufactured by Altera and sold by
32 //Altera or its authorized distributors. Please refer to the
33 //applicable agreement for further details.
34
35
36 // synopsys translate_off
37 `timescale 1 ps / 1 ps
38 // synopsys translate_on
39 module test (
40     clock,
41     q);
42
43     input    clock;
44     output [7:0] q;
45
46     wire [7:0] sub_wire0;
47     wire [7:0] q = sub_wire0[7:0];
48
49     lpm_counter lpm_counter_component (
50         .clock (clock),
51         .q (sub_wire0),
52         .aclr (1'b0),
53         .aload (1'b0),
54         .aset (1'b0),
55         .cin (1'b1),
56         .clk_en (1'b1),
57         .cnt_en (1'b1),
58         .cout (),
59         .data ({8{1'b0}}),
60         .eq (),
61         .sclr (1'b0),
62         .sload (1'b0),
63         .sset (1'b0),
64         .updown (1'b1));
65
66     defparam
67         lpm_counter_component.lpm_direction = "UP";
68
69     lpm_counter_component.lpm_width = 8;
```

ftp://ftp.altera.com/up/pub/Tutorials/DE2/Digital Logic/tut_lpms_verilog.pdf

Memories

Memories in Verilog

- `reg bit; // a single register`
 - `reg [31:0] word; // a 32-bit register`
 - `reg [31:0] array[15:0]; // 16 32-bit regs`
-
- ```
// combinational (asynch) read
assign read_data = array[index];

// clocked (synchronous) write
always @(posedge clock)
 array[index] <= write_data;
```

# Multi-port Memories (aka regfiles)

```
reg [31:0] regfile[30:0]; // 31 32-bit words

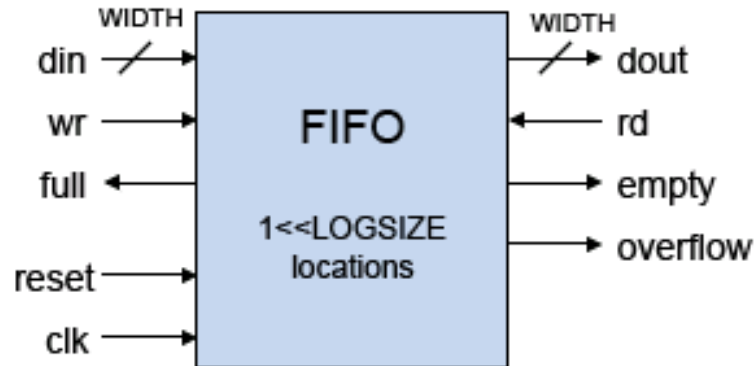
// Beta register file: 2 read ports, 1 write
wire [4:0] ra1,ra2,wa; ← Address
wire [31:0] rd1,rd2,wd; ← Data

assign ra1 = inst[20:16];
assign ra2 = ra2sel ? inst[25:21] : inst[15:11];
assign wa = wasel ? 5'd30 : inst[25:21];

// read ports
assign rd1 = (ra1 == 5'd31) ? 32'd0 : regfile[ra1];
assign rd2 = (ra2 == 5'd31) ? 32'd0 : regfile[ra2];
// write port
always @(posedge clk)
 if (werf) regfile[wa] <= wd;

assign z = ~| rd1; // used in BEQ/BNE instructions
```

# FIFOs

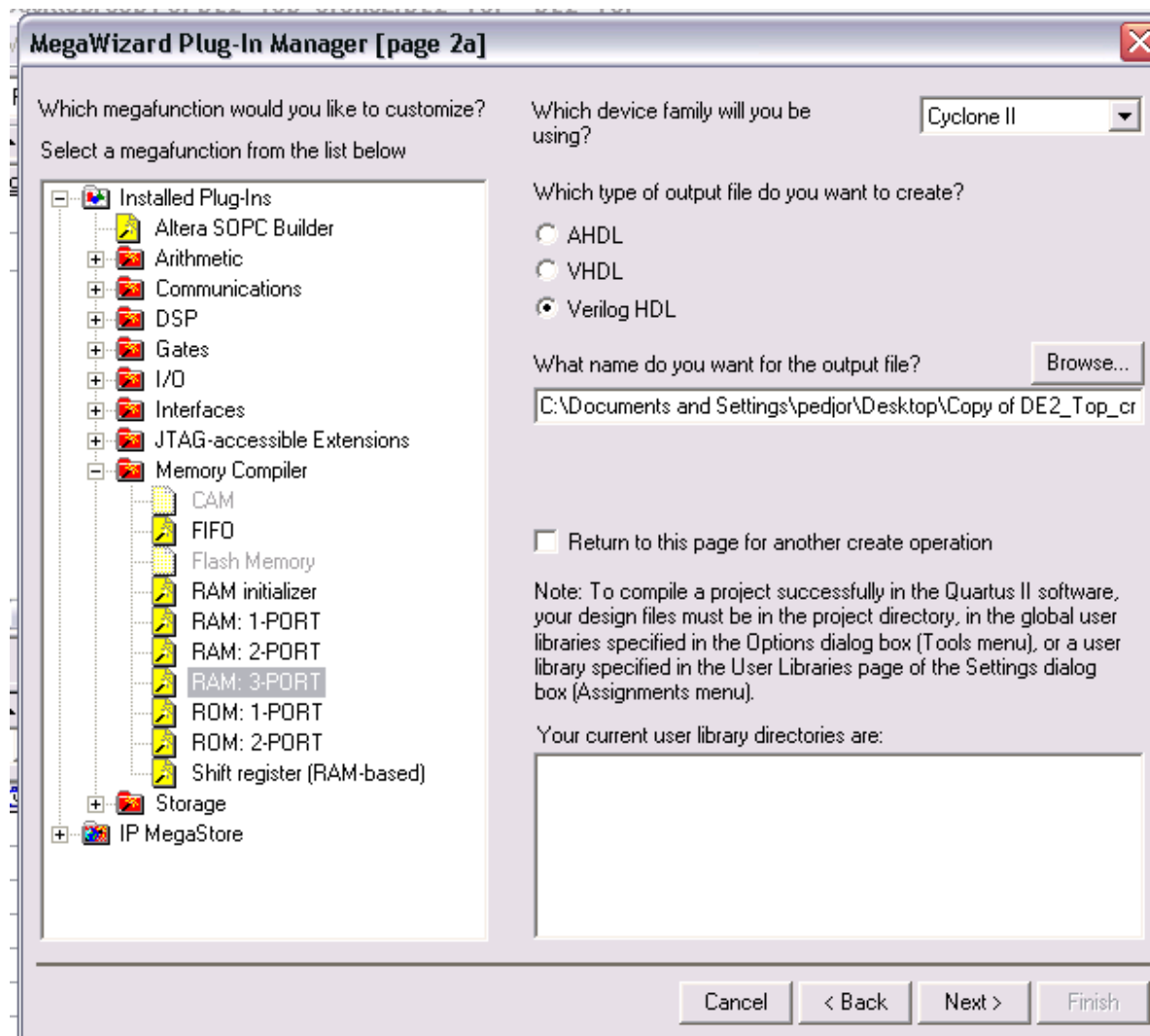


```

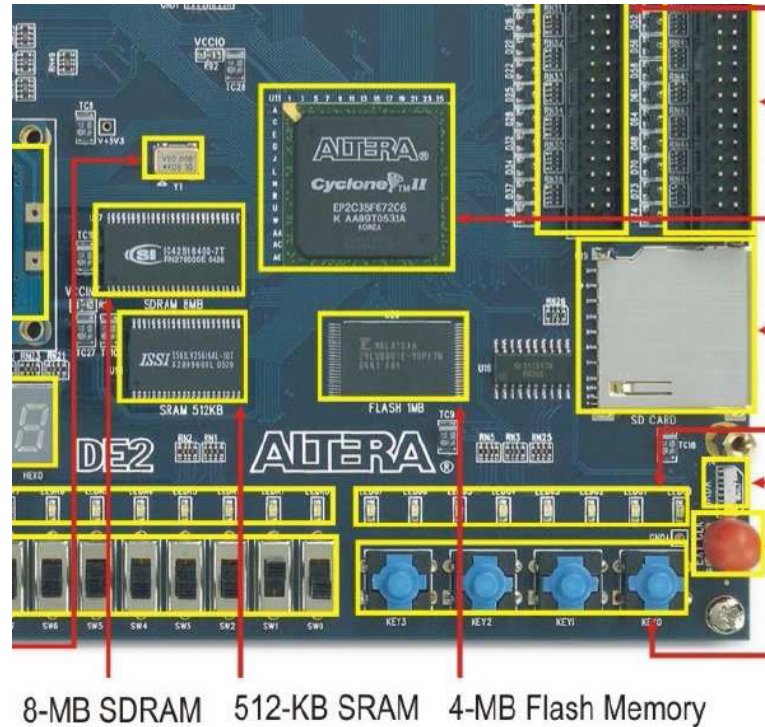
// a simple synchronous FIFO (first-in first-out) buffer
// Parameters:
// LOGSIZE (parameter) FIFO has 1<<LOGSIZE elements
// WIDTH (parameter) each element has WIDTH bits
// Ports:
// clk (input) all actions triggered on rising edge
// reset (input) synchronously empties fifo
// din (input, WIDTH bits) data to be stored
// wr (input) when asserted, store new data
// full (output) asserted when FIFO is full
// dout (output, WIDTH bits) data read from FIFO
// rd (input) when asserted, removes first element
// empty (output) asserted when fifo is empty
// overflow (output) asserted when WR but no room, cleared on next RD
module fifo #(parameter LOGSIZE = 2, // default size is 4 elements
 WIDTH = 4) // default width is 4 bits
 (input clk,reset,wr,rd, input [WIDTH-1:0] din,
 output full,empty,overflow, output [WIDTH-1:0] dout);

```

...



# Memories external to the FPGA



When interfacing external memories you should look at the datasheet!  
Need to understand the exact protocol, addressing, timing, etc.  
Some tools may build you the interface to external memories

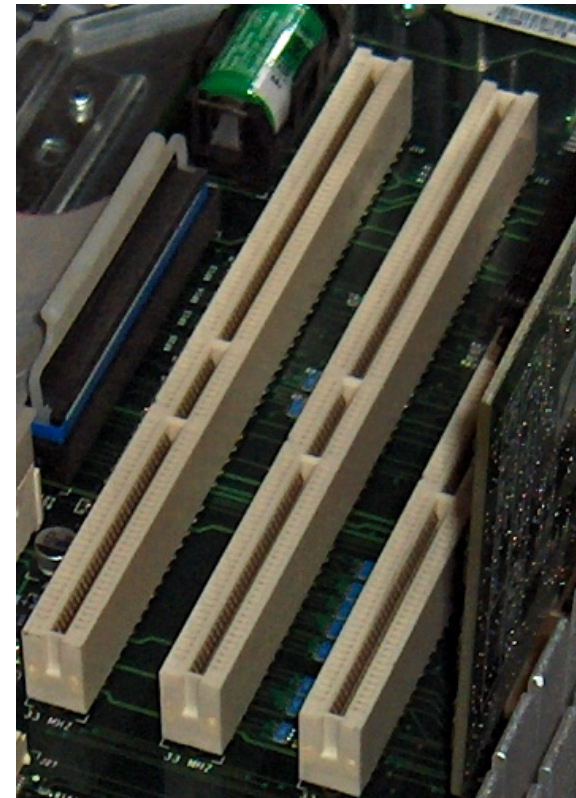
# I/O - parallel

Parallel protocols are typically easy and fast...

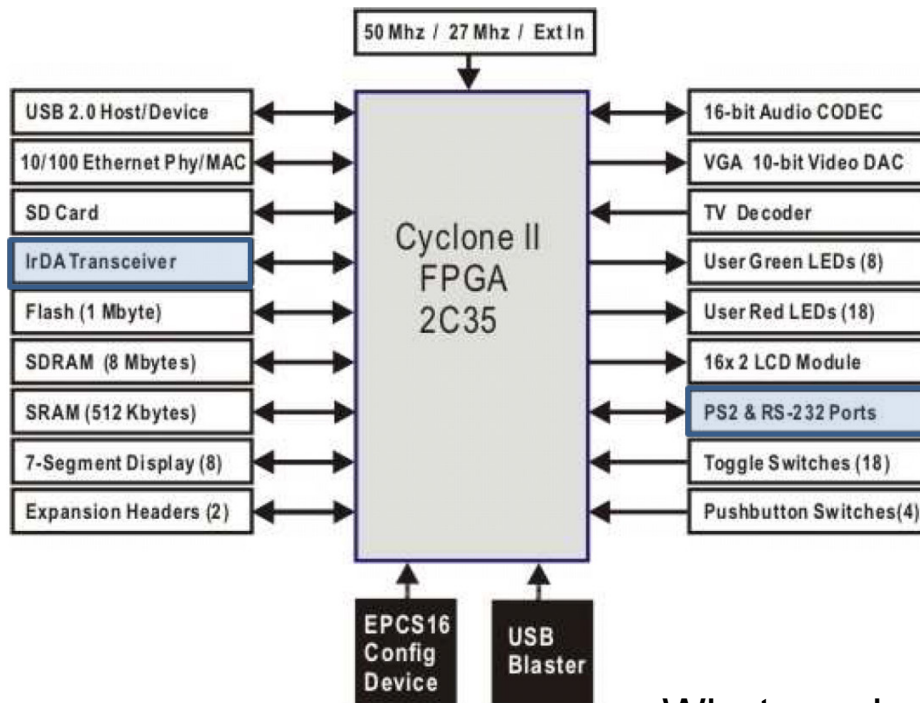
However the resources allocated are big (pins)

Typically the protocol language is very easy.  
Time constraints easy to deal with

e.g.: The PCI protocol



# I/O - serial



Readily available:  
IrDA, RS232, PS2

You can build some other...  
I2C - Inter-Integrated Circuit  
Serial Peripheral Interface (SPI)  
...

What you need to know?

- Pinout
- Clocking requirements
- Protocol (the language)
- The commands of the peripheral you are communicating with
- A zillion standards
  - Asynchronous (no explicit clock) vs. Synchronous (CLK line in addition to DATA line).
  - Recent trend to reduce signaling voltages: save power, reduce transition times
  - Control/low-bandwidth Interfaces: SPI, I<sup>2</sup>C, 1-Wire, PS/2, AC97
  - Networking: RS232, Ethernet, T1, Sonet
  - Computer Peripherals: USB, FireWire, Fiber Channel, Infiniband, SATA, Serial Attached SCSI



# RS232 – The serial port

- Characteristics
  - Large voltages => special interface chips  
(1/mark: -12V to -3V, 0/space: 3V to 12V)
  - Separate xmit and rcv wires: full duplex
  - Slow transmission rates (1 bit time = 1 baud); most interfaces support standardized baud rates: 1200, 2400, 4800, 9600, 19.2K, 38.4K, 57.6K, 115.2K
  - Format
    - Wire is held at 1/mark when idle
    - Start bit (1 bit of "0" at start of transmission)
    - Data bits (LSB first, can be 5 to 8 bits of data)
    - Parity bit (none, even, odd)
    - Stop bits (1, 1.5 or 2 bits of 1/mark at end of symbol)
    - Most common 8-N-1: eight data bits, no parity, one stop bit

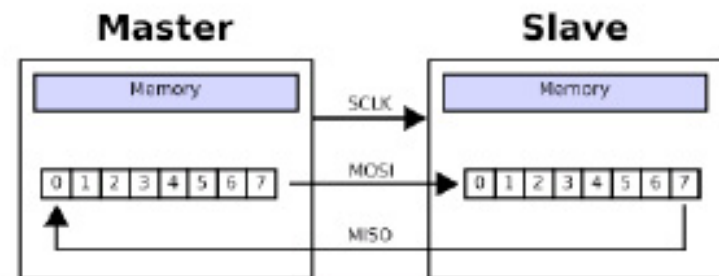
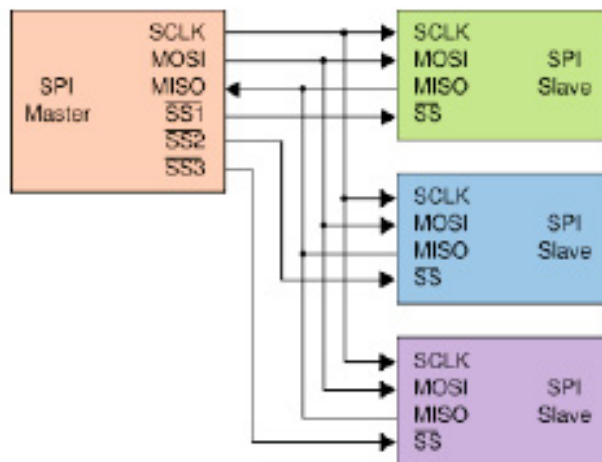
**Sending:** easy - Just compose your signal respecting time constraints

**Receiving:** Need to oversample and understand where the bits are...  
and check the protocol format: start, stop, parity



# SPI (Serial Peripheral Interface)

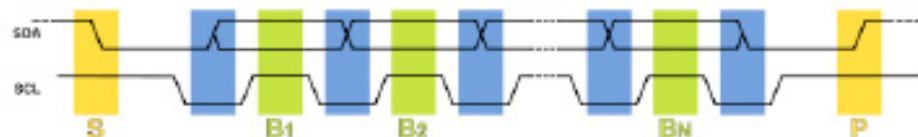
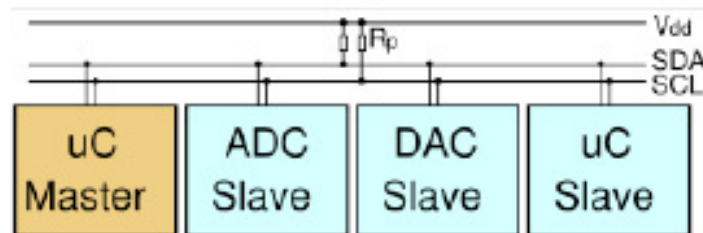
- Simple, 3-wire interface + devices selects
  - SCLK generated by master (1-70MHz). Assert data on one edge, sample data on the other. Default state of SCLK and assignment of edges is often programmable.
  - Master Out Slave In (MOSI) data shifted out of master register into slave register
  - Master In Slave Out (MISO) data shifted out of slave register and into master register
  - Selects (usually active low) determine which device is active. Assertion often triggers an action in the slave, so master waits some predetermined time then shifts data.



Figures from Wikipedia

# I<sup>2</sup>C (Inter-Integrated Circuit)

- 2 open-drain wires (SCL = clock, SDA = data)
- Multiple-master, each transmission addresses a particular device, many devices have many different sub-addresses (internal registers)
- Format (all addresses/data send MSB first):
  - Sender: Start [S] bit (SDA↓ while SCL high)
  - Sender: One or more 8-bit data packets, each followed by 1-bit ACK
    - Data changed when SCL low, sampled at SCL↑
    - Receiver: Active-low ACK generated after each data packet
  - Sender: Stop [P] bit (SDA↑ while SCL high)
- SCL and SDA have pullup resistors, senders only drive low, go high-impedance to let pullups make line high (so multiple drivers okay!)
  - Receiver can hold SCL low to stretch clock timing, sender must wait until SCL goes high before moving to next bit.
  - Multiple senders can contend using SDA for arbitration

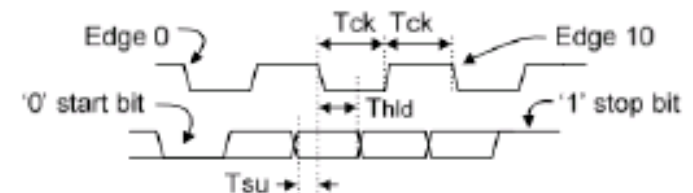


# PS/2 Keyboard/Mouse Interface

- 2-wire interface (CLK, DATA), bidirectional transmission of serial data at 10-16kHz

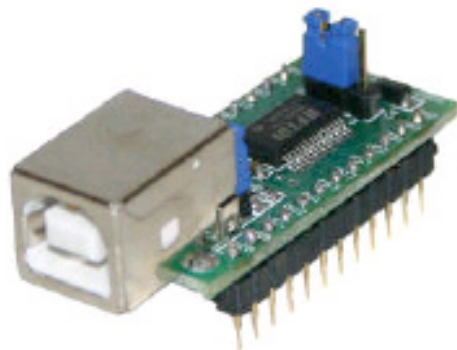
- Format

- Device generates CLK, but host can request-to-send by holding CLK low for 100us
- DATA and CLK idle at "1", CLK starts when there's a transmission. DATA changes on CLK↑, sampled on CLK↓
- 11-bit packets: one start bit of "0", 8 data bits (LSB first), odd parity bit, one stop bit of "1".
- Keyboards send scan codes (not ASCII!) for each press, 8'hF0 followed by scan code for each release
- Mice send button status,  $\Delta x$  and  $\Delta y$  of movement since last transmission

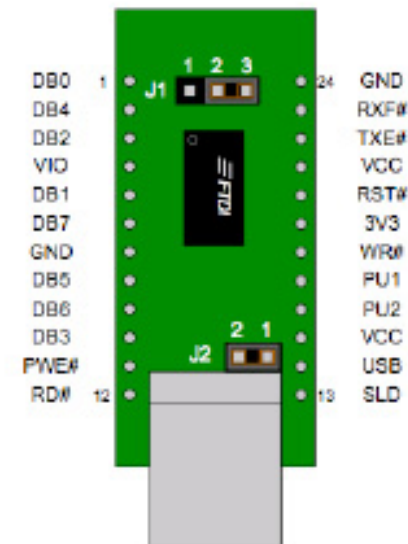


# USB (Universal Serial Bus)

- 2-wire (D+,D-) for high-speed, bidirectional polled transmission between master and addressable endpoints in multiple devices. Full speed (12Mbps) and High speed (480Mbps) data rates.
- Multi-level tiered-star topology (127 devices, including hubs)
- FTDI UM245R USB-to-FIFO module for bidirectional data transfer using a handshake protocol, also asynchronous "bit-bang" mode with selectable baud rates.
  - 24-pin DIP module, wire to user pins
  - Drivers for Windows workstations in lab



Figures from ftdi.com



# I/O - Analog



# I/O – Digital ↔ Analog



FPGAs are digital devices!  
They can't generate or receive analog signals.

You need to use a DAC or ADC to interface analog devices!

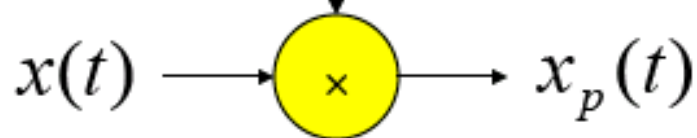


# Discrete Time

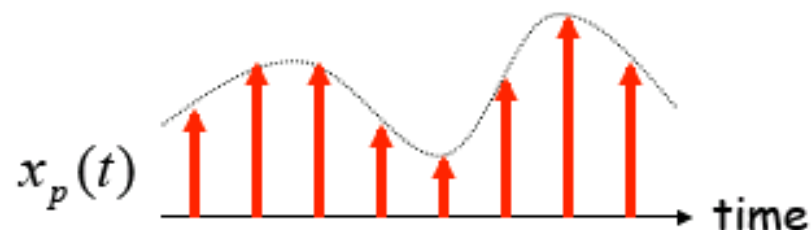
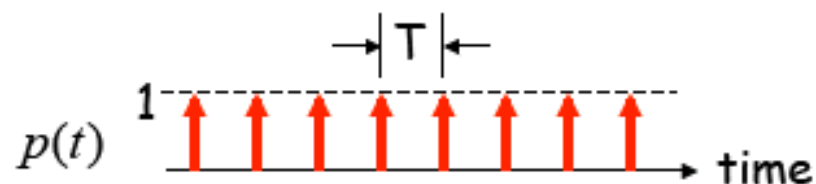
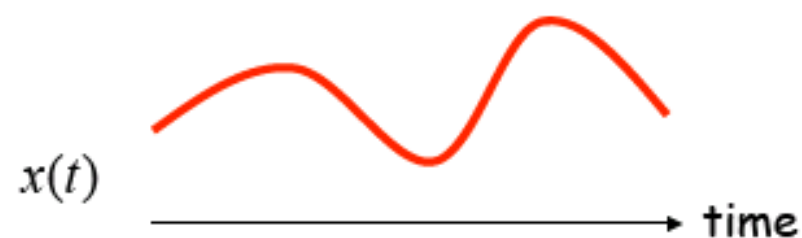
Let's use an **impulse train** to sample a continuous-time function at a regular interval  $T$ :

$\delta(x)$  is a narrow impulse at  $x=0$ ,  
where  $\int_{-\infty}^{\infty} f(t)\delta(t-a)dt = f(a)$

$$p(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT)$$

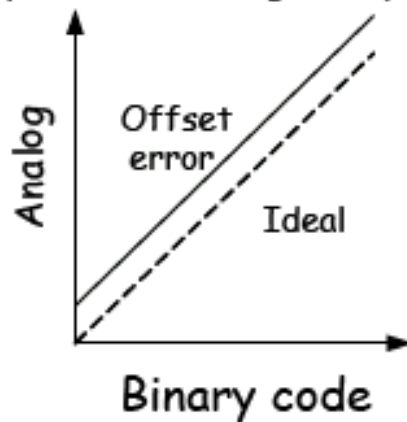


Time Domain

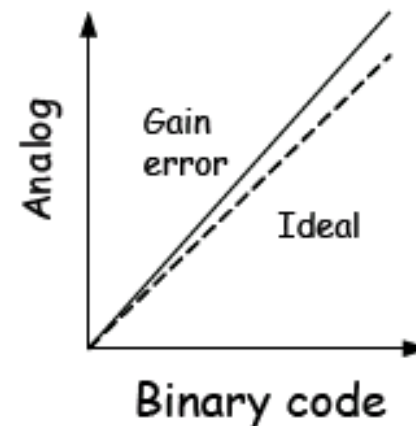


# Non-idealities in Data Conversion

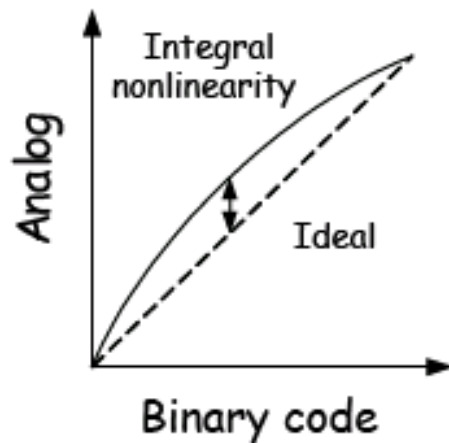
**Offset** - a constant voltage offset that appears at the output when the digital input is 0



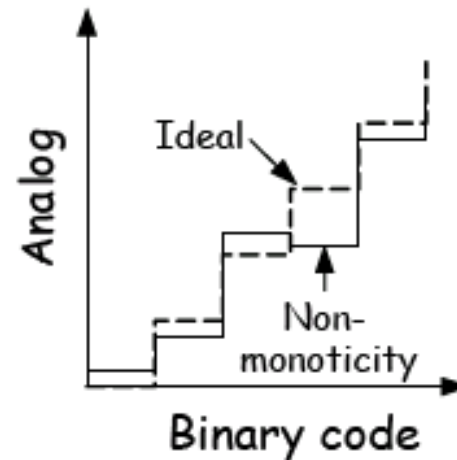
**Gain error** - deviation of slope from ideal value of 1



**Integral Nonlinearity** - maximum deviation from the ideal analog output voltage



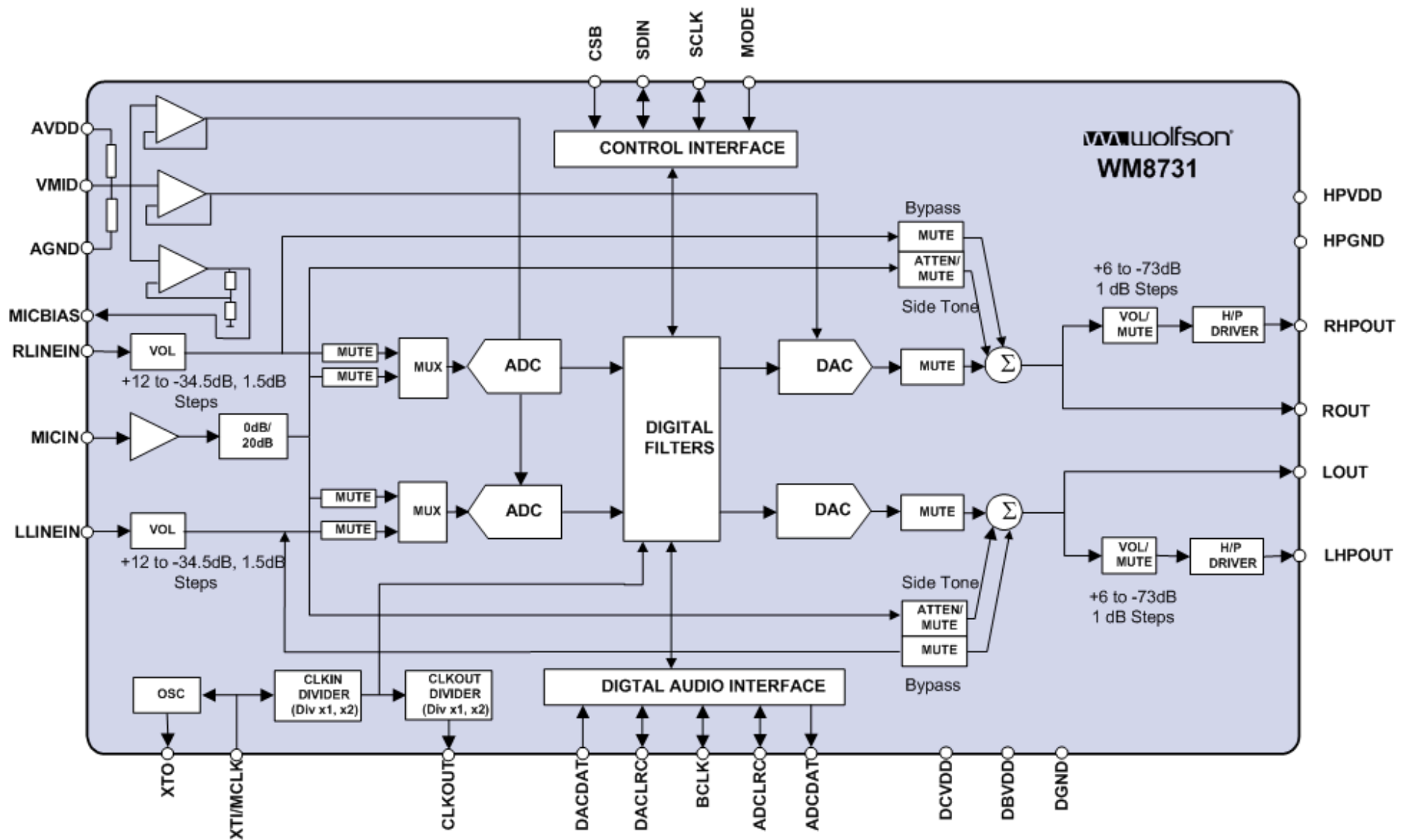
**Differential nonlinearity** - the largest increment in analog output for a 1-bit change





# Sound (analog signal)

Means ADC/DAC



**BREAK**

# Applications of Flat-Panel Displays

## SMALL FORMAT



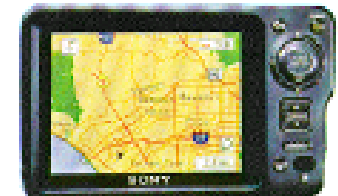
Medical Defibrillator



MP3 Player



Personal Digital Assistant

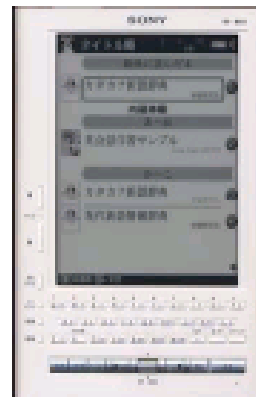


Car Navigation & Entertainment

## LARGE FORMAT



Desktop Monitor (color)



Electronic Book



Large Screen Television (color)

Courtesy of PixTech

# Some Display Terminologies

| <b>Term</b>                 | <b>Definition</b>                                                                   |
|-----------------------------|-------------------------------------------------------------------------------------|
| <b>Pixel</b>                | Picture element—The smallest unit that can be addressed to give color and intensity |
| <b>Pixel Matrix</b>         | Number of Rows by the Number of Columns of pixels that make up the display          |
| <b>Aspect Ratio</b>         | Ratio of display width to display height; for example 4:3, 16:9                     |
| <b>Resolution</b><br>(ppi)  | Number of pixels per unit length (ppi=pixels per inch)                              |
| <b>Frame Rate</b><br>(Hz)   | Number of Frames displayed per second                                               |
| <b>Viewing Angle</b><br>(°) | Angular range over which images from the display could be viewed without distortion |
| <b>Diagonal Size</b>        | Length of display diagonal                                                          |
| <b>Contrast Ratio</b>       | Ratio of the highest luminance (brightest) to the lowest luminance (darkest)        |

# Information Capacity of Displays

## (Pixel Count)

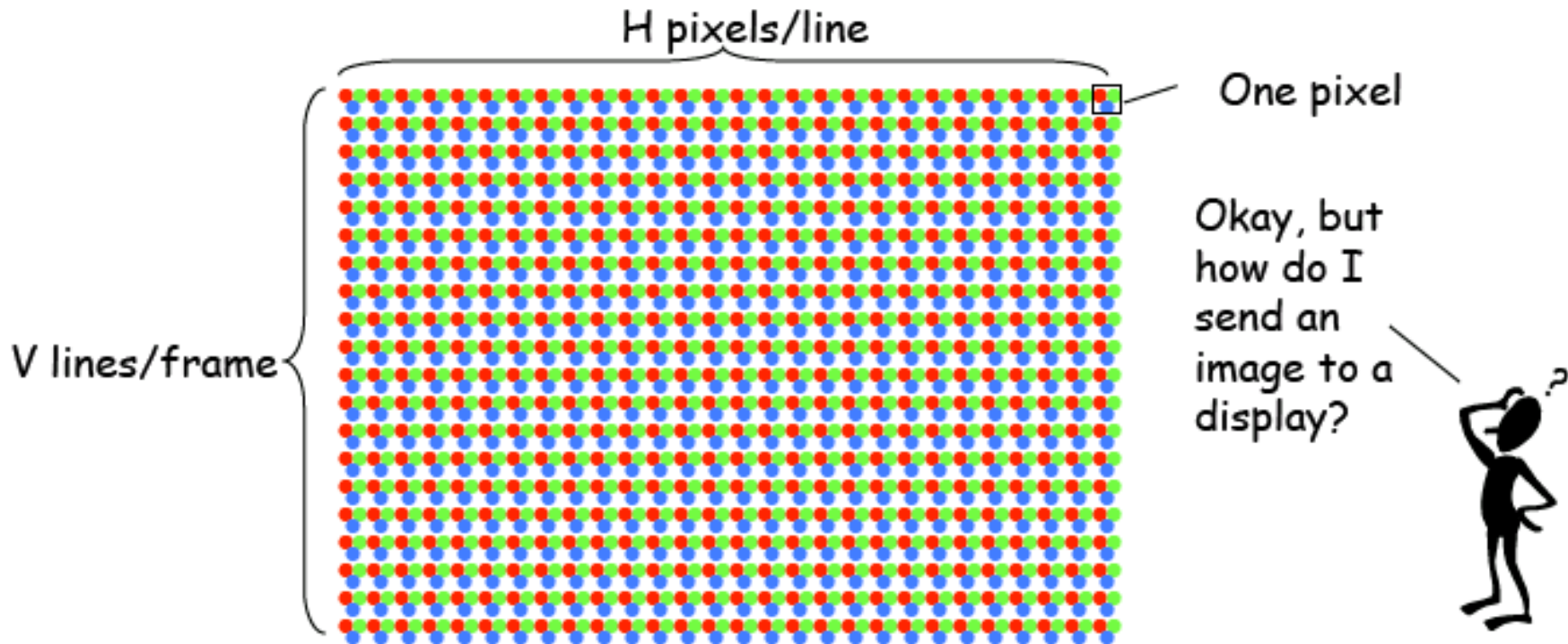
| Resolution                                             | Pixel               | Ratio |
|--------------------------------------------------------|---------------------|-------|
| Video Graphic Array<br>( <b>VGA</b> )                  | 640 x 480 x RGB     | 4:3   |
| Super Vedio Graphic Array<br>( <b>SVGA</b> )           | 800 x 600 x RGB     | 4:3   |
| eXtended Graphic Array<br>( <b>XGA</b> )               | 1,024 x 768 x RGB   | 4:3   |
| Super eXtended Graphic Array<br>( <b>SXGA</b> )        | 1,280 x 1,024 RGB   | 5:4   |
| Super eXtended Graphic Array plus<br>( <b>SXGA+</b> )  | 1,400 x 1,080 x RGB | 4:3   |
| Ultra eXtended Graphic Array<br>( <b>UXGA</b> )        | 1,600 x 1,200 x RGB | 4:3   |
| Quad eXtended Graphics Array<br>( <b>QXGA</b> )        | 2048 x 1536 x RGB   | 4:3   |
| Quad Super eXtended Graphics Array<br>( <b>QSXGA</b> ) | 2560 x 2048 x RGB   | 4:3   |

# Video(analog signals)

Means DAC

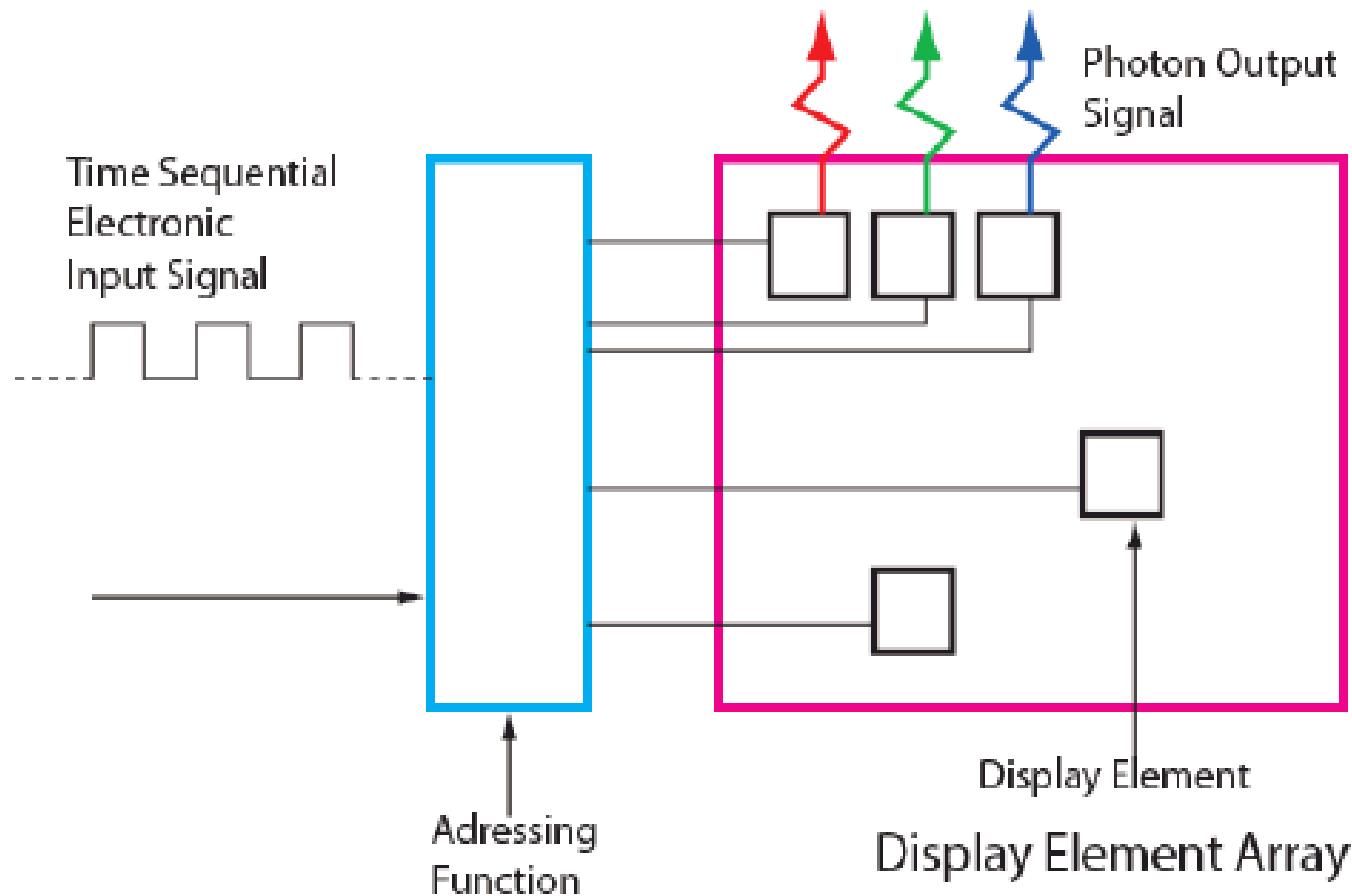
## The CRT: Generalized Video Display

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, ...)

# How Do Displays Work?



Pankove

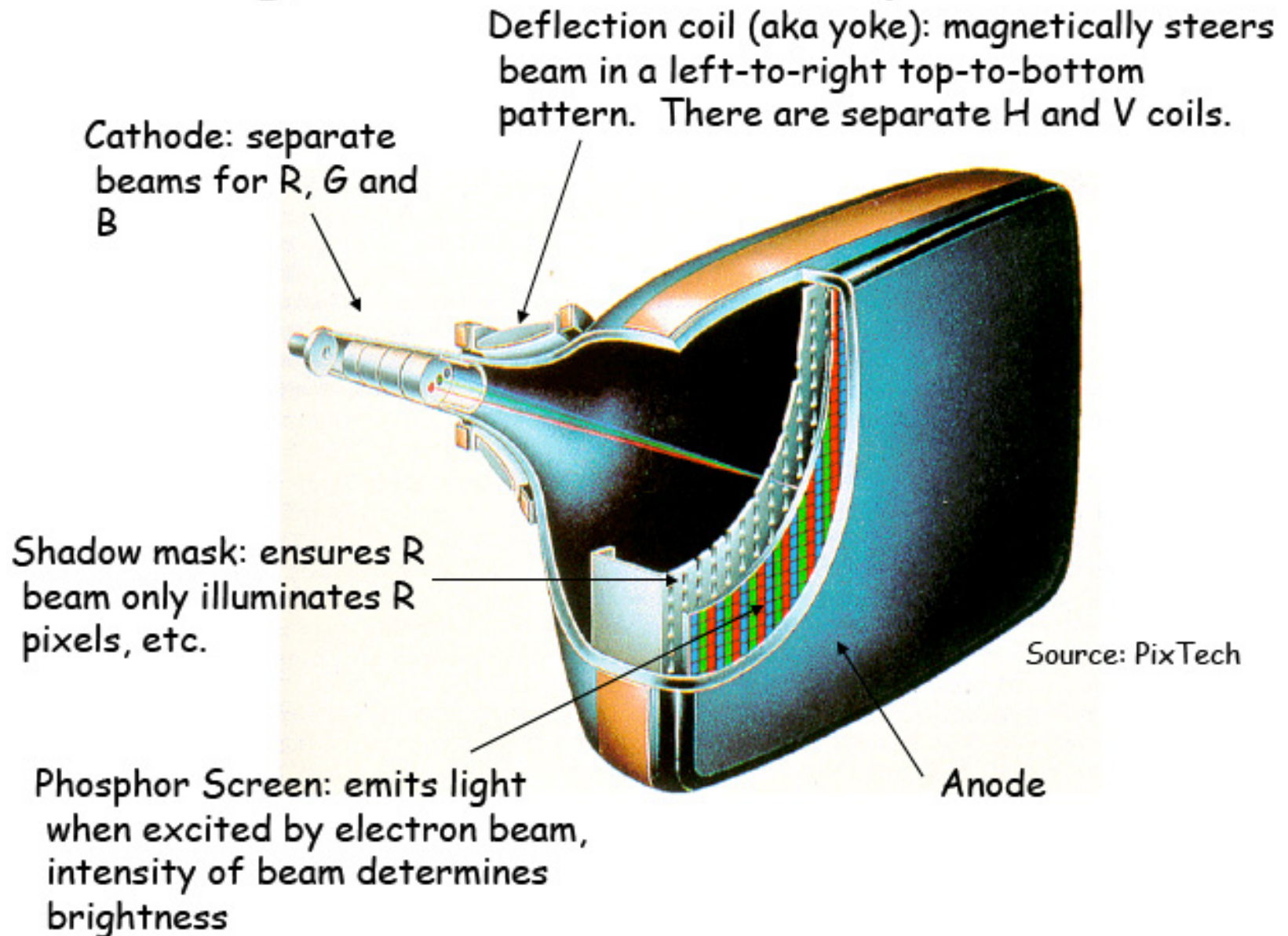
- “**Time Sequential Electrical Signals**” converted into **images**.
  - Signals routed to the display elements (**similar to memory addressing**)
  - Pixels convert the electrical signal into light of color and intensity (**inverse of image capture**)

# Classifications of Displays by Technology

- Displays could be classified into two broad categories
  - Light Generation (**Emissive Displays**)
  - Light Modulation (**Light Valve Displays**)
- **Emissive Displays** generate photons from electrical excitation of the picture element (pixels)
  - Cathode Ray Tubes (CRTs), Organic Light Emitting Displays (OLEDs), Plasma Displays (PDs)
- **Light Valve Displays** spatially and temporally modulate the intensity pattern of the picture elements (pixels)
  - Liquid Crystal Displays (LCDs), Digital Light Processors (DLPs), Electrophoretic Displays (EPDs)

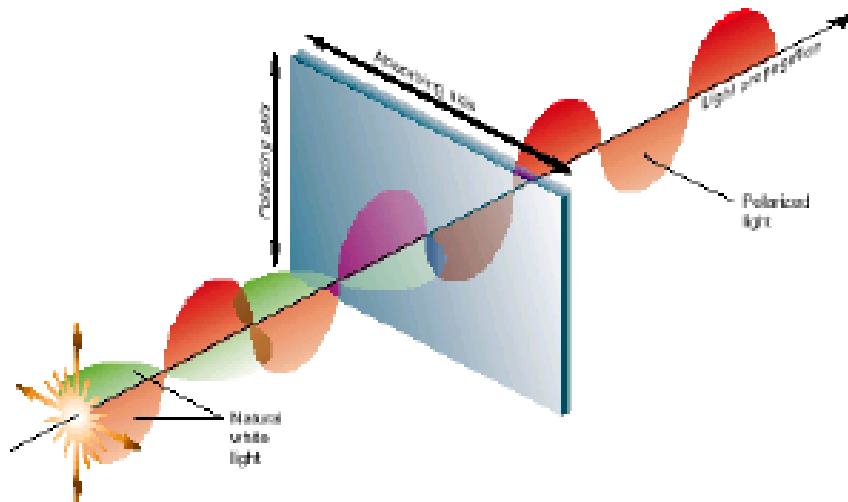
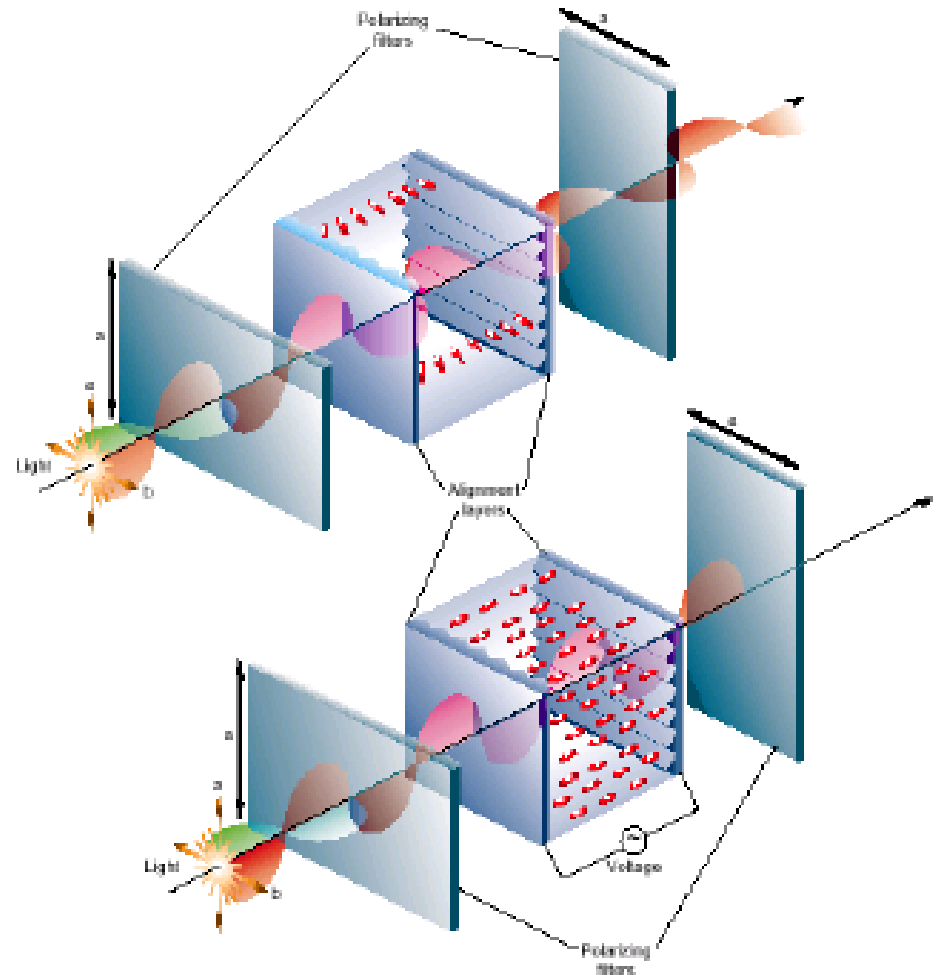
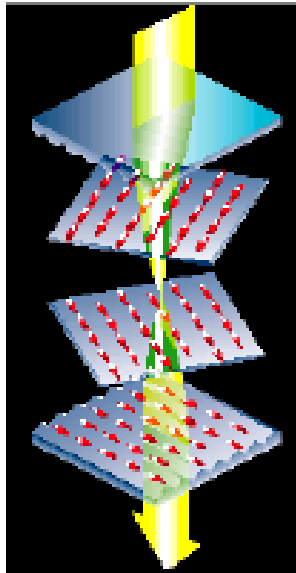


# Background: Cathode Ray Tubes



# Liquid Crystal Displays

Liquid Crystals rotate the plane of polarization of light when a voltage is applied across the cell

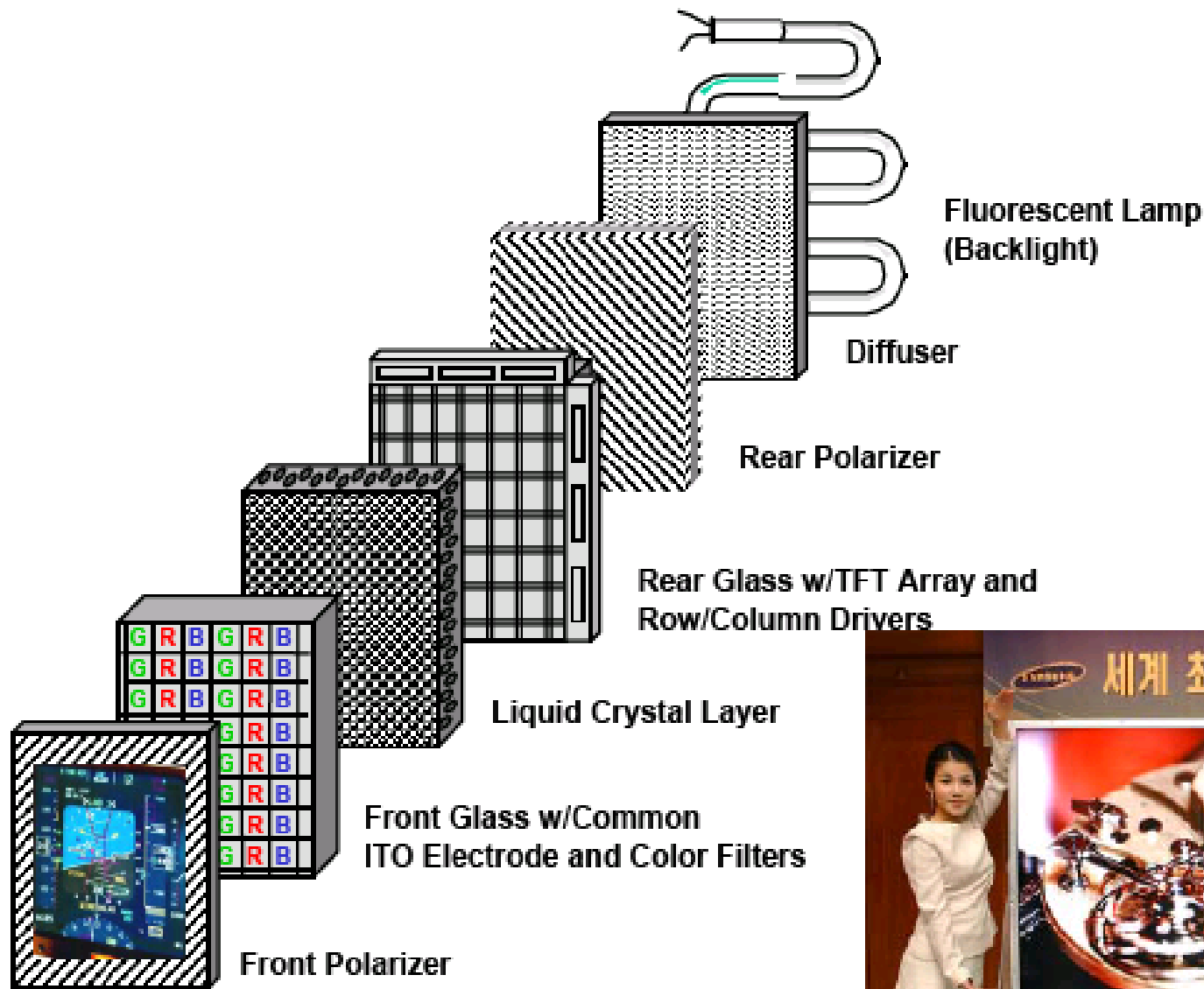


**Polarization Rotator**

Courtesy of Silicon Graphics

# TFT AMLCD

Active Matrix LCD



## 82" TFT AMLCD



K. Sarma

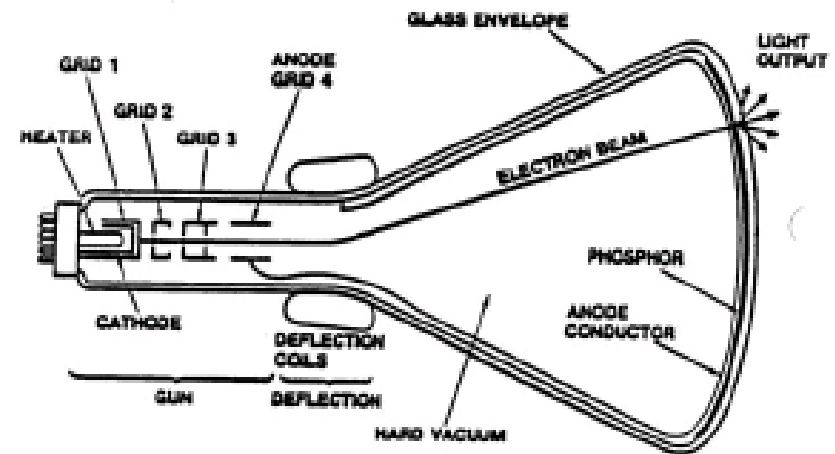
SID 05

# Standard Display Addressing Modes

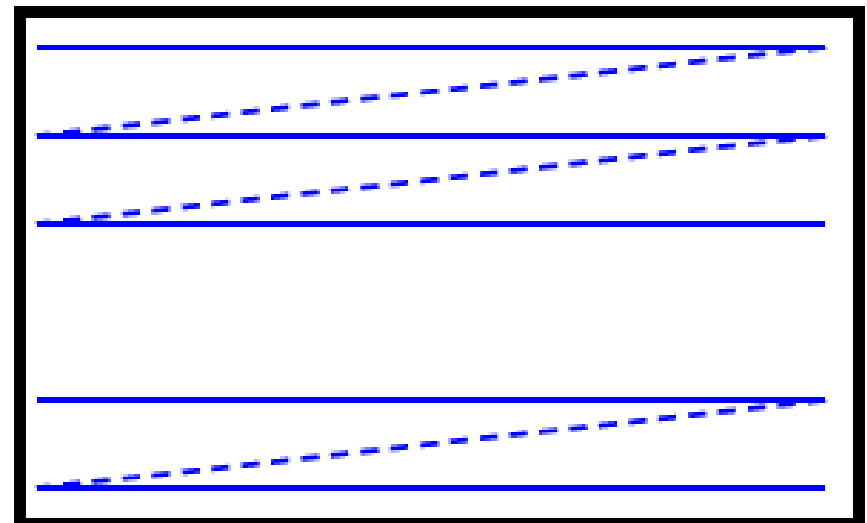
- Sequential Addressing (pixel at a time)
  - CRT, Laser Projection Display
- Matrix Addressing (line at a time)
  - Row scanning, PM LCD, AMLCD, FED, PDPs, OLEDs
- Direct Addressing
  - 7-segment LCD
- Random Addressing
  - Stroke-mode CRT

# Sequential Addressing (Raster Scan)

- Time is multiplexed
  - Signal exists in a time cell
- A pixel is displayed at a time
  - Single data line
- Rigid time sequence and relative spatial location of signal
  - Raster scan
- Data rate scales with number of pixels
- Duty cycle scales with number of pixels
- Horizontal sync coordinates lines
- Vertical sync coordinates frames
- Blanking signals (vertical & horizontal) so that retraces are invisible



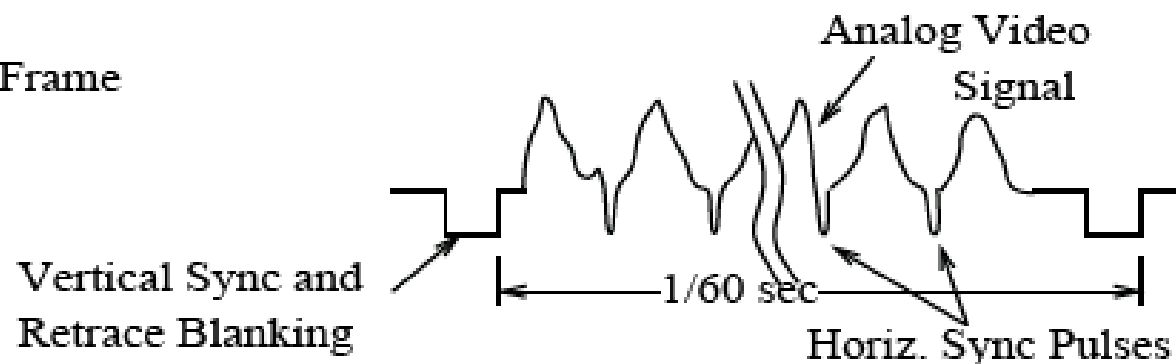
————— Scan Lines  
----- Retrace Lines



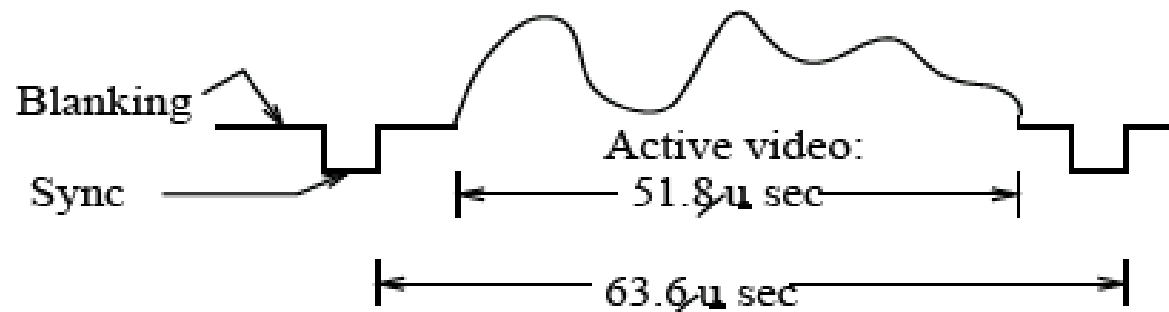
# Composite Frames

- The 'frame' is a single picture (snapshot).
  - It is made up of many lines.
  - Each frame has a synchronizing pulse (vertical sync).
  - Each line has a synchronizing pulse (horizontal sync).
  - Brightness is represented by a positive voltage.
  - Horizontal and Vertical intervals both have blanking so that retraces are not seen (invisible).

Composite Frame

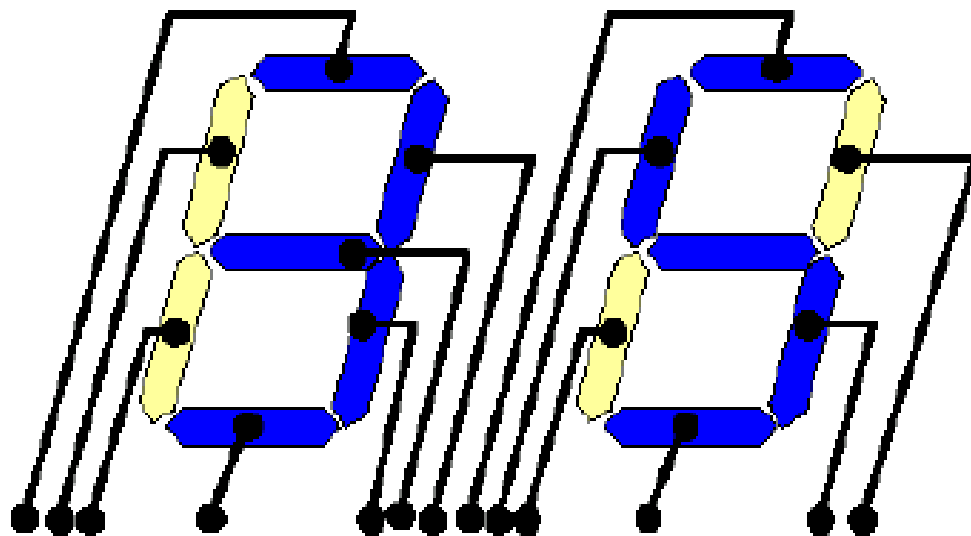


Horizontal Line



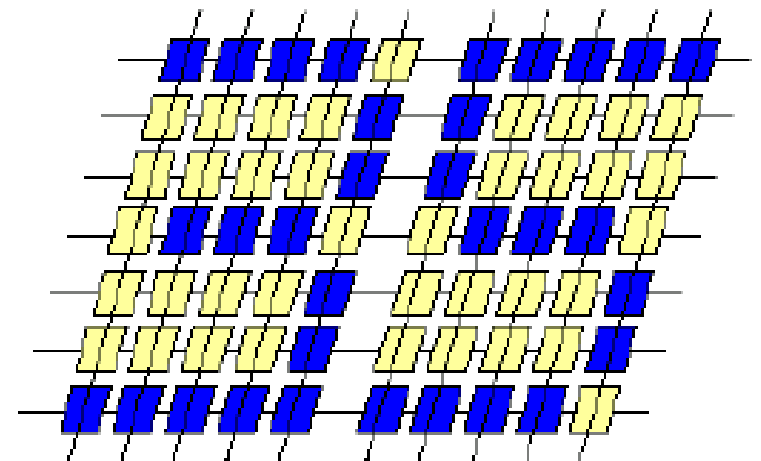
# Direct vs. Matrix Addressing

**Direct Driving**



**Segment Display**  
(7-segment)

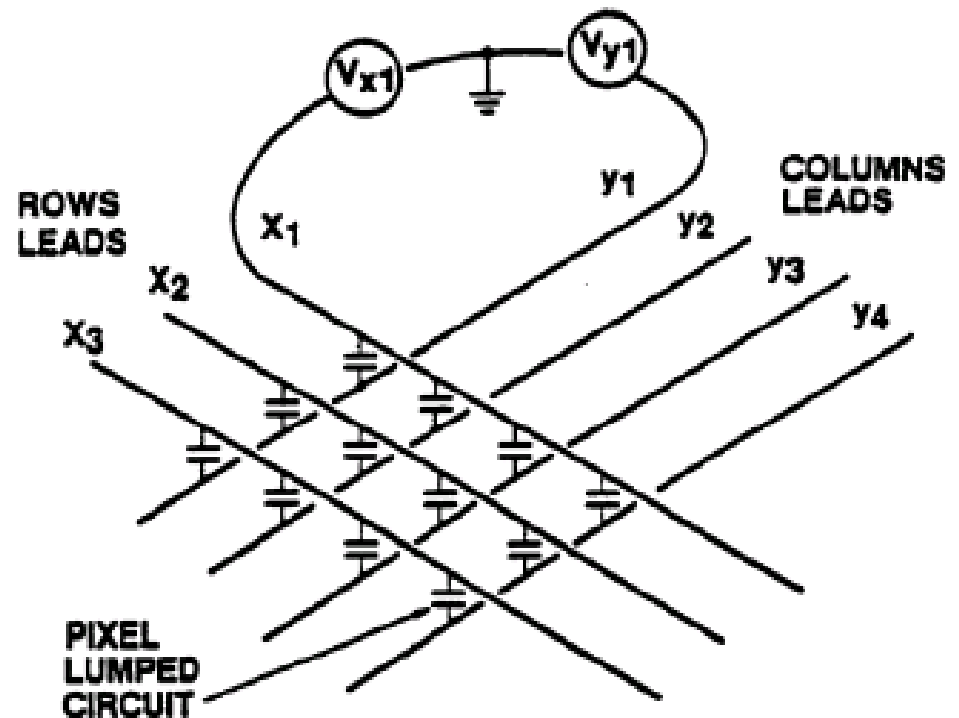
**Multiplex Driving**



**Matrix Display**  
(dot-matrix)

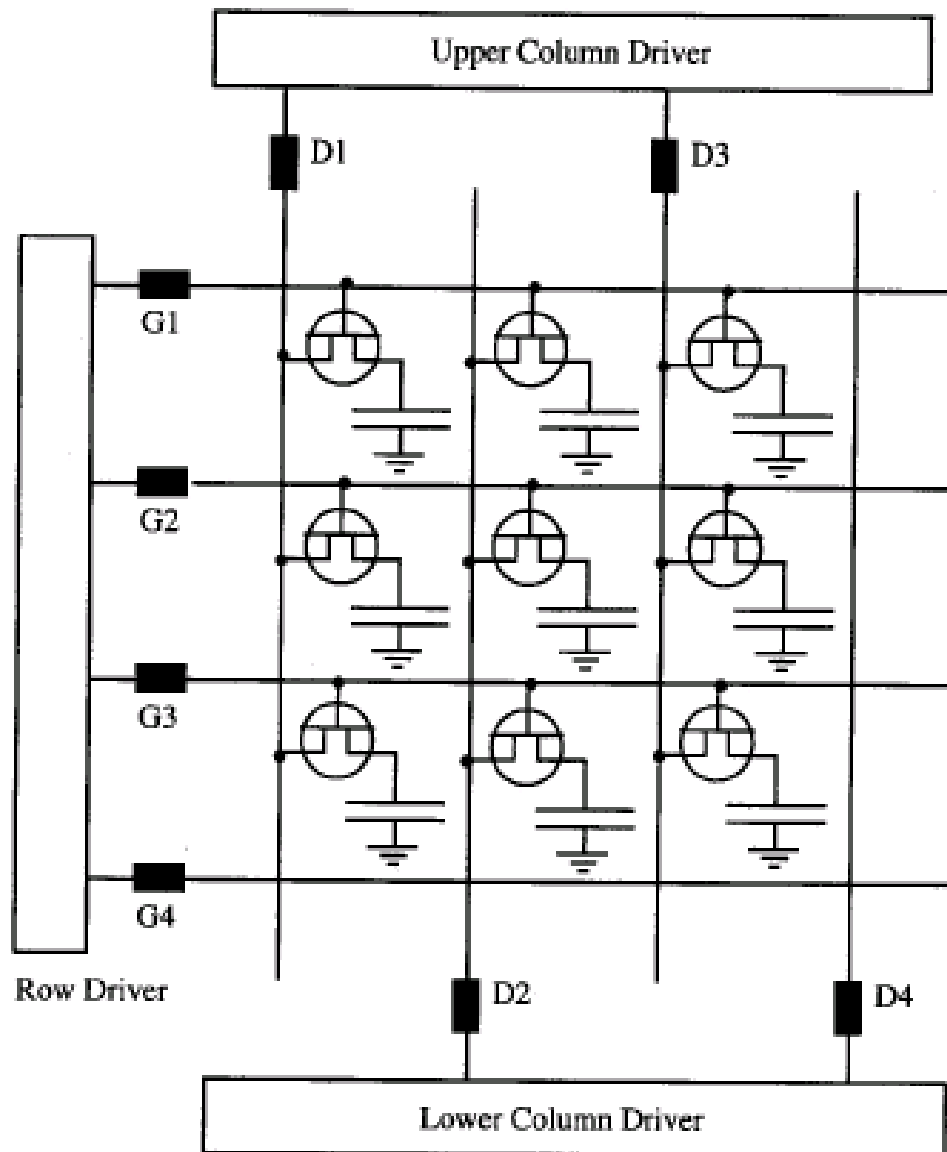
# Matrix Addressing

- Time multiplexed
- Row at a time scanning
  - A column displayed during the time assigned to a row
- For a N rows by M columns display
  - $M + N$  electrodes are required
- Row scanning rate scales with number of rows
- Data rate scales with number of pixels
- Duty cycle scales with number of rows



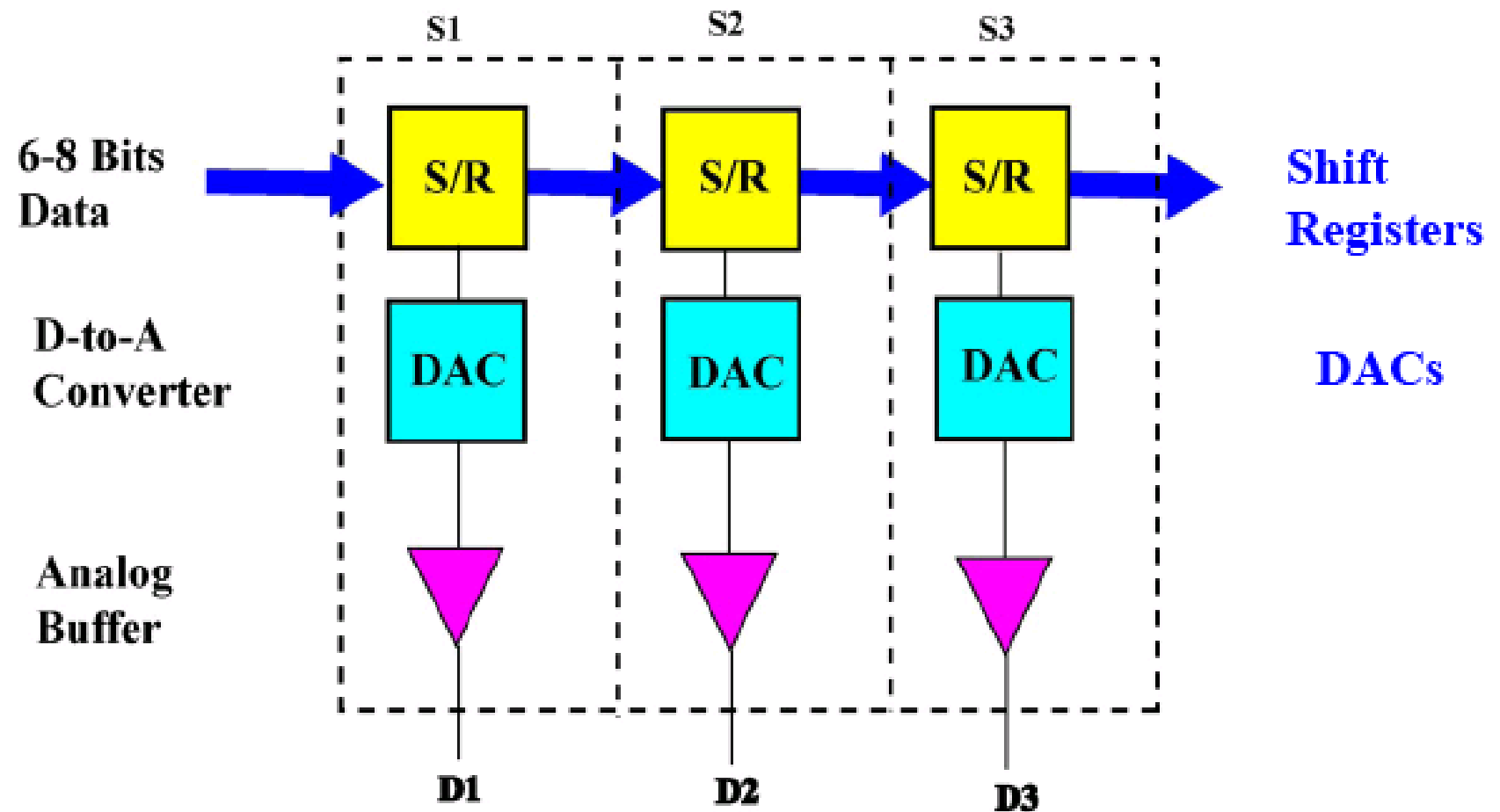


# Active Matrix Addressing



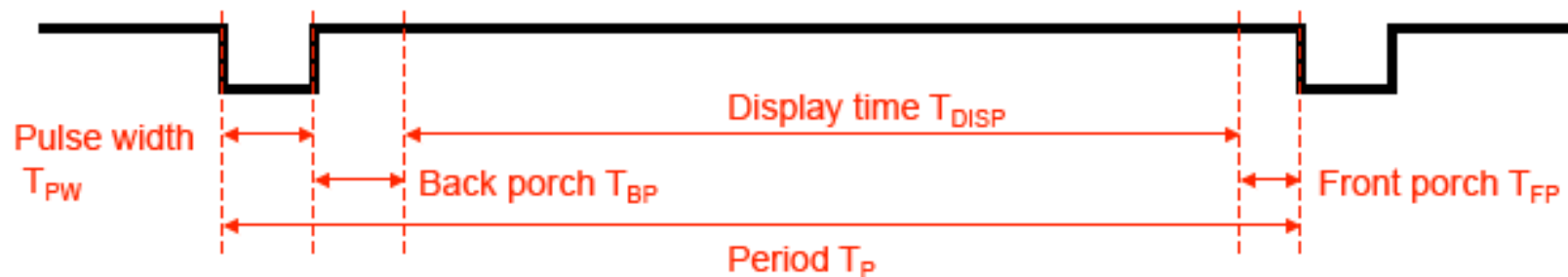
- Introduce non linear device that improves the selection.
- Storage of data values on capacitor so that pixel duty cycle is 100%
- Improve brightness of display by a factor of  $N$  (# of rows) over passive matrix drive
- Display element could be LC, EL, OLED, FED etc

# Digital Data Drivers



# Sync Signal Timing

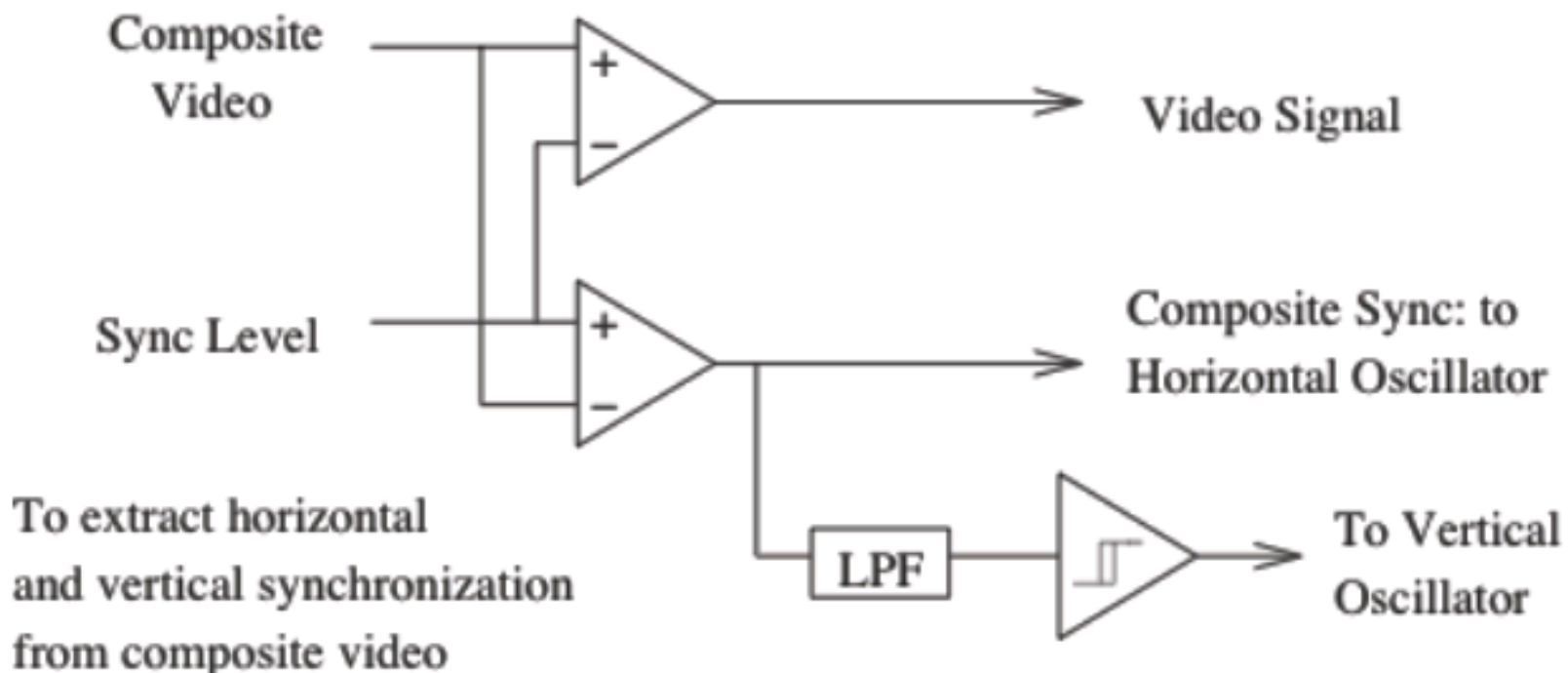
The most common ways to send an image to a video display (even displays that don't use deflection coils, eg, LCDs) require you to generate two sync signals: one for the horizontal dimension (HS) and one for the vertical dimension (VS).





| <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         |

# 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.



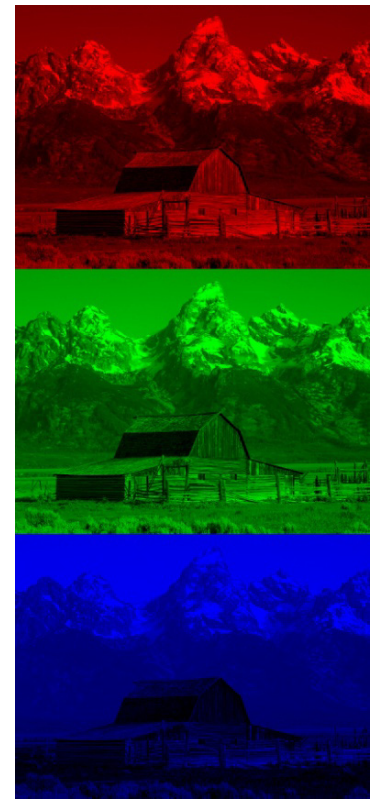
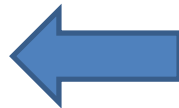
# Video Feature Extraction

- A common technique for finding features in a real-time video stream is to locate the center-of-mass for pixels of a given color
  - Using RGB can be a pain since a color (eg, red) will be represented by a wide range of RGB values depending on the type and intensity of light used to illuminate the scene. Tedious and finicky calibration process required.
- Consider using a HSL/HSV color space
  - H = hue (see diagram)
  - S = saturation, the degree by which color differs from neutral gray (0% to 100%)  

  - L = lightness, illumination of the color (0% to 100%)  

- Filter pixels by hue!



# YCrCb to RGB (for display)

- 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



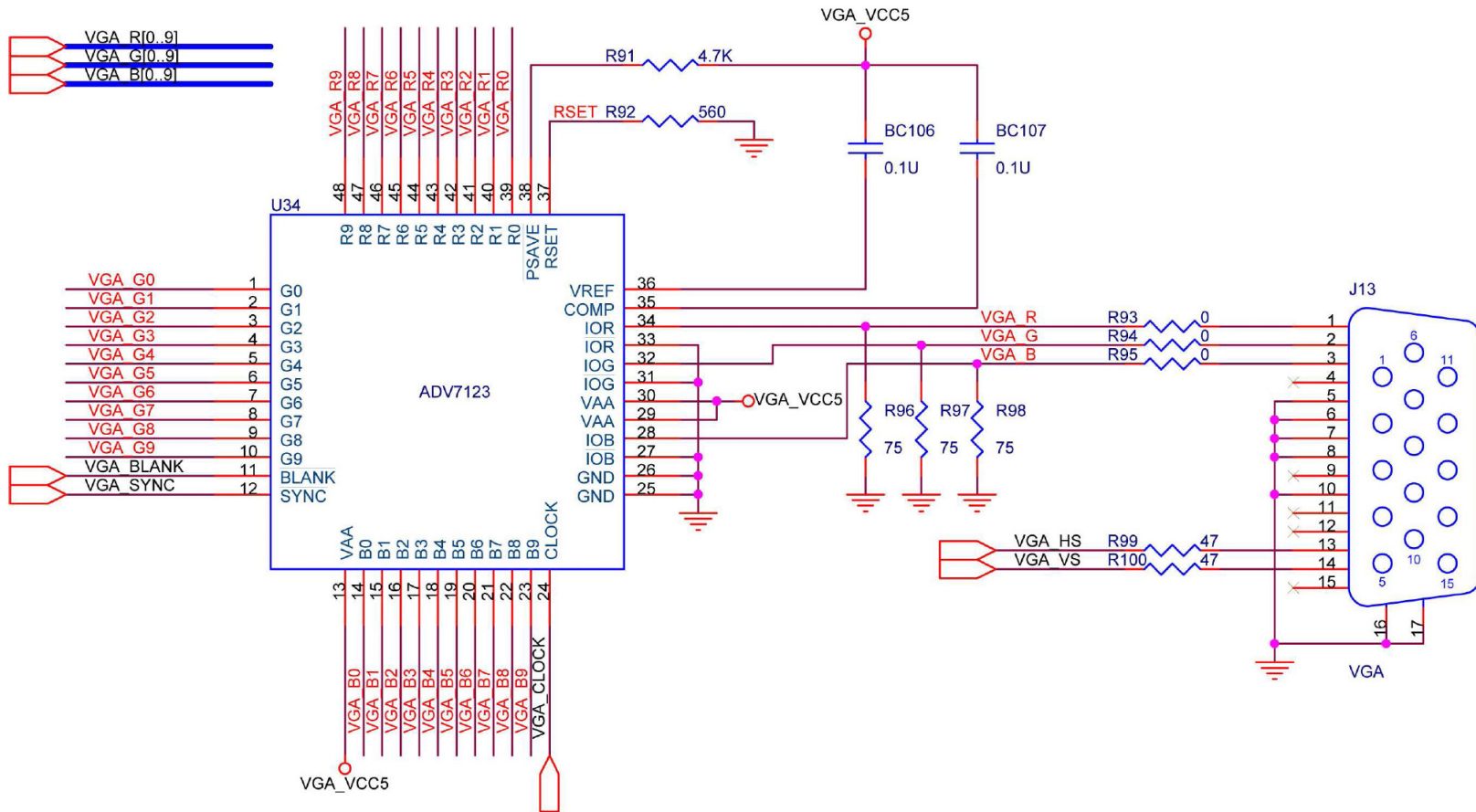


# VGA

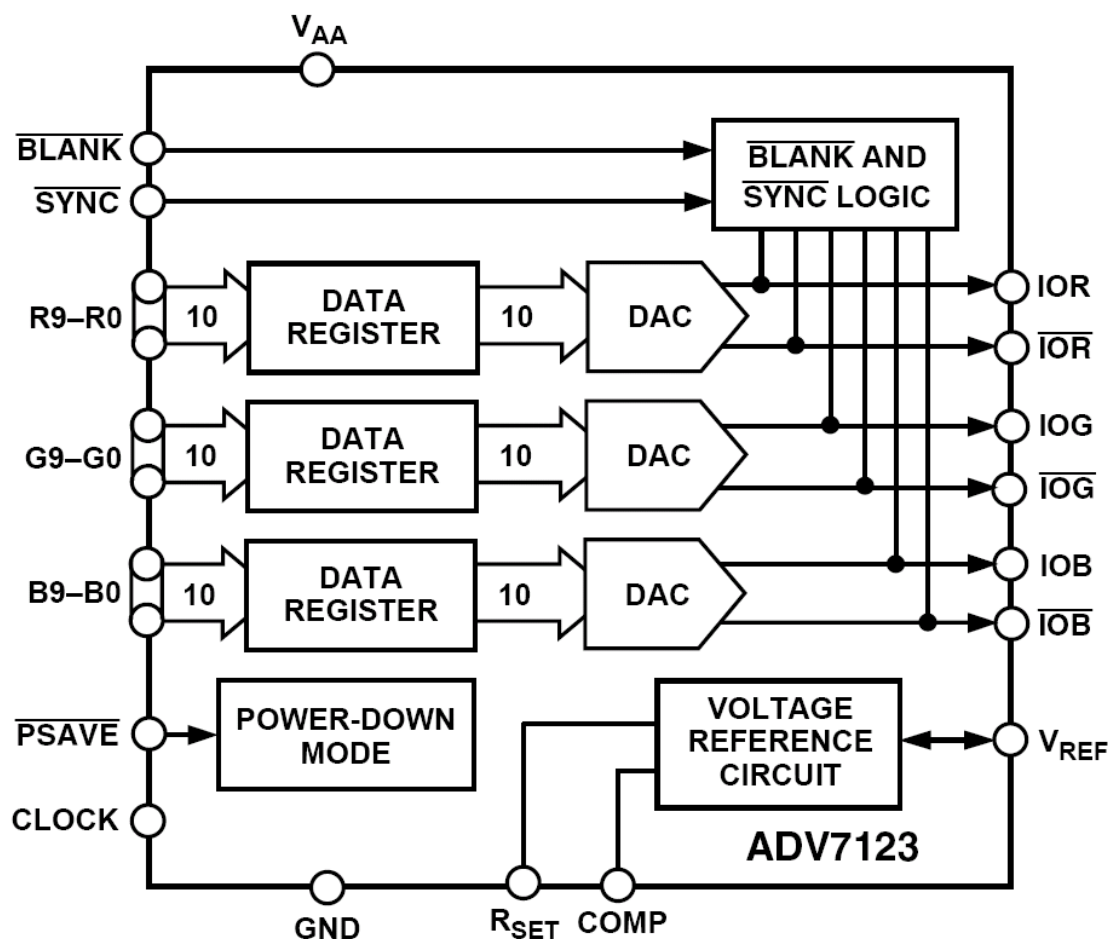
DE2 User manual: page 37: 4.7 Using VGA

## VGA output

- Uses the ADV7123 240-MHz triple 10-bit high-speed video DAC
- With 15-pin high-density D-sub connector
- Supports up to 1600 x 1200 at 100-Hz refresh rate
- Can be used with the Cyclone II FPGA to implement a high-performance TV Encoder

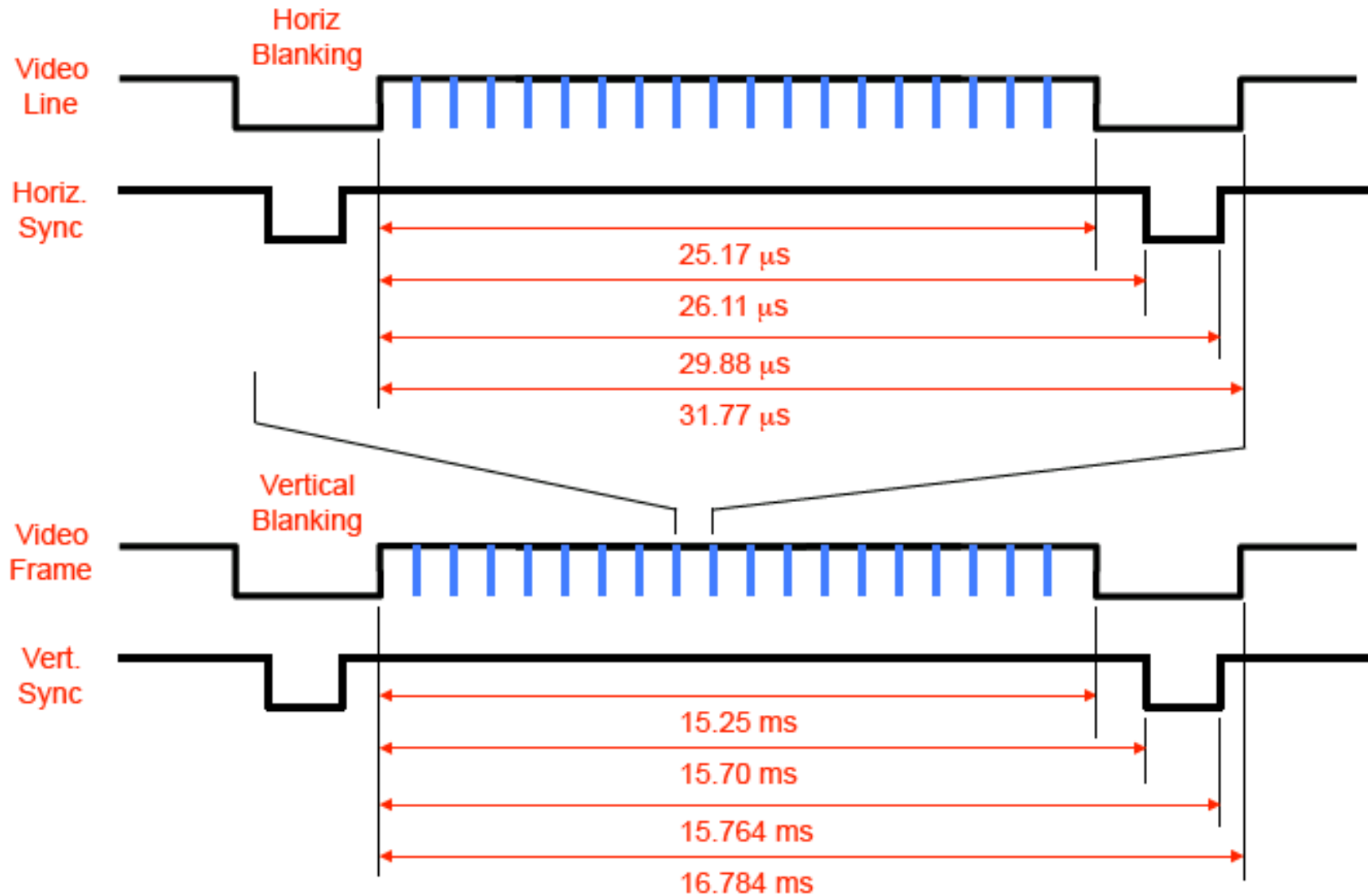


**FUNCTIONAL BLOCK DIAGRAM**

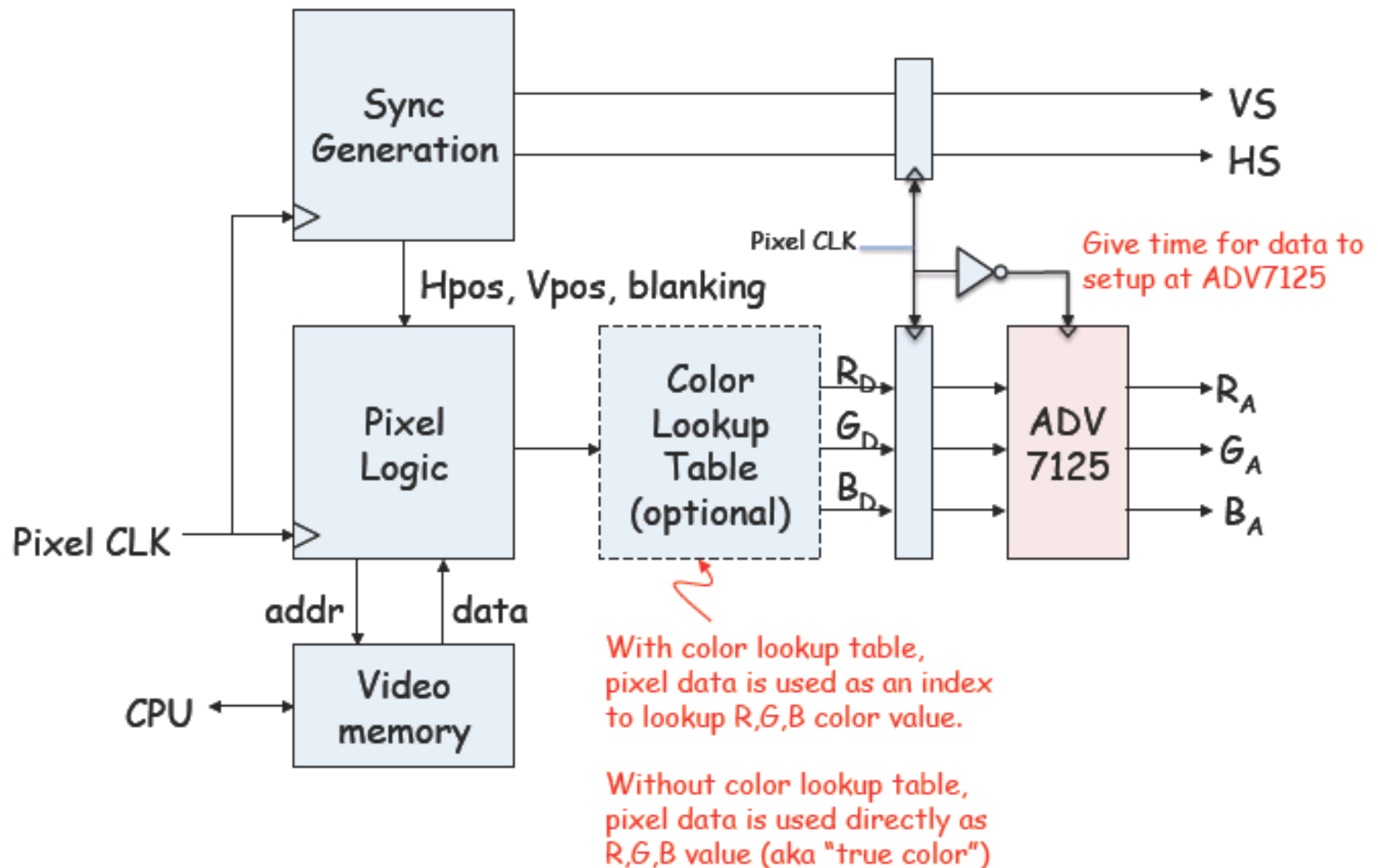




# VGA (640x480) Video

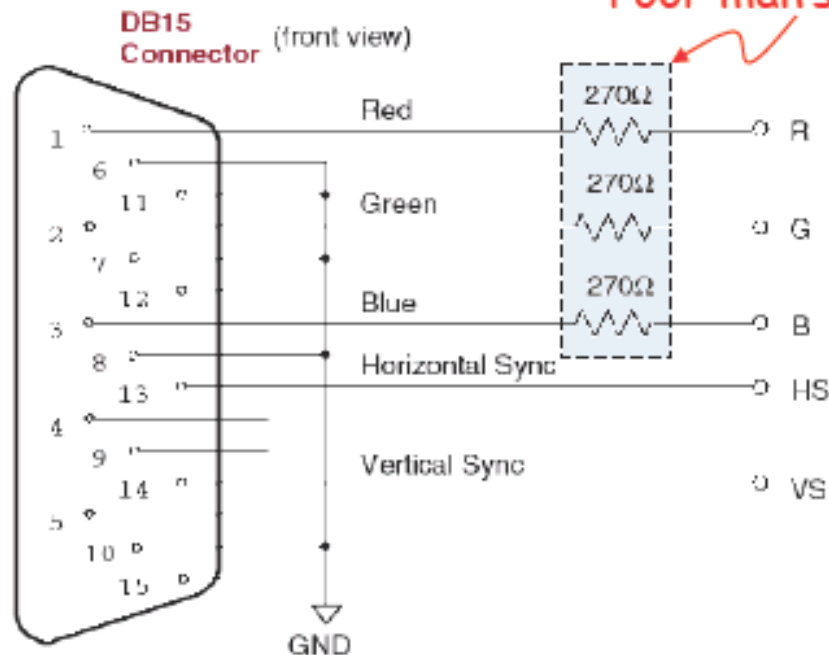


# Generating VGA-style Video



# 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 hsyncon = (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
```

# Video Test Patterns

- Big white rectangle (good for "auto adjust" on monitor)

```
always @(posedge clk) begin
 if (vblank | (hblank & ~hreset)) rgb <= 0;
 else
 rgb <= 7;
end
```

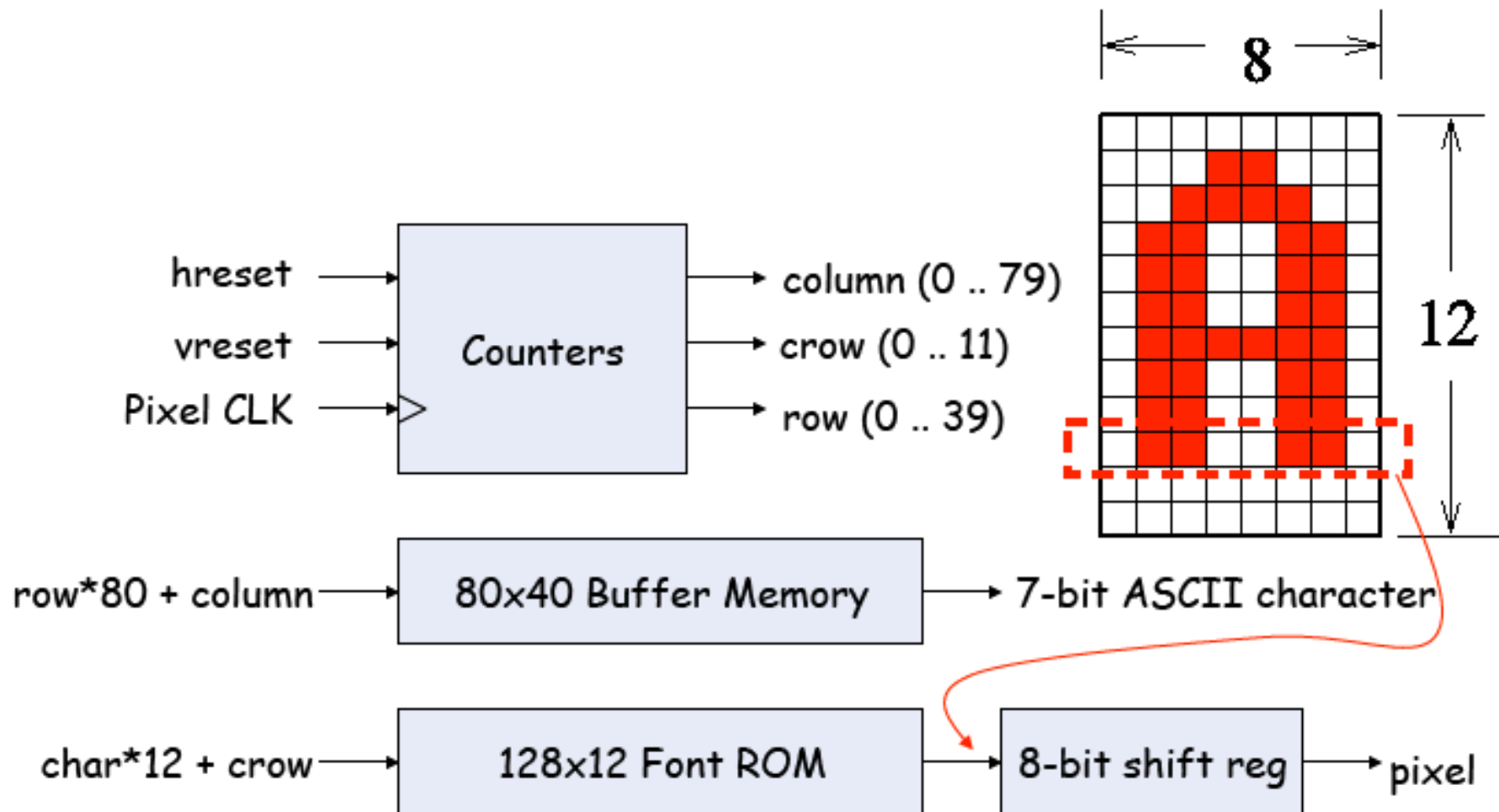
- Color bars

```
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        |

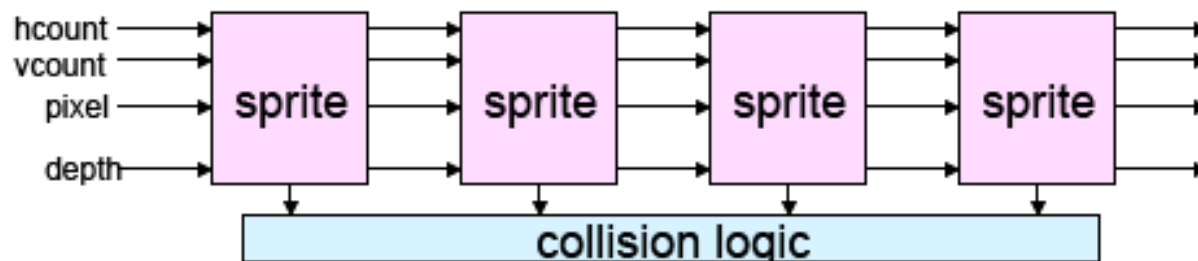
# Character Display

(80 columns x 40 rows, 8x12 glyph)



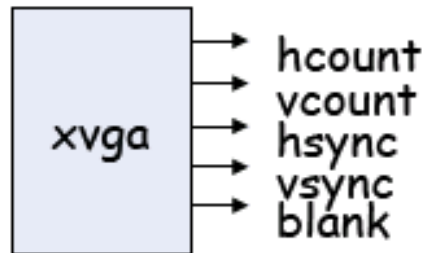
# Game Graphics using Sprites

- Sprite = game object occupying a rectangular region of the screen (it's bounding box).
  - Usually it contains both opaque and transparent pixels.
  - Given (H,V), sprite returns pixel (0=transparent) and depth
  - Pseudo 3D: look at current pixel from all sprites, display the opaque one that's in front (min depth): see sprite pipeline below
  - Collision detection: look for opaque pixels from other sprites
  - Motion: smoothly change coords of upper left-hand corner
- Pixels can be generated by logic or fetched from a bitmap (memory holding array of pixels).
  - Bitmap may have multiple images that can be displayed in rapid succession to achieve animation.
  - Mirroring and 90° rotation by fooling with bitmap address, crude scaling by pixel replication, or resizing filter.



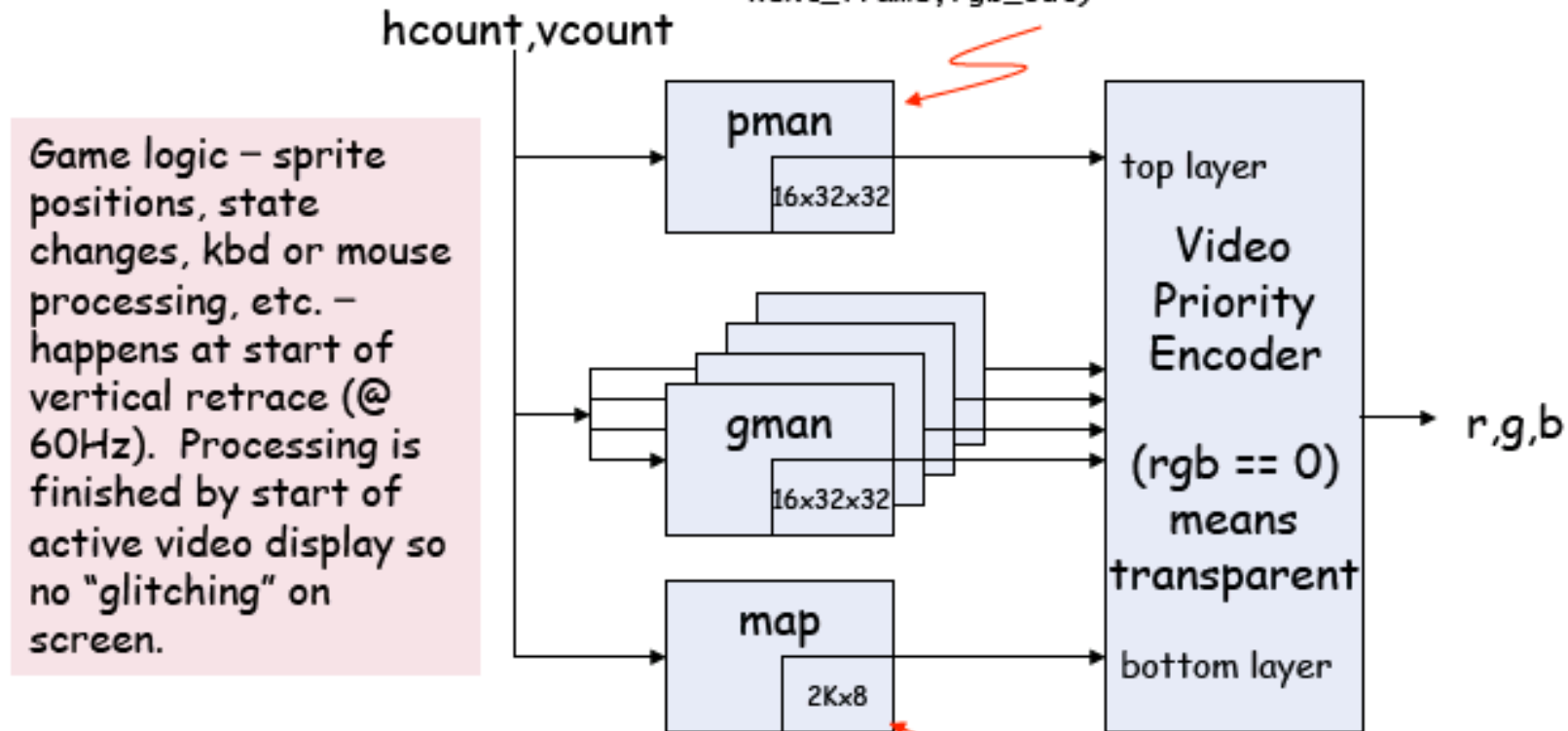


# Pacman



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

## And finally... Microprocessors



Lots of tools and tutorials...

e.g. NIOS IDE (Integrated Development Environment)

DE2 demonstrations

### Tools – SOPC builder

<http://www.altera.com/education/demonstrations/sopc-builder/sopc-builder-demo.html>

[ftp://ftp.altera.com/up/pub/Tutorials/DE2/Computer\\_Organization/tut\\_sopc\\_introduction\\_verilog.pdf](ftp://ftp.altera.com/up/pub/Tutorials/DE2/Computer_Organization/tut_sopc_introduction_verilog.pdf)

The SOPC Builder is a tool used in conjunction with the Quartus II CAD software. It allows the user to easily create a system based on the Nios II processor, by simply selecting the desired functional units and specifying their parameters.

There are other choices of  $\mu$ -processors to implement. E.g.: mips  
Operating systems can be used. E.g.  $\mu$ -Clinux™





# Procurar pal decode/encode



# The lab this week

## The VGA test screen

(It is not a picture!!)

Remember to split the program in blocks!!  
(Will be usefull for your next project)

# Multiplication in Verilog

You can use the "\*" operator to multiply two numbers:

```
wire [9:0] a,b;
wire [19:0] result = a*b; // unsigned multiplication!
```

If you want Verilog to treat your operands as signed two's complement numbers, add the keyword `signed` to your `wire` or `reg` declaration:

```
wire signed [9:0] a,b;
wire signed [19:0] result = a*b; // signed multiplication!
```

Remember: unlike addition and subtraction, you need different circuitry if your multiplication operands are signed vs. unsigned. Same is true of the >>> (arithmetic right shift) operator. To get signed operations all operands must be signed.

To make a signed constant: 10'sh37C





# Interlace

Non-interlaced (aka progressive) scanning:

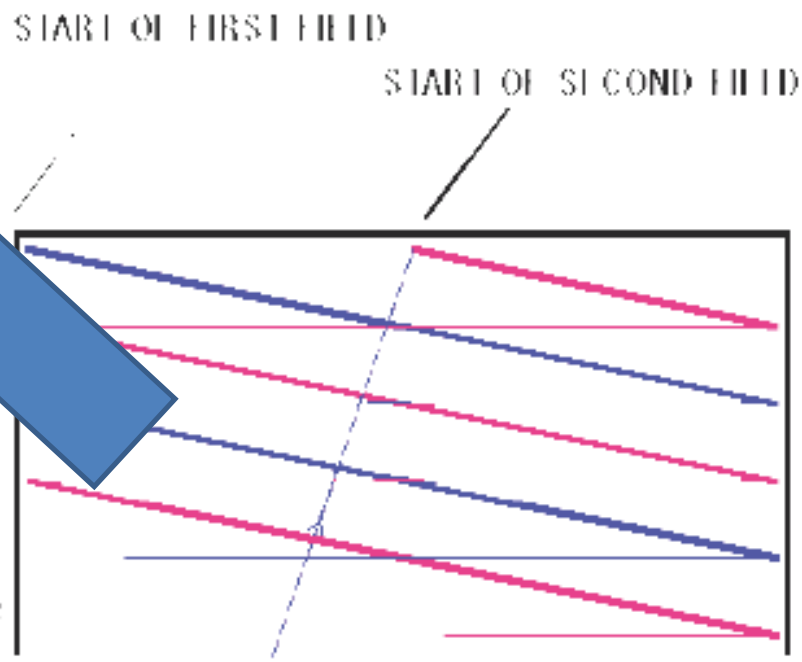
- VS period is a multiple of HS period
- Frame rate = 60Hz to avoid flicker

Interlaced scanning:

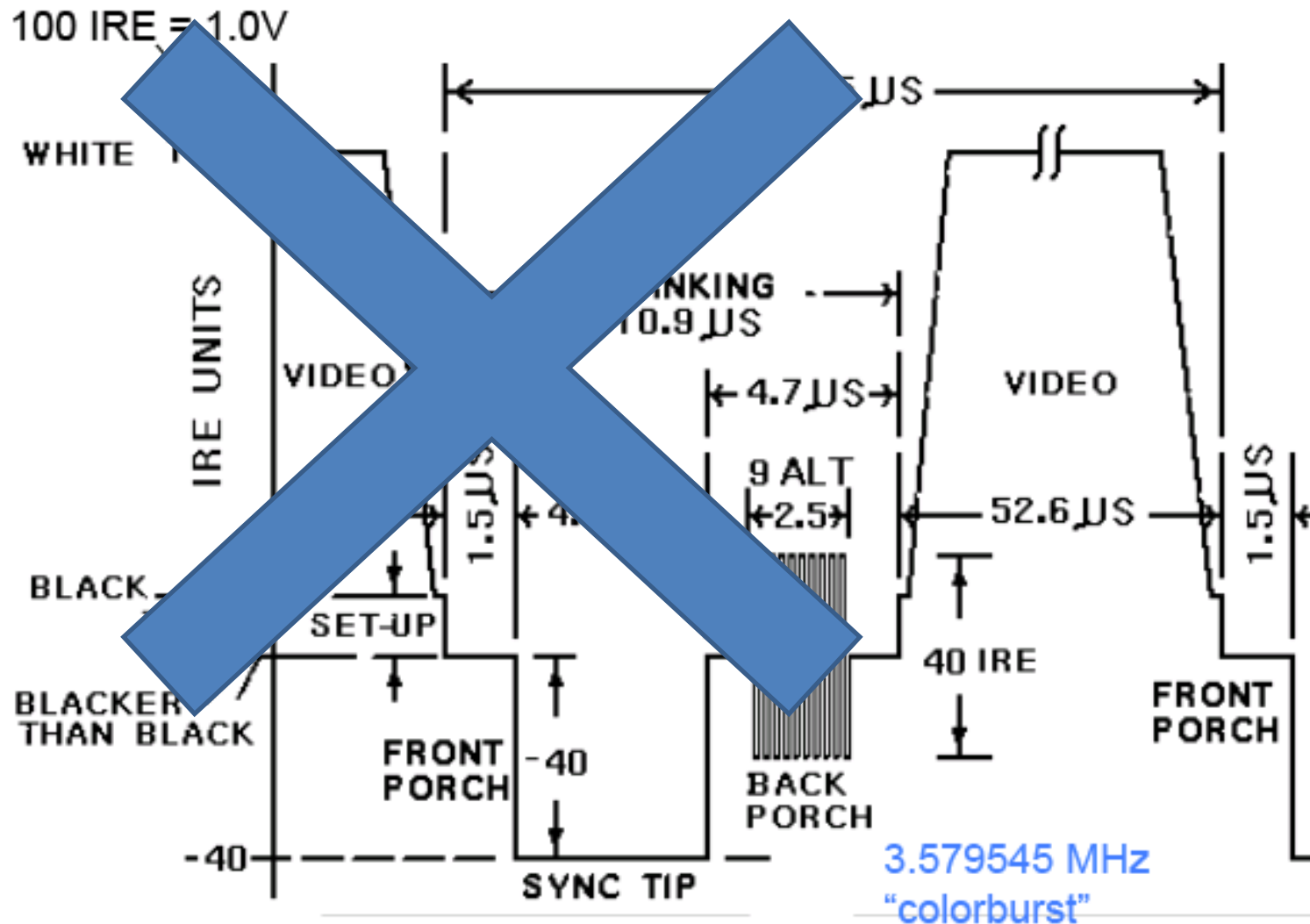
VS period is *not* a multiple of HS period, so successive frames are offset relative to each other. In interlaced scan, so vertical position of scan lines varies from frame to frame.

NTSC example:

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



# NTSC\*: Composite Video Encoding

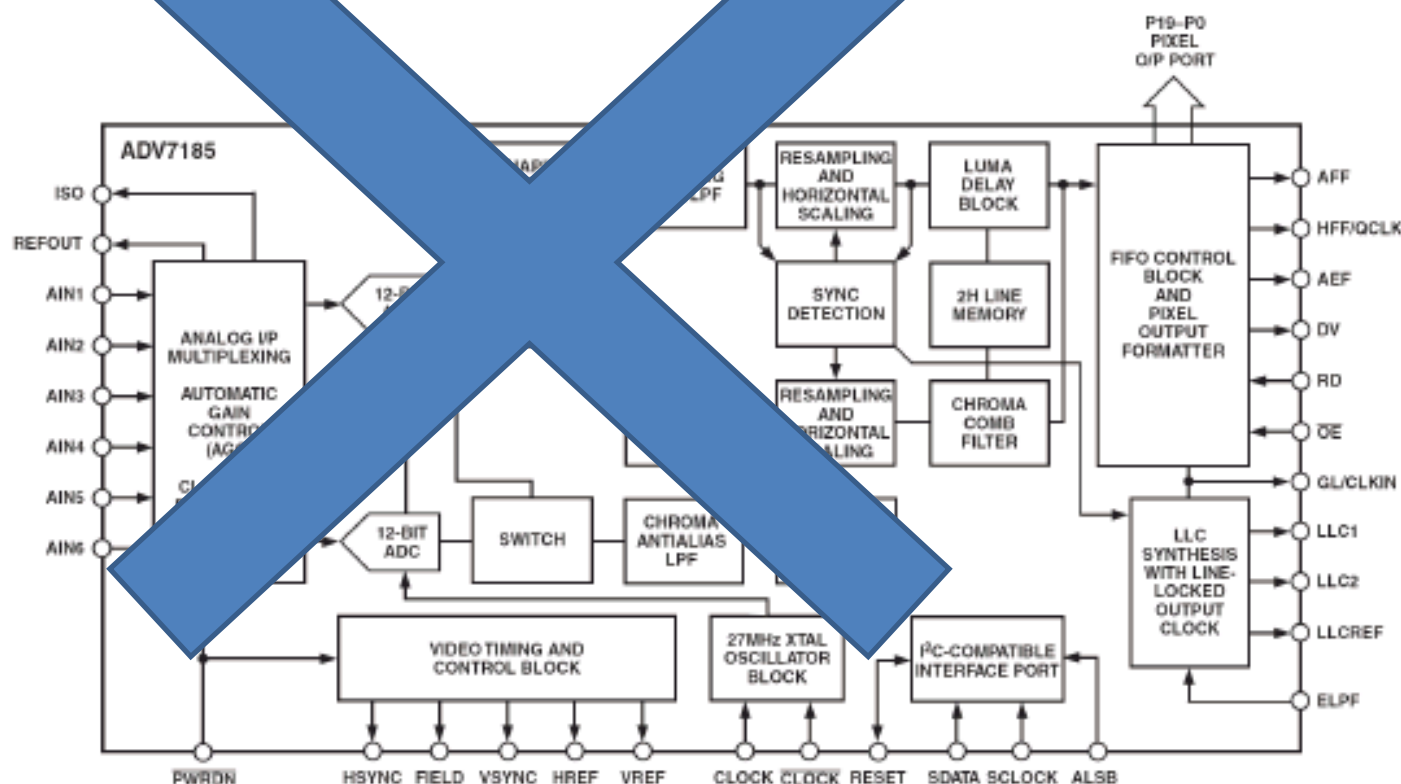


\*National Television System Committee: 1940

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

# Labkit: ADV7185 NTSC Decoder

- Decodes NTSC and PAL video (composite or S-video)
- Produces YUV422 (10-bit) or CCIR601 (8-bit) digital data



# Labkit: ADV7185 NTSC Decoder

- Decodes NTSC and PAL video (composite or S-video)
- Produces YCbCr656 (10-bit) or CCIR601 (8-bit) digital data

| BLANKING PERIOD |    |    | TIMING REFERENCE CODE |    |    |     | 720 PIXELS |     |     |     | 720 DATA |     |                    |      | TIMING REFERENCE CODE |    |    |     | BLANKING PERIOD |    |     |
|-----------------|----|----|-----------------------|----|----|-----|------------|-----|-----|-----|----------|-----|--------------------|------|-----------------------|----|----|-----|-----------------|----|-----|
| ...             | 80 | 10 | FF                    | 00 | 00 | SAV | ...        | ... | ... | ... | Y2       | ... | C <sub>R</sub> 718 | Y719 | FF                    | 00 | 00 | EAV | 80              | 10 | ... |

Pixel 1: Y<sub>1</sub>, C<sub>B</sub>0, C<sub>R</sub>0

Pixel 0: Y<sub>0</sub>, C<sub>B</sub>0, C<sub>R</sub>0

8-bit SAV/EAV: 1FVHabcd  
 10-bit SAV/EAV: 1FVHabcd00  
 F = field 0: even field 1/odd, 1: field 0  
 V = vsync (0 for SAV)  
 H = hsync (0 for SAV)  
 a = V^H  
 b = F^H  
 c = F^V  
 d = F^V^H  
 8h'80, 10'h200 = start of even field  
 8h'C7, 10'h31C = start of odd field

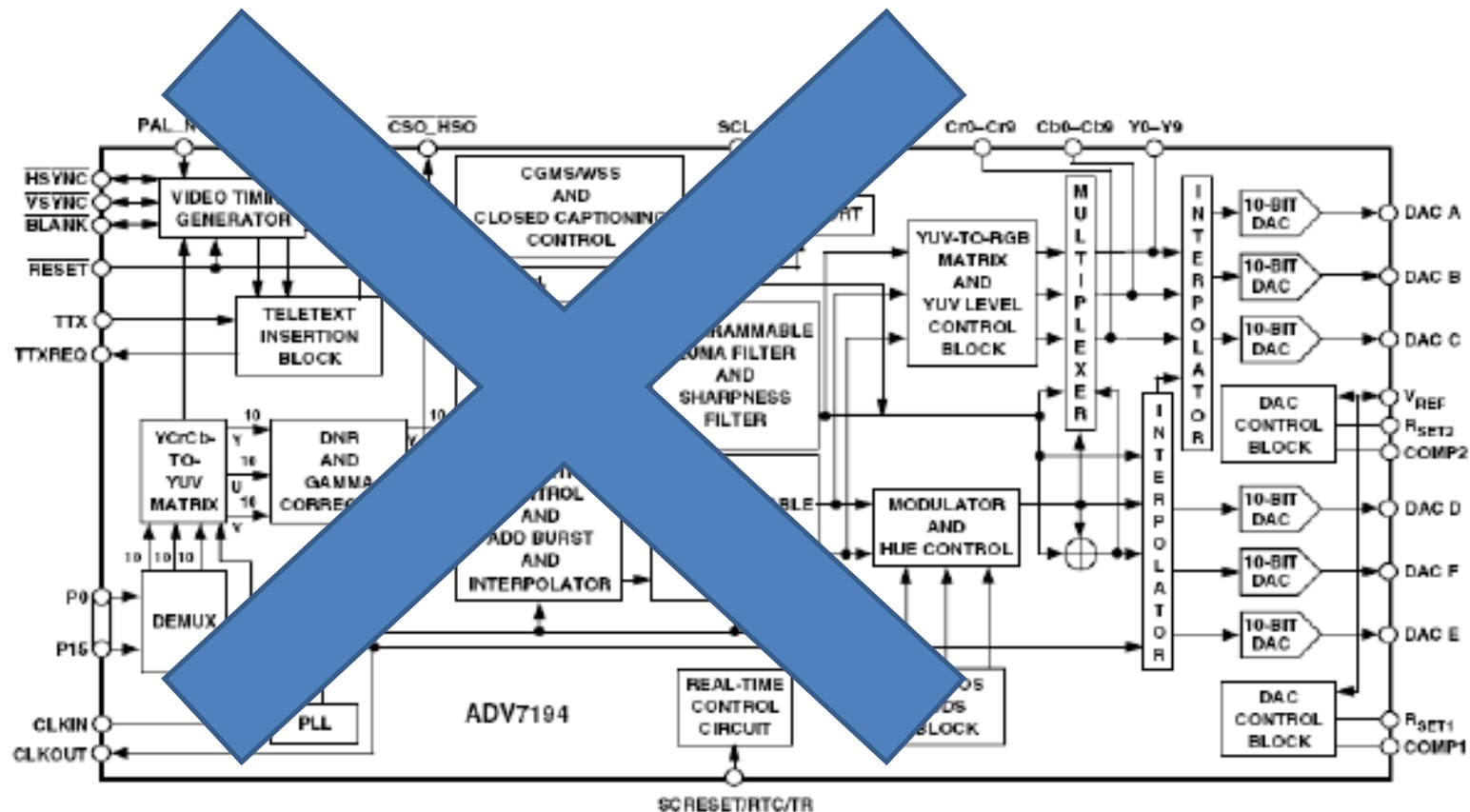
8-bit data:

Y in range 16-235;  
 C<sub>R</sub>, C<sub>B</sub> in range 16-240  
 (offset by 128)

10-bit data:

Y in range 64-943;  
 C<sub>R</sub>, C<sub>B</sub> in range 64-963  
 (offset by 512)

# Labkit: AD7194 Digital Video Encoder



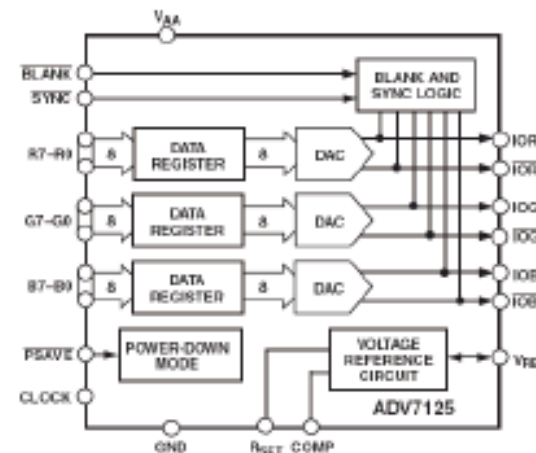
CCIR 601/656 4:2:2 digital video data → analog baseband TV signal

# 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
// 27MHz * (24/10) = 64.8MHz
```

- (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 ...?



# Tools – Mega Wizard

[ftp://ftp.altera.com/up/pub/Tutorials/DE2/Digital\\_Logic/tut\\_lpms\\_verilog.pdf](ftp://ftp.altera.com/up/pub/Tutorials/DE2/Digital_Logic/tut_lpms_verilog.pdf)

## Tools – SOPC builder

<http://www.altera.com/education/demonstrations/sopc-builder/sopc-builder-demo.html>

[ftp://ftp.altera.com/up/pub/Tutorials/DE2/Computer\\_Organization/tut\\_sopc\\_introduction\\_verilog.pdf](ftp://ftp.altera.com/up/pub/Tutorials/DE2/Computer_Organization/tut_sopc_introduction_verilog.pdf)

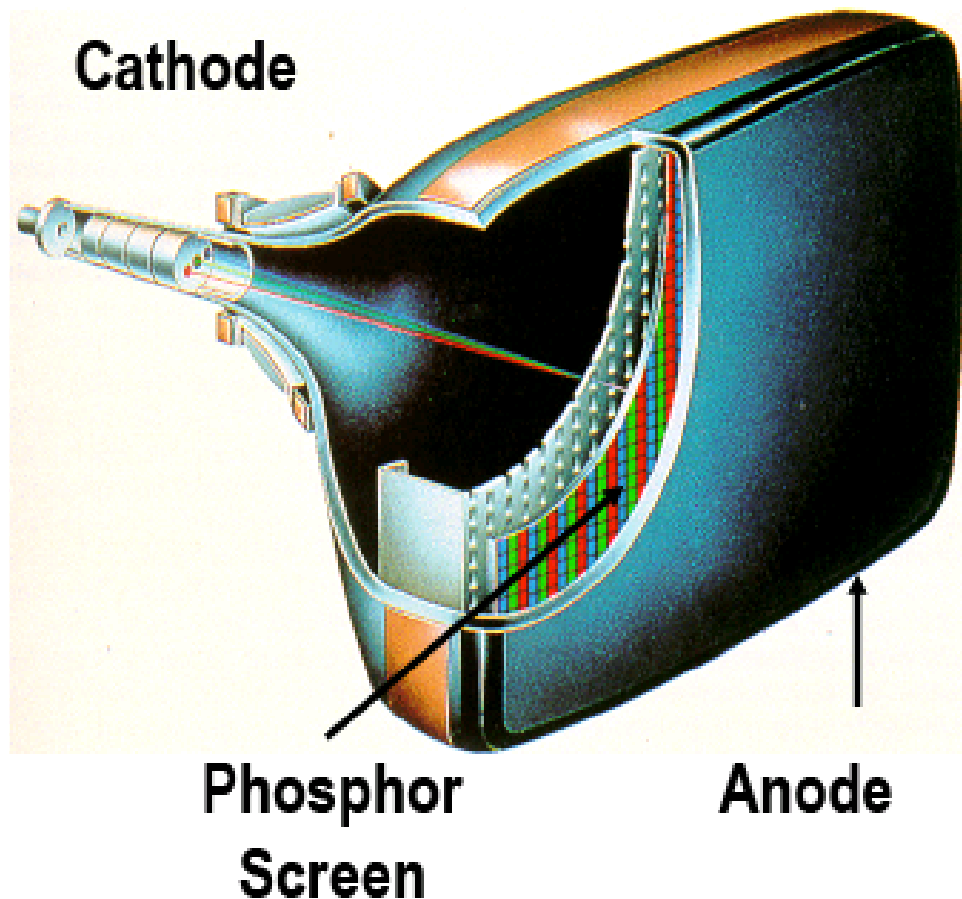
The SOPC Builder is a tool used in conjunction with the Quartus II CAD software. It allows the user to easily create a system based on the Nios II processor, by simply selecting the desired functional units and specifying their parameters.

NOVA  
VERSÃO



# Cathode Ray Tube

## CRT Display



Electrons beam “boiled off a metal” by heat (**thermionic emission**) is sequentially scanned across a phosphor screen by magnetic deflection. The electrons are accelerated to the screen acquiring energy and generate light on reaching the screen (**cathodoluminescence**)

Courtesy of PixTech

# RS232 interface

- Transmit: easy, just build FSM to generate desired waveform with correct bit timing
- Receive:
  - Want to sample value in middle of each bit time
  - Oversample, eg, at 16x baud rate
  - Look for 1→0 transition at beginning of start bit
  - Count to 8 to sample start bit, then repeatedly count to 16 to sample other bits
  - Check format (start, data, parity, stop) before accepting data.

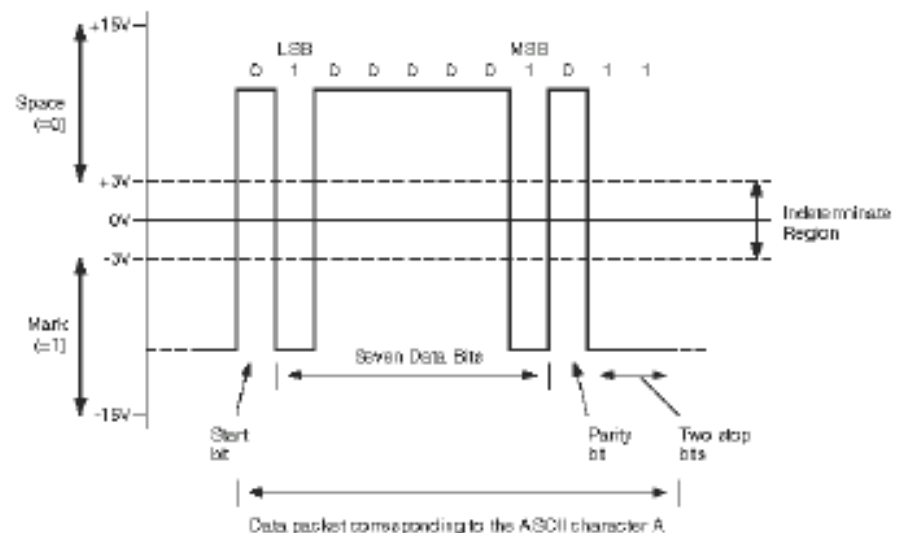


Figure from  
<http://www.arcelect.com/rs232.htm>