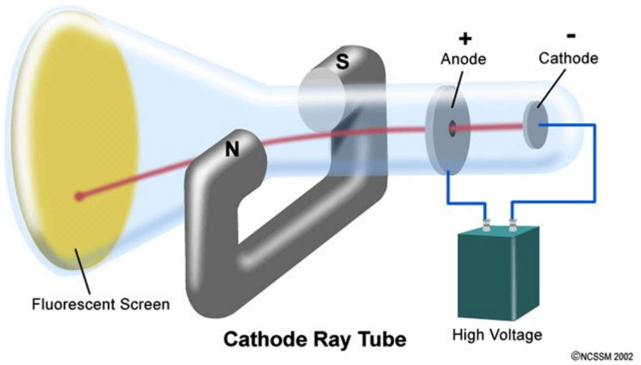
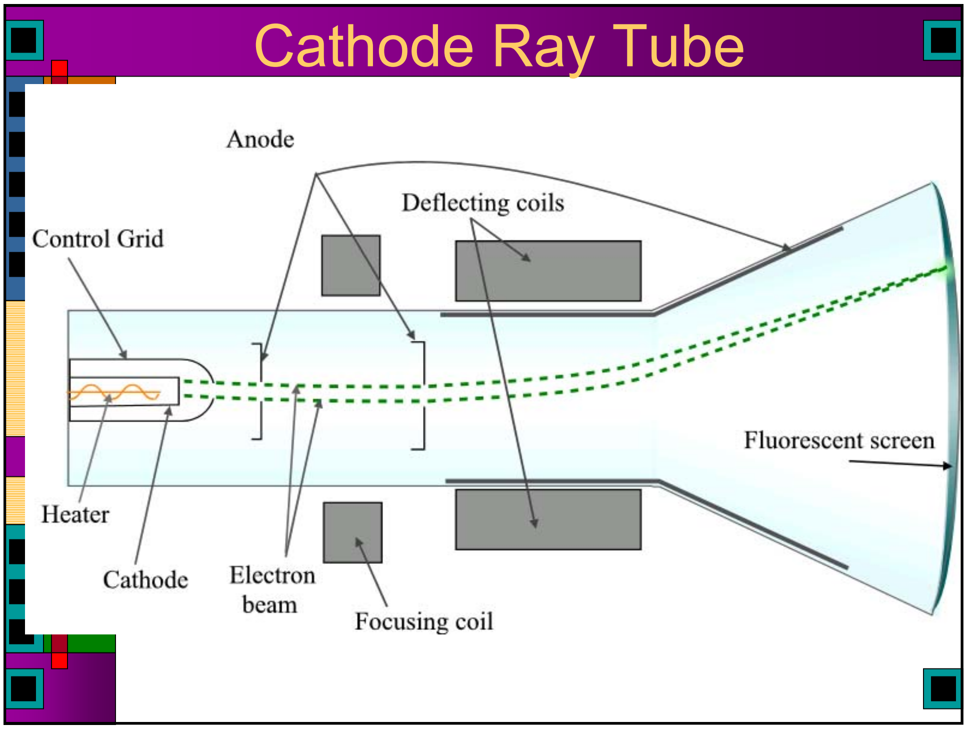
Display Technology

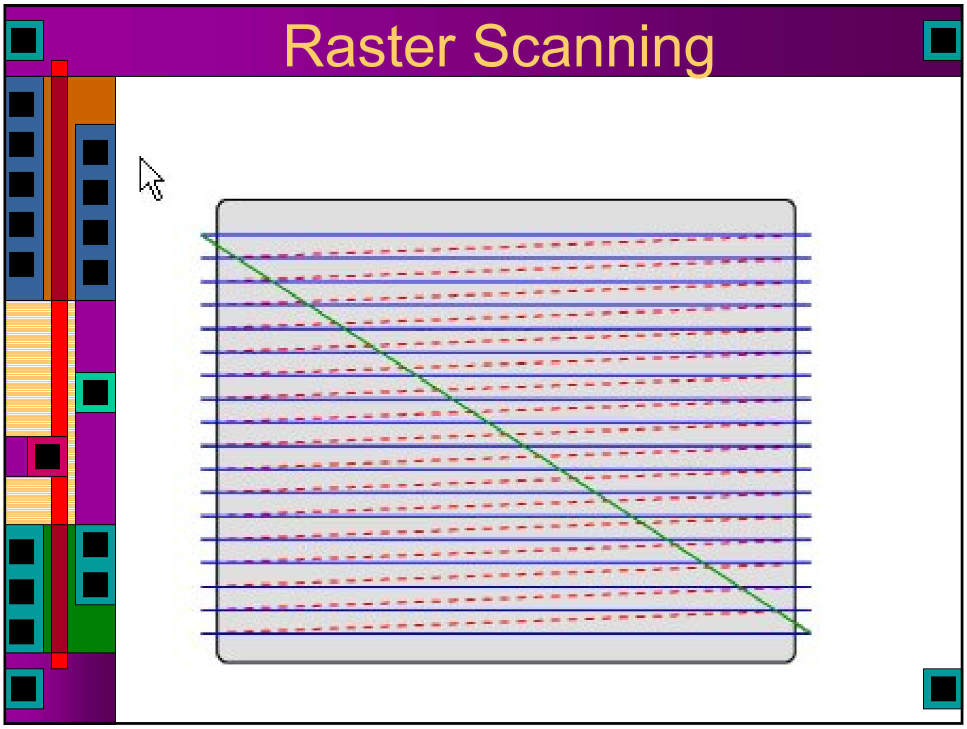
- Images stolen from various locations on the web...

Cathode Ray Tube



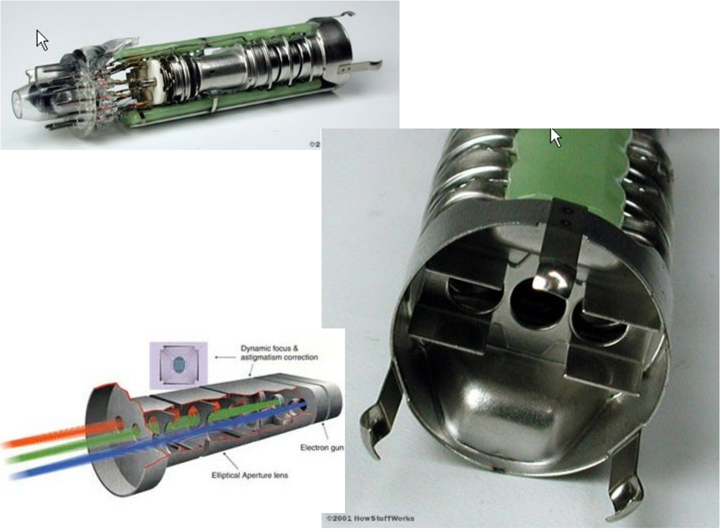


Cathode Ray Tube



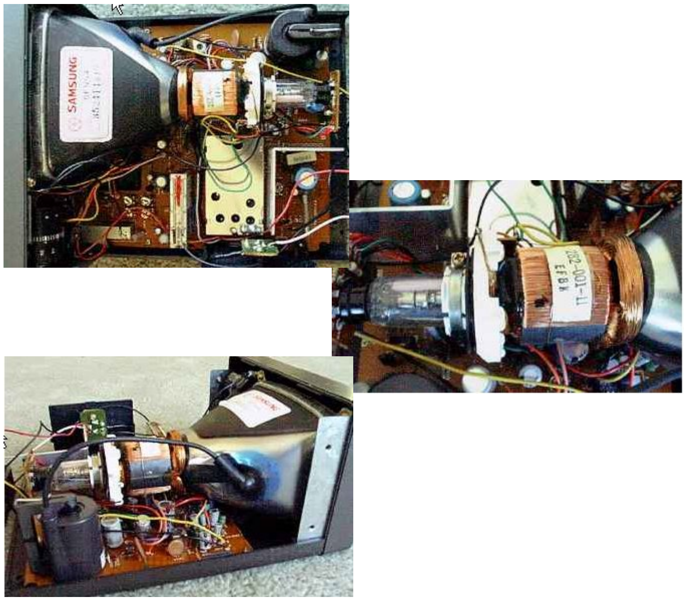
Raster Scanning

Electron Gun

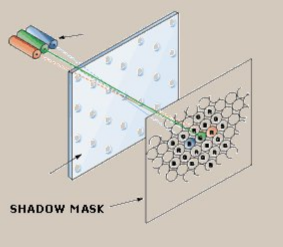


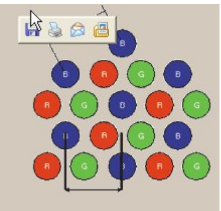
©2001 HowStuffWorks

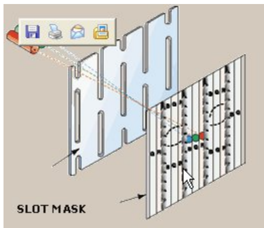
Beam Steering Coils

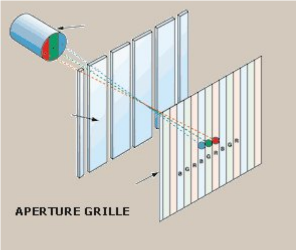


Color

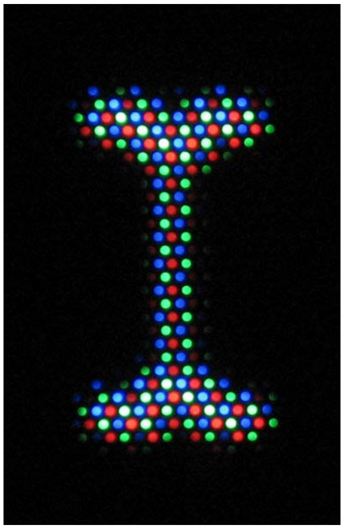


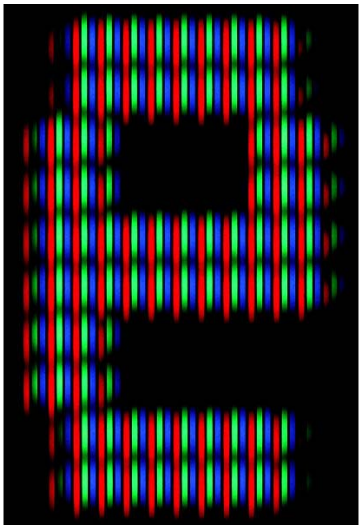


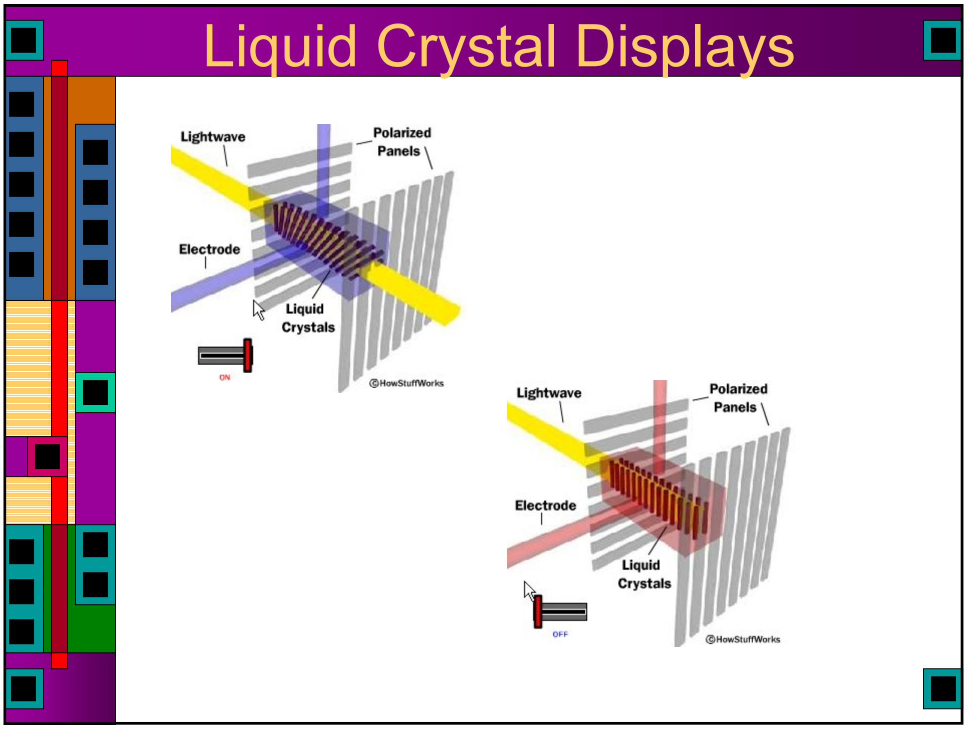




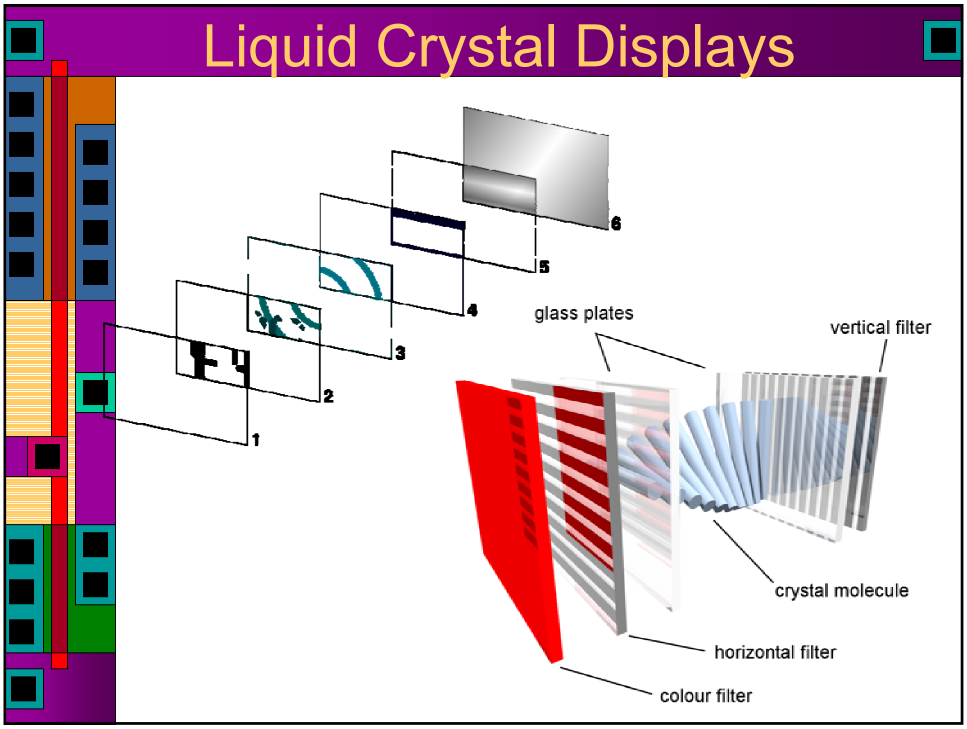
Shadow Mask and Aperture Grille



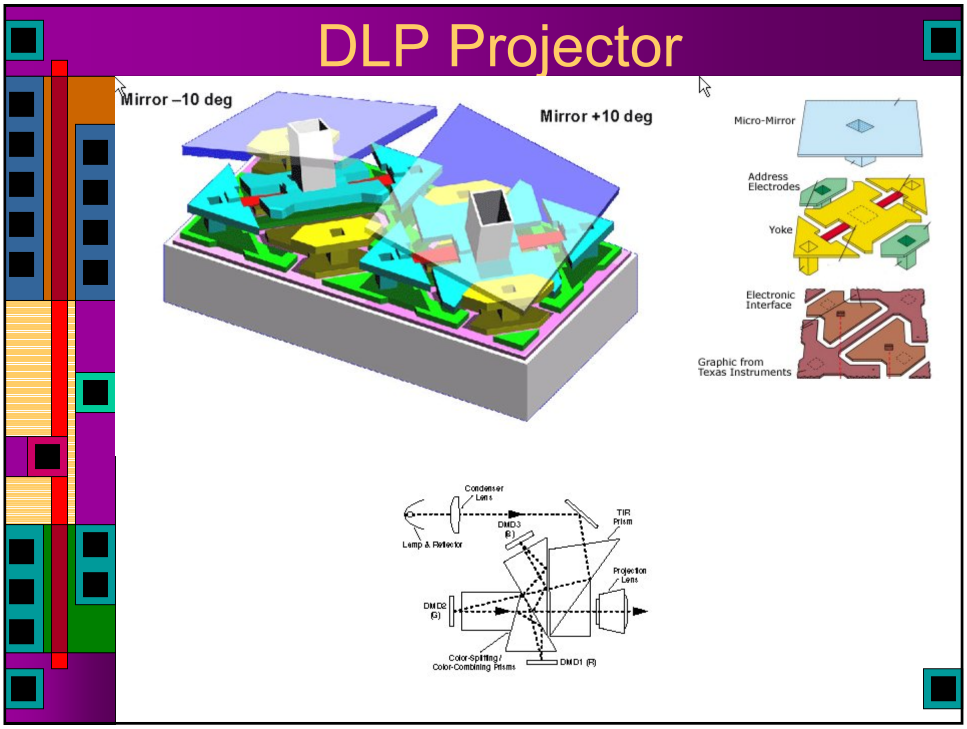




Liquid Crystal Displays

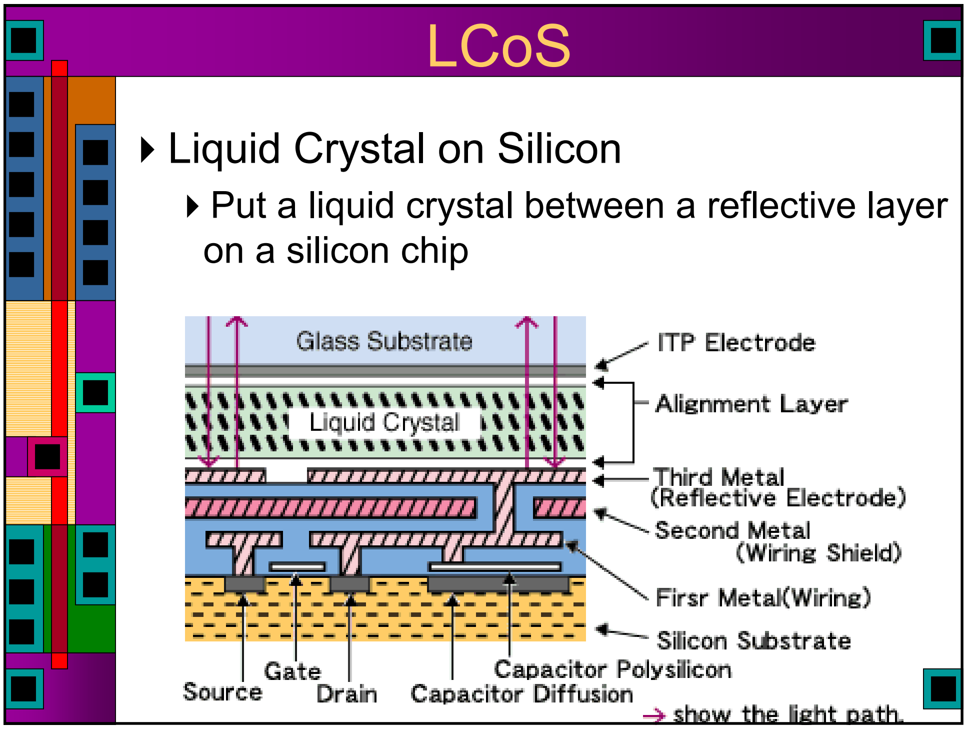


Liquid Crystal Displays



DLP Projector

Graphic from Texas Instruments

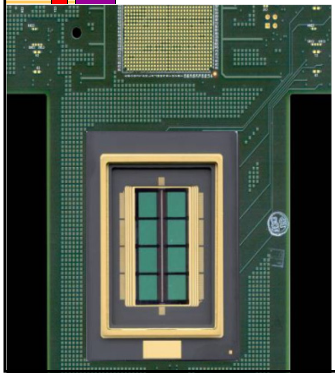


LCoS

- Liquid Crystal on Silicon
  - Put a liquid crystal between a reflective layer on a silicon chip

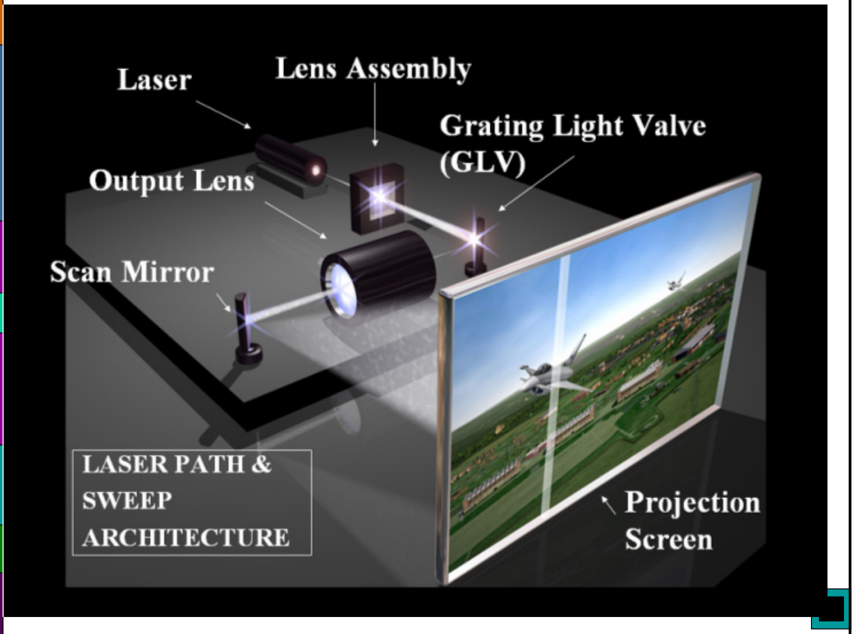
→ show the light path.

Grating Light Valve (GLS)

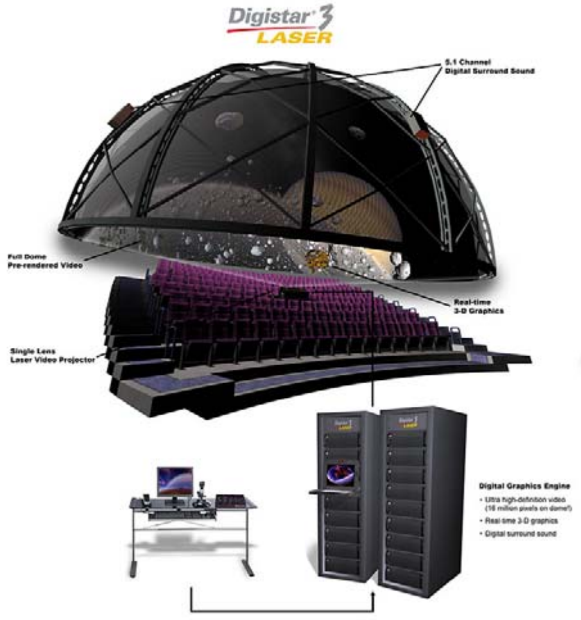


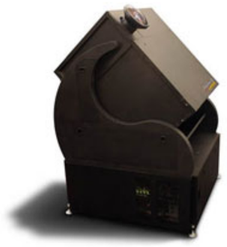
- lots (8000 currently) of micro ribbons that can bend slightly
  - Make them reflective
  - The bends make a diffraction grating that controls how much light where
  - Scan it with a laser for high light output
  - 4000 pixel wide frame ever 60Hz

Grating Light Valve (GLS)



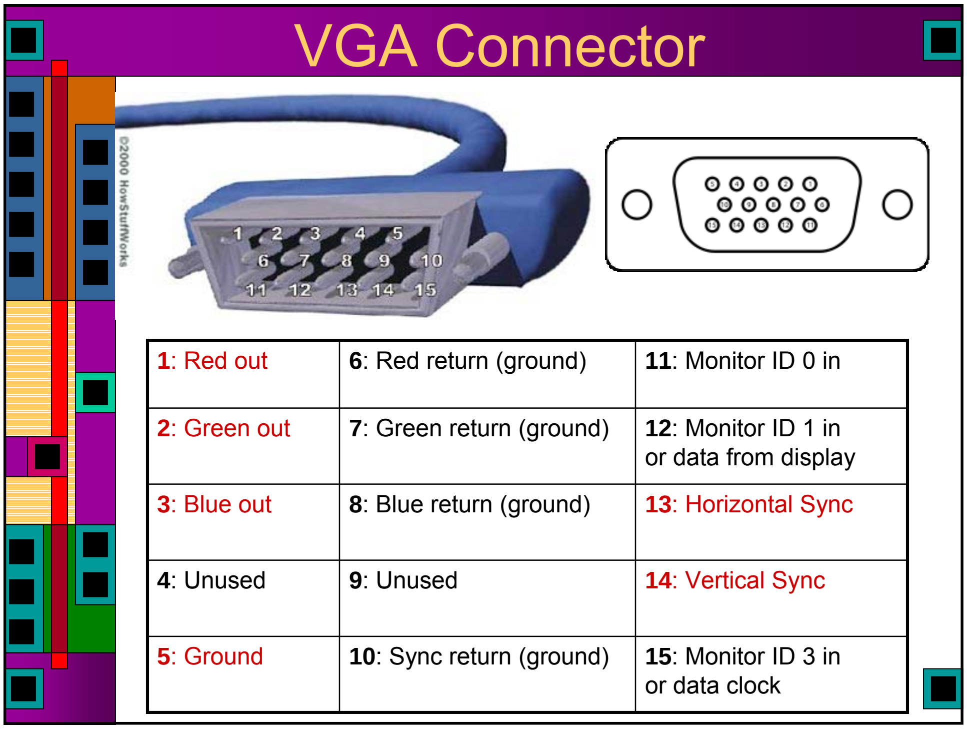
Digistar 3 Dome Projector


Digistar 3 LASER
5.1 Channel Digital Surround Sound
Full Dome Pre-rendered Video
Real-time 3-D Graphics
Single Lens Laser Video Projector
Digital Graphics Engine
• Ultra high-definition video (16 million pixels on dome!)
• Real-time 3-D graphics
• Digital surround sound



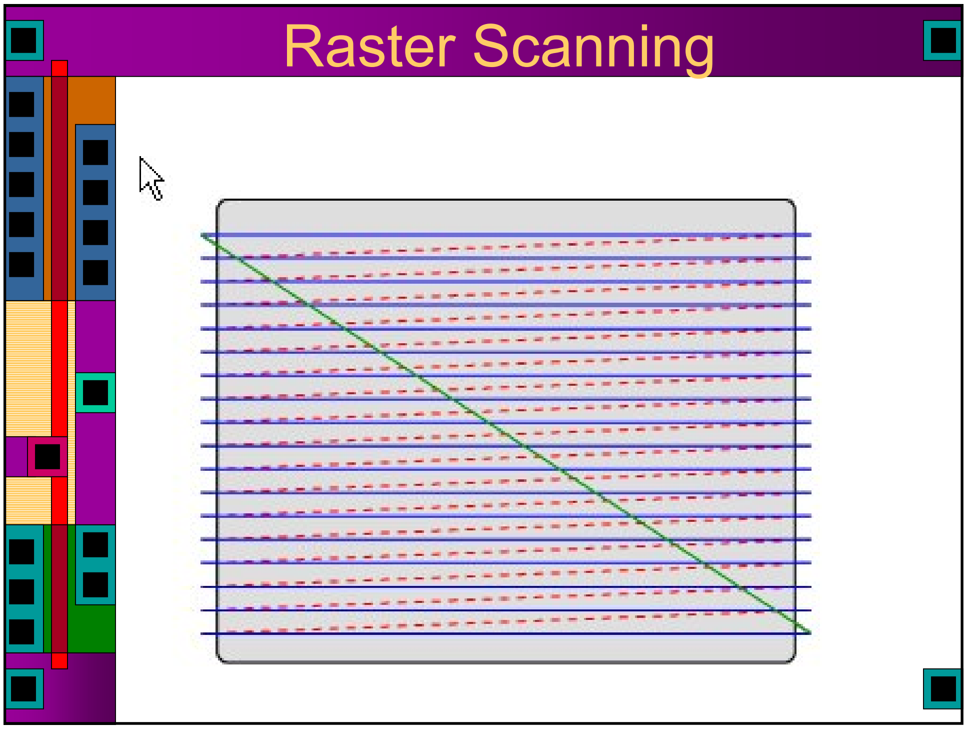
VGA

- Stands for Video Graphics Array
- A standard defined by IBM back in 1987
  - 640 x 480 pixels
  - Now superseded by much higher resolution standards...
- Also means a specific analog connector
  - 15-pin D-subminiature VGA connector

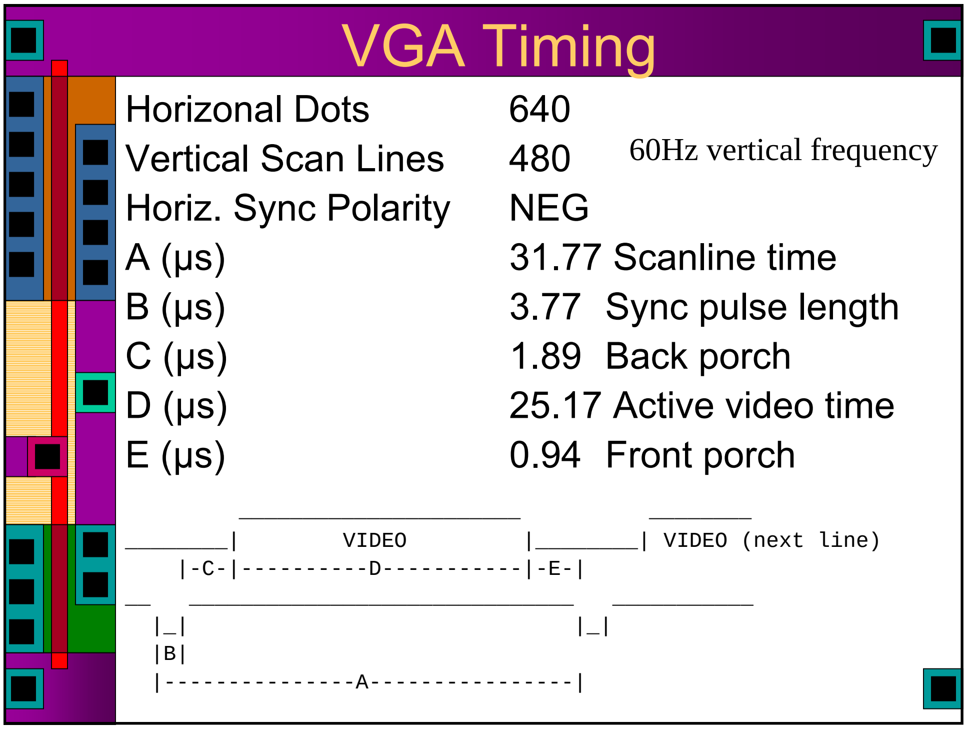


VGA Connector

1: Red out	**6**: Red return (ground)	**11**: Monitor ID 0 in
2: Green out	**7**: Green return (ground)	**12**: Monitor ID 1 in or data from display
3: Blue out	**8**: Blue return (ground)	**13**: Horizontal Sync
4: Unused	**9**: Unused	**14**: Vertical Sync
5: Ground	**10**: Sync return (ground)	**15**: Monitor ID 3 in or data clock



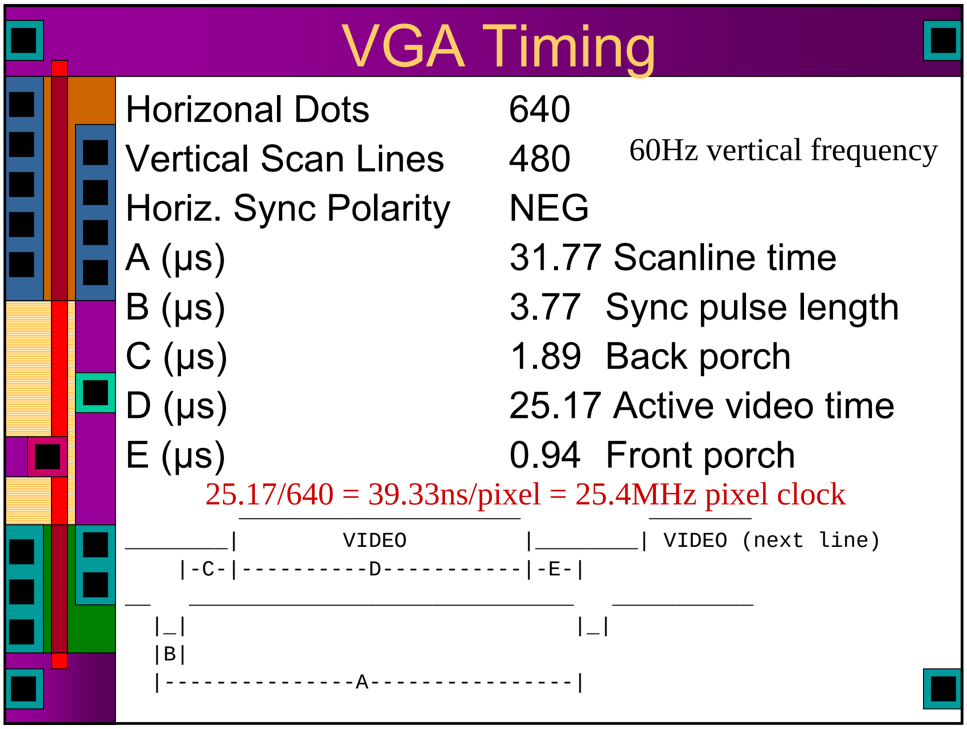
Raster Scanning



VGA Timing

Horizonal Dots	640	
Vertical Scan Lines	480	60Hz vertical frequency
Horiz. Sync Polarity	NEG	
A (µs)	31.77 Scanline time	
B (µs)	3.77 Sync pulse length	
C (µs)	1.89 Back porch	
D (µs)	25.17 Active video time	
E (µs)	0.94 Front porch	

```
         ______________________            ________
________|         VIDEO          |_______| VIDEO (next line)
    |-C-|----------D------------|-E-|
__   ______________________________   ___________
  |_|                              |_|
  |B|
  |---------------A----------------|
```

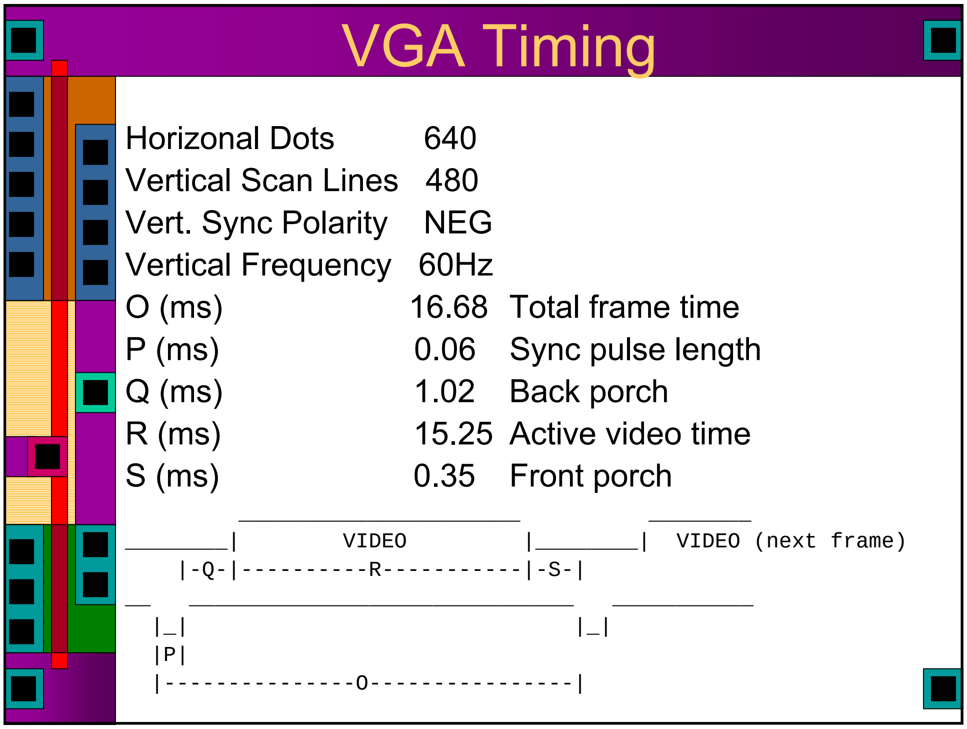


VGA Timing

Horizonal Dots	640	
Vertical Scan Lines	480	60Hz vertical frequency
Horiz. Sync Polarity	NEG	
A (µs)	31.77 Scanline time	
B (µs)	3.77 Sync pulse length	
C (µs)	1.89 Back porch	
D (µs)	25.17 Active video time	
E (µs)	0.94 Front porch	

25.17/640 = 39.33ns/pixel = 25.4MHz pixel clock

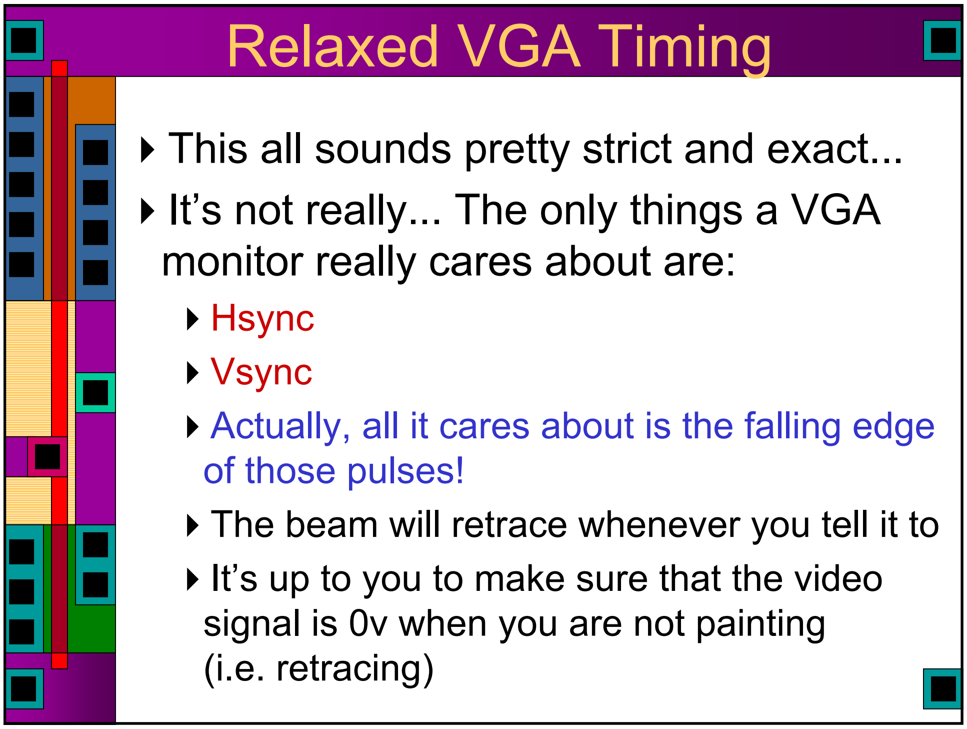
```
         ______________________            ________
________|         VIDEO          |_______| VIDEO (next line)
    |-C-|----------D------------|-E-|
__   ______________________________   ___________
  |_|                              |_|
  |B|
  |---------------A----------------|
```



VGA Timing

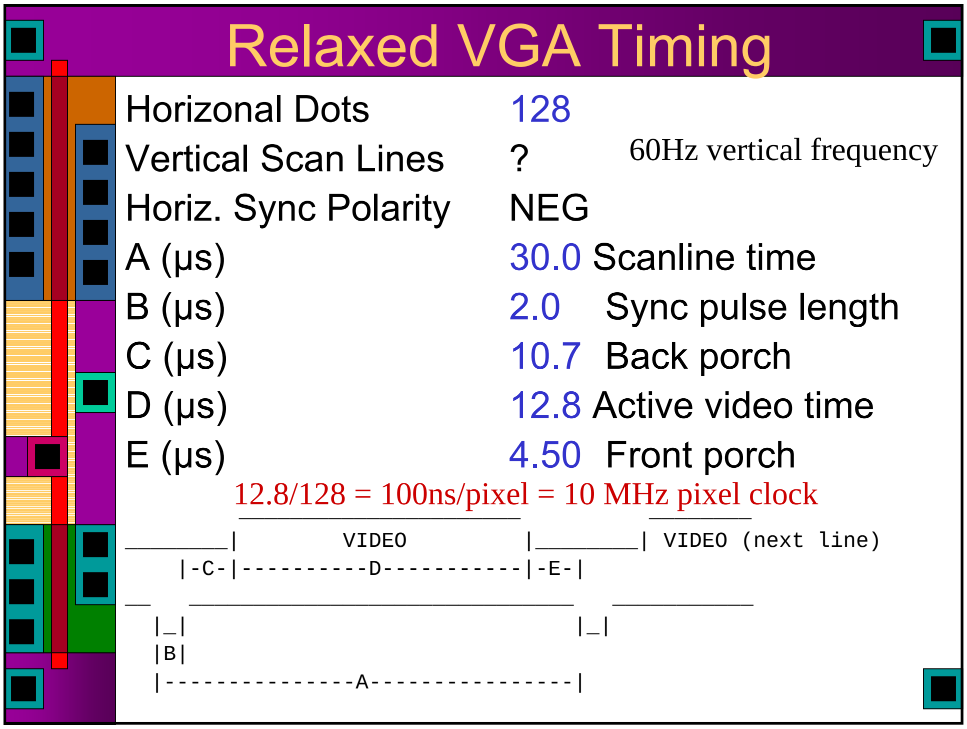
Horizonal Dots	640	
Vertical Scan Lines	480	
Vert. Sync Polarity	NEG	
Vertical Frequency	60Hz	
O (ms)	16.68	Total frame time
P (ms)	0.06	Sync pulse length
Q (ms)	1.02	Back porch
R (ms)	15.25	Active video time
S (ms)	0.35	Front porch

```
         _______________________         ________
________|         VIDEO         |________|  VIDEO (next frame)
    |-Q-|----------R-----------|-S-|
__   ______________________________   ___________
  |_|                            |_|
  |P|
  |---------------O---------------|
```



Relaxed VGA Timing

- This all sounds pretty strict and exact...
- It's not really... The only things a VGA monitor really cares about are:
  - Hsync
  - Vsync
  - Actually, all it cares about is the falling edge of those pulses!
  - The beam will retrace whenever you tell it to
  - It's up to you to make sure that the video signal is 0v when you are not painting (i.e. retracing)

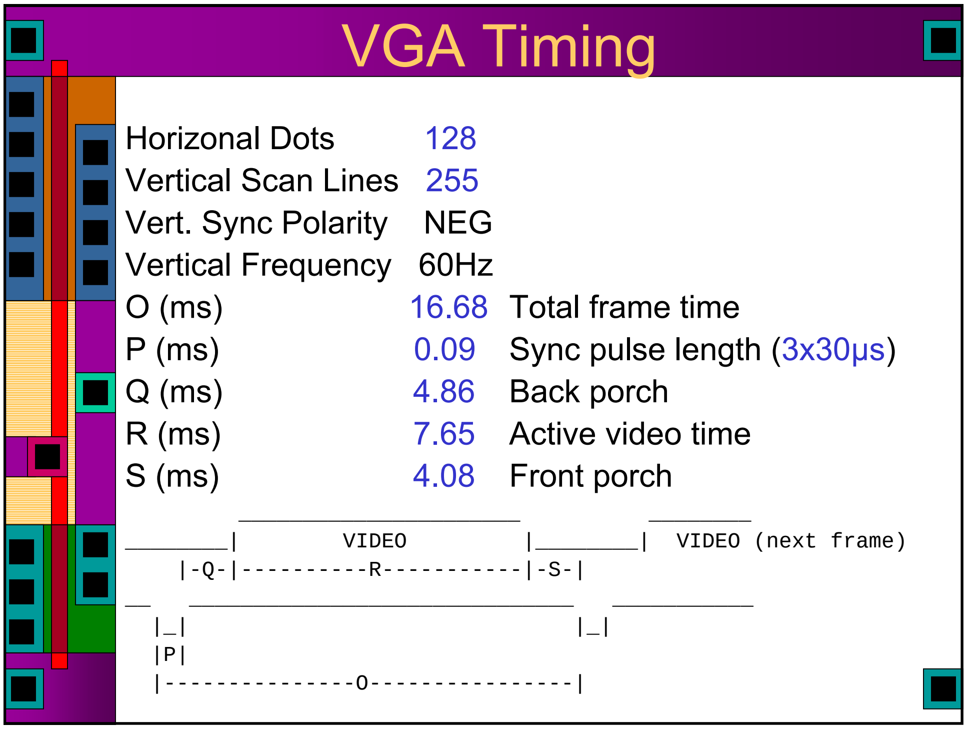


Relaxed VGA Timing

Horizonal Dots	128	
Vertical Scan Lines	?	60Hz vertical frequency
Horiz. Sync Polarity	NEG	
A (μs)	30.0	Scanline time
B (μs)	2.0	Sync pulse length
C (μs)	10.7	Back porch
D (μs)	12.8	Active video time
E (μs)	4.50	Front porch

12.8/128 = 100ns/pixel = 10 MHz pixel clock

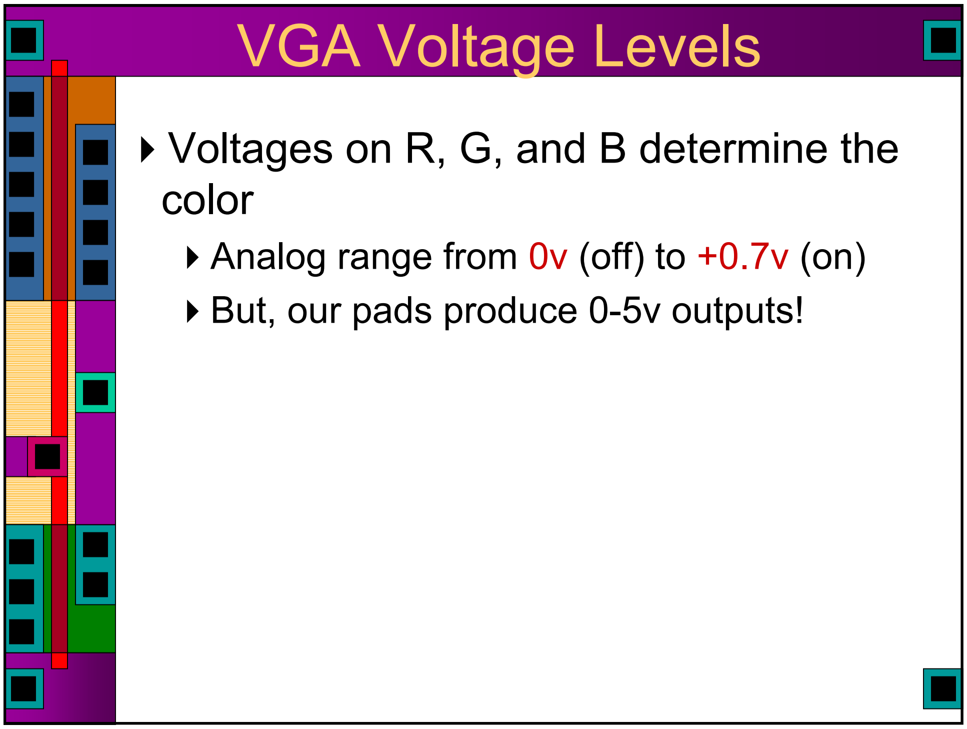
```
         ______________________          ________
________|        VIDEO         |________| VIDEO (next line)
    |-C-|----------D-----------|-E-|

__   ______________________________   ___________
  |_|                              |_|
  |B|
  |---------------A----------------|
```



VGA Timing

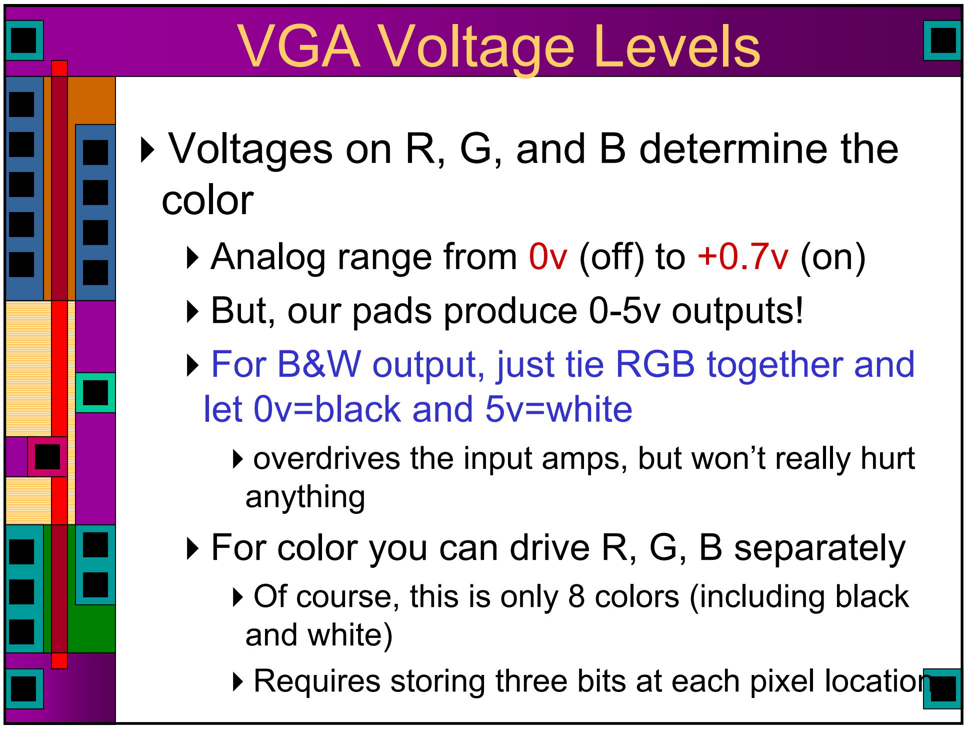
Horizonal Dots	128	
Vertical Scan Lines	255	
Vert. Sync Polarity	NEG	
Vertical Frequency	60Hz	
O (ms)	16.68	Total frame time
P (ms)	0.09	Sync pulse length (3x30μs)
Q (ms)	4.86	Back porch
R (ms)	7.65	Active video time
S (ms)	4.08	Front porch

```
         ______________________          ________
________|        VIDEO         |________|  VIDEO (next frame)
    |-Q-|----------R-----------|-S-|

__   ______________________________   ___________
  |_|                              |_|
  |P|
  |---------------O----------------|
```



VGA Voltage Levels

- Voltages on R, G, and B determine the color
  - Analog range from 0v (off) to +0.7v (on)
  - But, our pads produce 0-5v outputs!

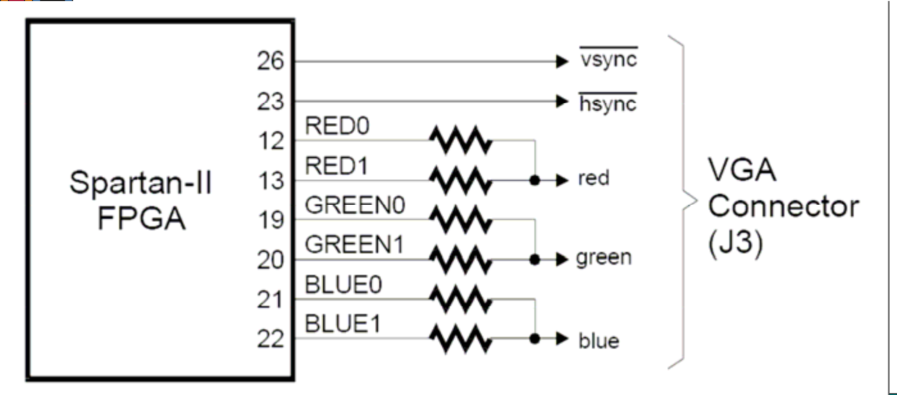


VGA Voltage Levels

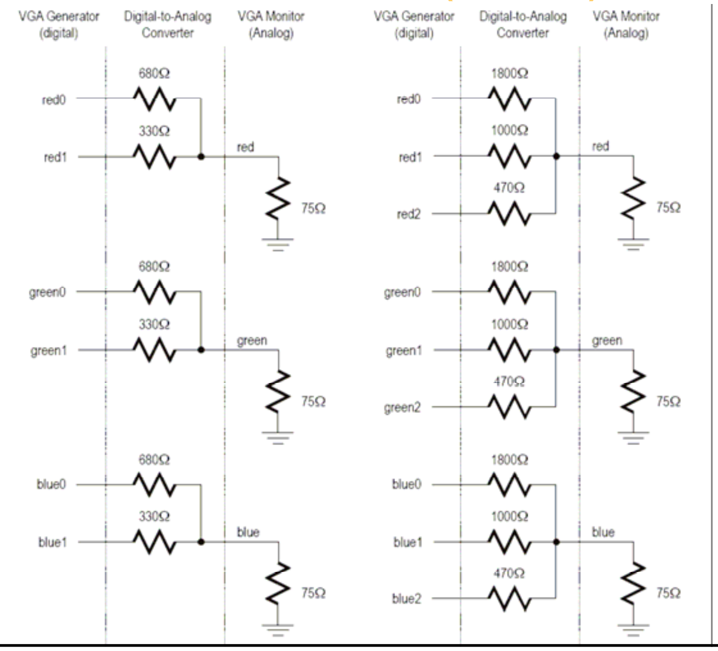
- Voltages on R, G, and B determine the color
  - Analog range from 0v (off) to +0.7v (on)
  - But, our pads produce 0-5v outputs!
  - For B&W output, just tie RGB together and let 0v=black and 5v=white
    - overdrives the input amps, but won't really hurt anything
  - For color you can drive R, G, B separately
    - Of course, this is only 8 colors (including black and white)
    - Requires storing three bits at each pixel locatio

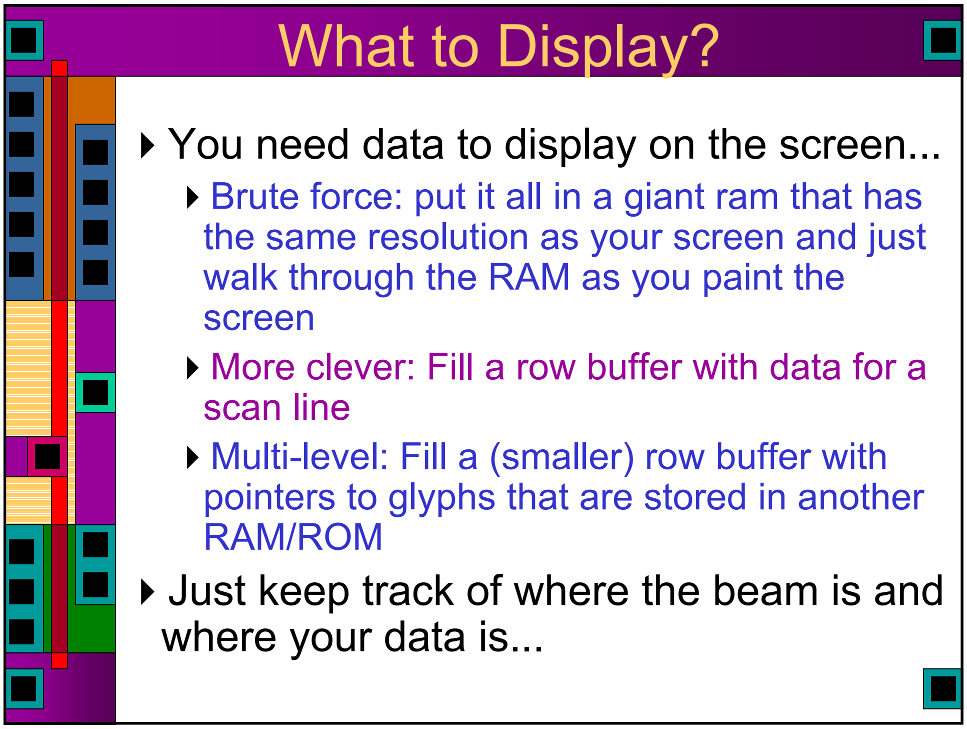
More colors

- More colors means more bits stored per pixel
- Also means D/A conversion to 0 to 0.7v range



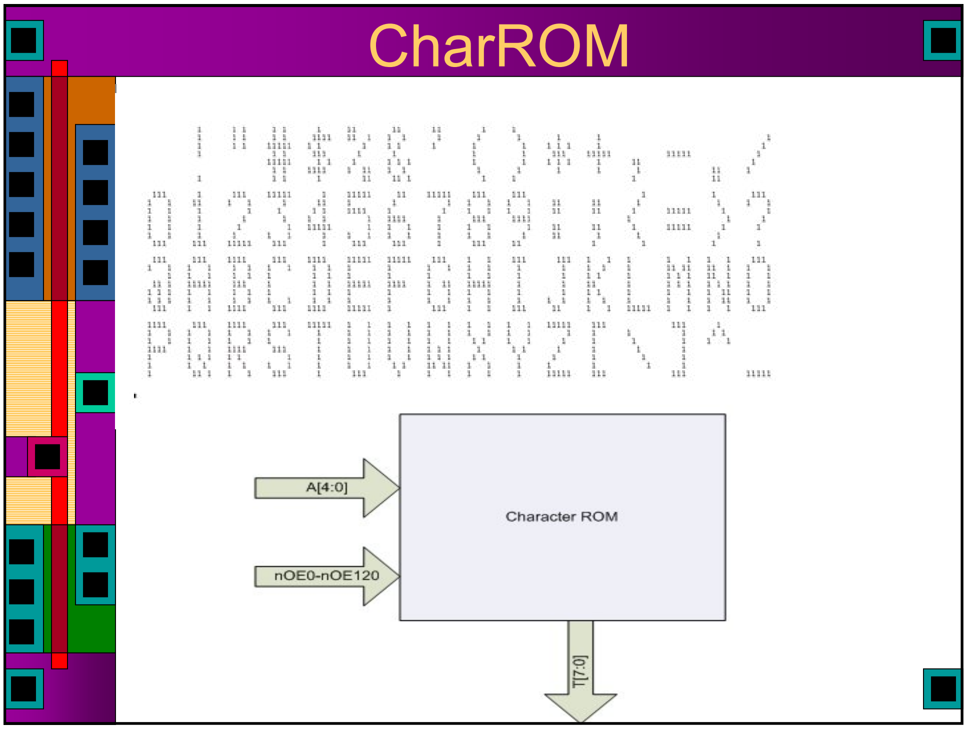
More Colors (Xess)



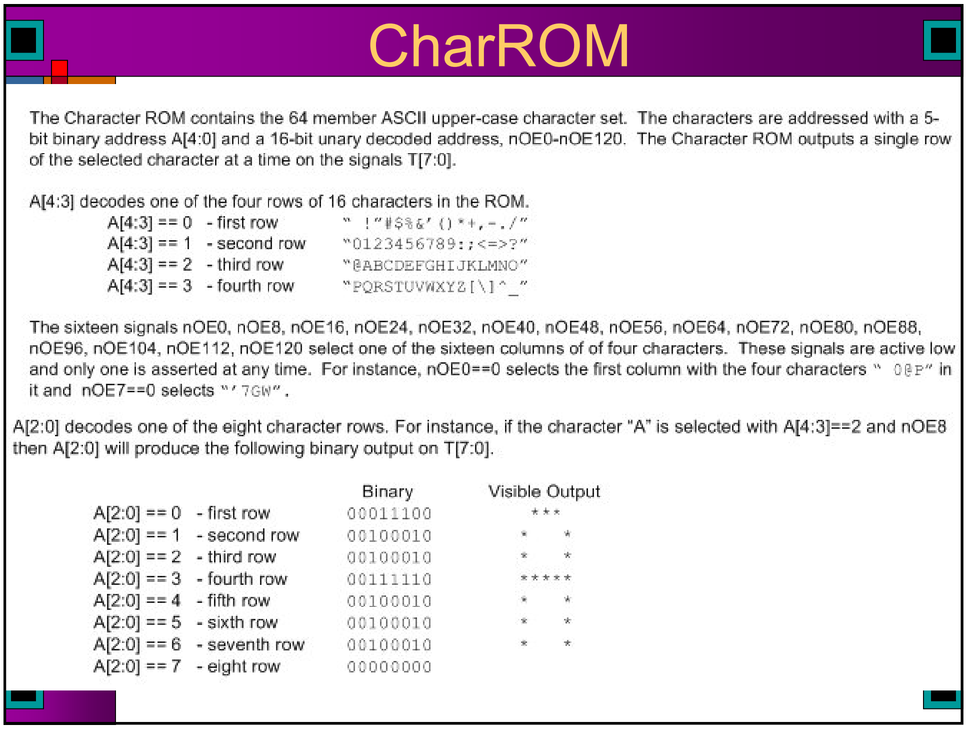


What to Display?

- You need data to display on the screen...
  - Brute force: put it all in a giant ram that has the same resolution as your screen and just walk through the RAM as you paint the screen
  - More clever: Fill a row buffer with data for a scan line
  - Multi-level: Fill a (smaller) row buffer with pointers to glyphs that are stored in another RAM/ROM
- Just keep track of where the beam is and where your data is...



CharROM



CharROM

The Character ROM contains the 64 member ASCII upper-case character set. The characters are addressed with a 5-bit binary address A[4:0] and a 16-bit unary decoded address, nOE0-nOE120. The Character ROM outputs a single row of the selected character at a time on the signals T[7:0].

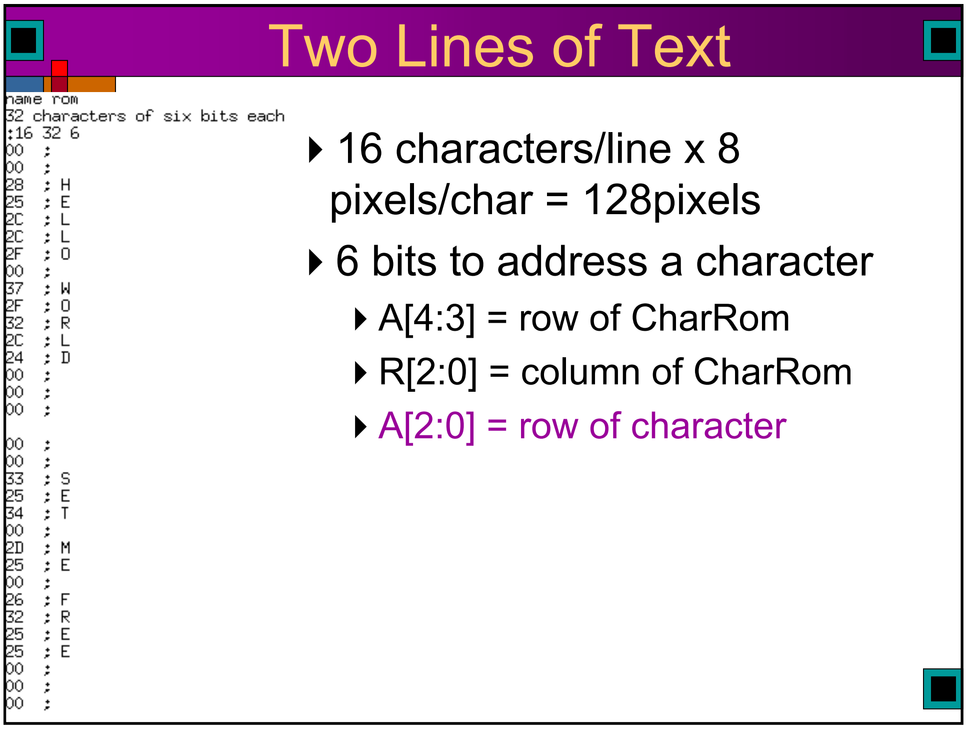
A[4:3] decodes one of the four rows of 16 characters in the ROM.

A[4:3] == 0	- first row	`" !"#$%&'()*+,-./"`
A[4:3] == 1	- second row	`"0123456789:;<=>?"`
A[4:3] == 2	- third row	`"@ABCDEFGHIJKLMNO"`
A[4:3] == 3	- fourth row	`"PQRSTUVWXYZ[\]^_"`

The sixteen signals nOE0, nOE8, nOE16, nOE24, nOE32, nOE40, nOE48, nOE56, nOE64, nOE72, nOE80, nOE88, nOE96, nOE104, nOE112, nOE120 select one of the sixteen columns of of four characters. These signals are active low and only one is asserted at any time. For instance, nOE0==0 selects the first column with the four characters " 0@P" in it and nOE7==0 selects "'7GW".

A[2:0] decodes one of the eight character rows. For instance, if the character "A" is selected with A[4:3]==2 and nOE8 then A[2:0] will produce the following binary output on T[7:0].

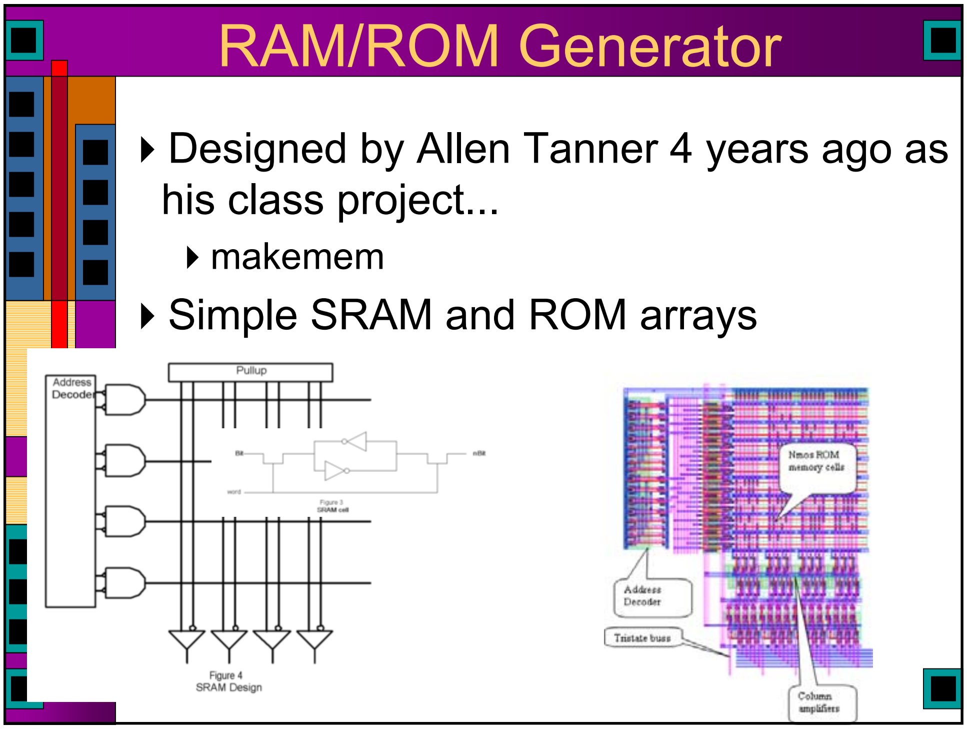
		Binary	Visible Output
A[2:0] == 0	- first row	00011100	`   ***`
A[2:0] == 1	- second row	00100010	`  *   *`
A[2:0] == 2	- third row	00100010	`  *   *`
A[2:0] == 3	- fourth row	00111110	`  *****`
A[2:0] == 4	- fifth row	00100010	`  *   *`
A[2:0] == 5	- sixth row	00100010	`  *   *`
A[2:0] == 6	- seventh row	00100010	`  *   *`
A[2:0] == 7	- eight row	00000000	



Two Lines of Text

```
name rom
32 characters of six bits each
:16 32 6
00  ;
00  ;
28  ; H
25  ; E
2C  ; L
2C  ; L
2F  ; O
00  ;
37  ; W
2F  ; O
32  ; R
2C  ; L
24  ; D
00  ;
00  ;
00  ;

00  ;
00  ;
33  ; S
25  ; E
34  ; T
00  ;
2D  ; M
25  ; E
00  ;
26  ; F
32  ; R
25  ; E
25  ; E
00  ;
00  ;
00  ;
```

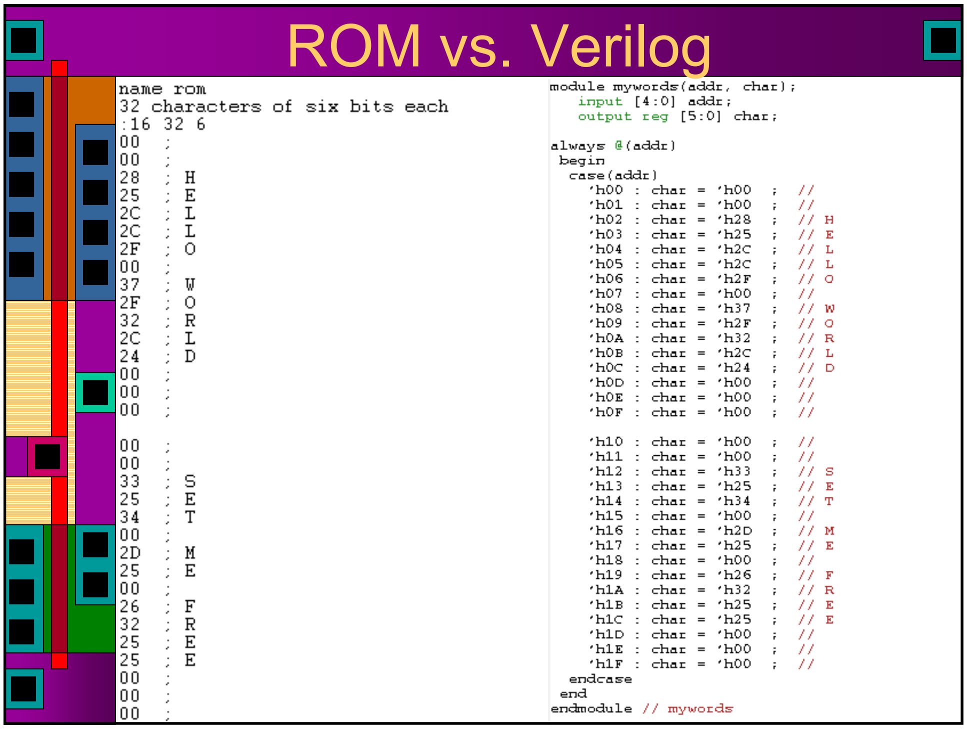
- 16 characters/line x 8 pixels/char = 128pixels
- 6 bits to address a character
  - A[4:3] = row of CharRom
  - R[2:0] = column of CharRom
  - A[2:0] = row of character



RAM/ROM Generator

- Designed by Allen Tanner 4 years ago as his class project...
  - makemem
- Simple SRAM and ROM arrays

Figure 4
SRAM Design

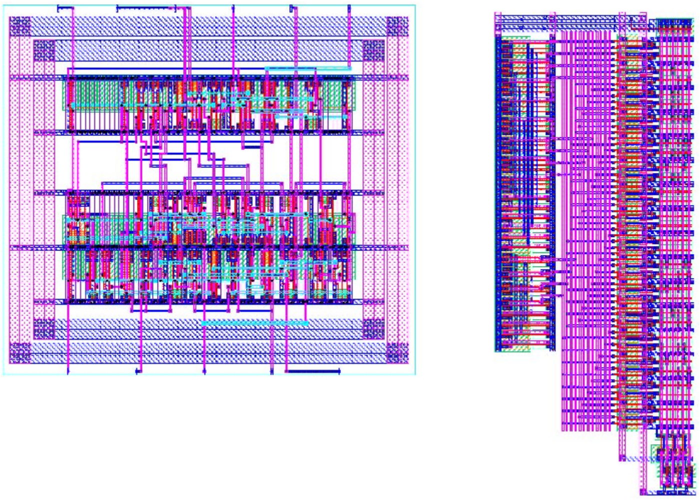


ROM vs. Verilog

```
name rom
32 characters of six bits each
:16 32 6
00  ;
00  ;
28  ;  H
25  ;  E
2C  ;  L
2C  ;  L
2F  ;  O
00  ;
37  ;  W
2F  ;  O
32  ;  R
2C  ;  L
24  ;  D
00  ;
00  ;
00  ;

00  ;
00  ;
33  ;  S
25  ;  E
34  ;  T
00  ;
2D  ;  M
25  ;  E
00  ;
26  ;  F
32  ;  R
25  ;  E
25  ;  E
00  ;
00  ;
00  ;
```

```
module mywords(addr, char);
    input [4:0] addr;
    output reg [5:0] char;

always @(addr)
 begin
  case(addr)
    'h00 : char = 'h00  ;  //
    'h01 : char = 'h00  ;  //
    'h02 : char = 'h28  ;  // H
    'h03 : char = 'h25  ;  // E
    'h04 : char = 'h2C  ;  // L
    'h05 : char = 'h2C  ;  // L
    'h06 : char = 'h2F  ;  // O
    'h07 : char = 'h00  ;  //
    'h08 : char = 'h37  ;  // W
    'h09 : char = 'h2F  ;  // O
    'h0A : char = 'h32  ;  // R
    'h0B : char = 'h2C  ;  // L
    'h0C : char = 'h24  ;  // D
    'h0D : char = 'h00  ;  //
    'h0E : char = 'h00  ;  //
    'h0F : char = 'h00  ;  //

    'h10 : char = 'h00  ;  //
    'h11 : char = 'h00  ;  //
    'h12 : char = 'h33  ;  // S
    'h13 : char = 'h25  ;  // E
    'h14 : char = 'h34  ;  // T
    'h15 : char = 'h00  ;  //
    'h16 : char = 'h2D  ;  // M
    'h17 : char = 'h25  ;  // E
    'h18 : char = 'h00  ;  //
    'h19 : char = 'h26  ;  // F
    'h1A : char = 'h32  ;  // R
    'h1B : char = 'h25  ;  // E
    'h1C : char = 'h25  ;  // E
    'h1D : char = 'h00  ;  //
    'h1E : char = 'h00  ;  //
    'h1F : char = 'h00  ;  //
  endcase
 end
endmodule // mywords
```

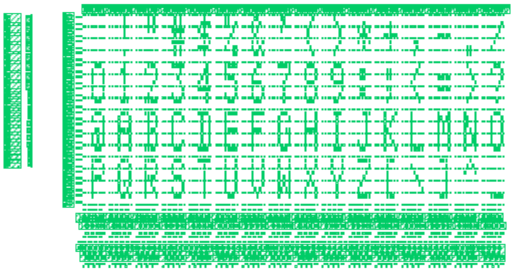
ROM vs. Verilog



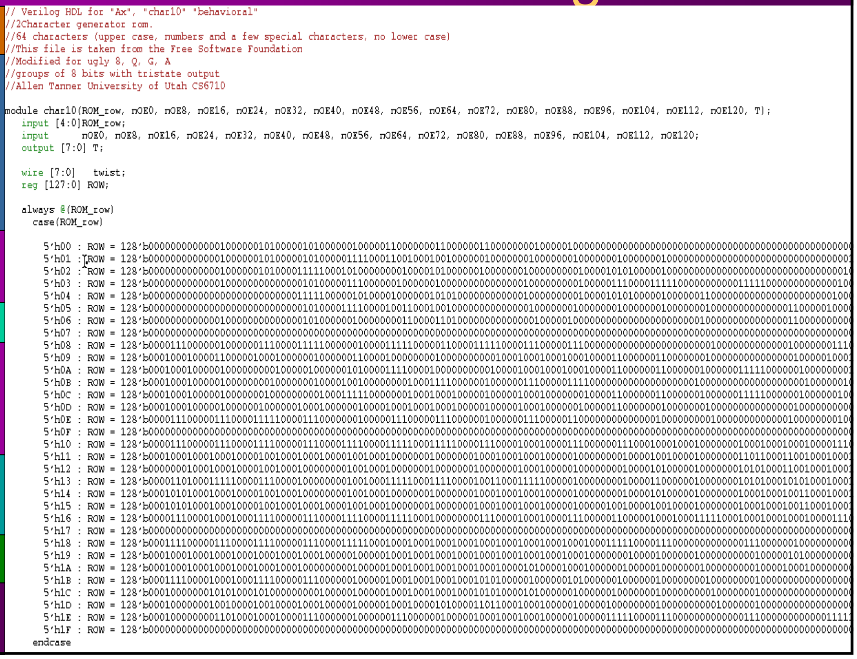
ROM vs. Verilog

```
Character generator rom.
64 characters (upper case, numbers and a few special characters, no lower case)
This file is taken from the Free Software Foundation
Modified for ugly 8, Q, G, A
groups of 8 bits with tristate output
Allen Tanner University of Utah CS6710
:2 32 128 8
00000000000001000000101000001010000001000001100000001100000011000000001000001000000000000000000000000000000000000000000000000000
00000000000001000000101000001010000011110001100100010010000001000000010000000100000001000000010000000000000000000000000000000001
00000000000001000000101000011111000101000000001000010100000010000000100000000010000101010000010000000000000000000000000000000010
00000000000001000000000000001010000011100000010000001000000000000000100000000010000011100001111100000000000111110000000000000100
00000000000000000000000000011111000001010000100000010101000000000000100000000010000101010000010000001100000000000000000000001000
00000000000000000000000000001010000111100001001100010010000000000000010000000100000001000000010000000100000000000000110000010000
00000000000001000000000000001010000001000000001100001101000000000000001000001000000000000000000000001000000000000000110000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00001110000001000000111000011111000000100001111100000110000111110000111000001110000000000000000000000010000000000000100000001110
00010001000011000001000100000010000001100001000000001000000000010001000100010001000011000000110000000100000000000000010000010001
00010001000001000000000100000100000010100001111000010000000000100001000100010001000011000000110000001000000111110000001000000001
00010001000001000000001000000010000100100000000100011110000001000000111000000111000000000000000000010000000000000000000100000010
00010001000001000000010000000001000111110000000100010001000010000001000100000001000011000000110000001000000111110000001000000100
00010001000001000000100000010001000000100001000100010001000010000001000100000010000011000000010000000100000000000000010000000000
00001110000011100001111100001110000000100000111000001110000010000000111000001100000000000000100000000010000000000000100000000100
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00001110000011100001111000001110000111100001111100011111000011100001000100001110000001110001000100010000000100010001000100001110
00010001000100010000100100010001000010010001000000010000000100010001000100000100000000100001001000010000000110110001100100010001
00000001000100010000100100010000000010010001000000010000000100000001000100000100000000100001010000010000000101010001100100010001
00001101000111110000111000010000000010010001111100011111000100110001111100000100000000100001100000010000000101010001010100010001
00010101000100010000100100010000000010010001000000010000000100010001000100000100000000100001010000010000000100010001001100010001
00010101000100010000100100010001000010010001000000010000000100010001000100000100000100100001001000010000000100010001001100010001
00001110000100010001111000001110000111100001111100010000000011100001000100001110000011000001000100011111000100010001000100001110
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00011110000011100001111000001110000111110001000100010001000100010001000100010001000111110000111000000000000011100000010000000000
00010001000100010001000100010001000001000001000100010001000100010001000100010001000000010000100000010000000000100000101000000000
00010001000100010001000100010000000001000001000100010001000100010000101000001010000000100000100000001000000000100001000100000000
00011110000100010001111000001110000001000001000100010001000101010000010000000100000001000000100000000100000000100000000000000000
00010000000101010001010000000001000001000001000100010001000101010000101000000100000010000000100000000010000000100000000000000000
00010000000100100001001000010001000001000001000100001010000101010001000100000100000100000000100000000001000000100000000000000000
00010000000011010001000100001110000001000000111000000100000010100001000100000100000111110000111000000000000011100000000000011111
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

ROM vs. Verilog

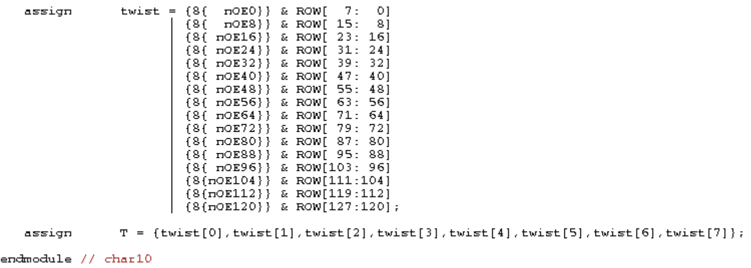


ROM vs. Verilog



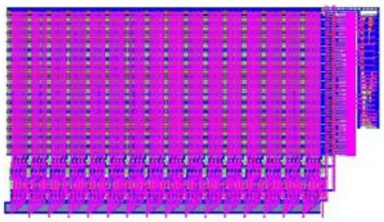
```
// Verilog HDL for "Ax", "char10" "behavioral"
//2Character generator rom.
//64 characters (upper case, numbers and a few special characters, no lower case)
//This file is taken from the Free Software Foundation
//Modified for ugly 8, Q, G, A
//groups of 8 bits with tristate output
//Allen Tanner University of Utah CS6710

module char10(ROM_row, nOE0, nOE8, nOE16, nOE24, nOE32, nOE40, nOE48, nOE56, nOE64, nOE72, nOE80, nOE88, nOE96, nOE104, nOE112, nOE120, T);
   input [4:0]ROM_row;
   input      nOE0, nOE8, nOE16, nOE24, nOE32, nOE40, nOE48, nOE56, nOE64, nOE72, nOE80, nOE88, nOE96, nOE104, nOE112, nOE120;
   output [7:0] T;

   wire [7:0]   twist;
   reg [127:0] ROW;

   always @(ROM_row)
    case(ROM_row)

       5'h00 : ROW = 128'b00000000000000100000010100000010100000001000001100000000110000001100000000100000100000000000000000000000000000000000000000000000
       5'h01 : ROW = 128'b0000000000000010000001010000001010000001111000110010001001000000010000000100000001000000010000000100000001000000000000000000000000000000001
       5'h02 : ROW = 128'b00000000000000100000010100001111100010100000000010000101000000100000001000000000100001010100000100000000000000000000000000000000000000010
       5'h03 : ROW = 128'b000000000000001000000000000000101000000111000000100000010000000000000010000000001000001110000111100000000000011110000000000000000100
       5'h04 : ROW = 128'b000000000000000000000000000011111000000101000010000000101010000000000001000000000100001010100000100000011000000000000000000000001000
       5'h05 : ROW = 128'b00000000000000000000000000001010000111100001001100010010000000000000000010000000100000001000000010000000100000000000000001100000100000
       5'h06 : ROW = 128'b00000000000000100000000000000010100000001000000001100001101000000000000000100000100000000000000000000000000000100000000000000000110000000
       5'h07 : ROW = 128'b00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
       5'h08 : ROW = 128'b0000111000000010000001110000111110000000100001111100000110000111110000111000001110000000000000000000000000100000000000000010000000111
       5'h09 : ROW = 128'b000100010000110000001000100000001000000011000100000000010000000000100010001000100010000110000001100000000100000000000000000100000100010
       5'h0A : ROW = 128'b0001000100000010000000001000000100000010100001110000100000000001000101000100010000110000001100000010000000111100000001000000001
       5'h0B : ROW = 128'b000100010000001000000001000000010000100100000000100011110000001000000001110000011110000000000000000000000100000000000000000000100000010
       5'h0C : ROW = 128'b0001000100000010000000100000000010001111100000001000100010000010000010100010000000100001100000011000000100000001111100000001000000010
       5'h0D : ROW = 128'b0001000100000010000001000000100010000000100001000100010001000001000001010001000000010000011000000010000000100000000000000001000000000
       5'h0E : ROW = 128'b00001110000011100001111100001110000000010000011100000111000000100000000111000001100000000000000010000000001000000000000010000000010
       5'h0F : ROW = 128'b00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
       5'h10 : ROW = 128'b000011100000111000011110000011100001111000011111000111110000111000010001000011100000011100010001000100000000100010001000100001110
       5'h11 : ROW = 128'b00010001000100010000100100010001000010010001000000010000000100010001000100000100000000100001001000010000000110110001100100010001
       5'h12 : ROW = 128'b000000010001000100001001000100000000010010001000000010000000100000001000100000100000000100001010000010000000101010001100100010001
       5'h13 : ROW = 128'b00001101000111110000111000010000000010010001111100011110000100110001111100000100000000100001100000010000000101010001010100010001
       5'h14 : ROW = 128'b0001010100010001000010010001000000000100100010000000100000001000100010001000001000000001000010100000100000001000100010011000100 01
       5'h15 : ROW = 128'b000101010001000100001001000100010000100100010000000100000001000100010001000001000001001000010010000100000001000100010011000100 01
       5'h16 : ROW = 128'b00001110000100010001111000001110000111100001111100010000000011100001000100001110000011000001000100011110001000100010001000011 10
       5'h17 : ROW = 128'b00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
       5'h18 : ROW = 128'b0001111000001110000111100000111000011111000100010001000100010001000100010001000111110000111000000000000011100000010000000000
       5'h19 : ROW = 128'b00010001000100010001000100010001000001000001000100010001000100010001000100010000000010001000000100000000001000001010000000
       5'h1A : ROW = 128'b0001000100010001000100010001000000000100000100010001000100010001000010100001000100000001000001000000010000000001000100010000000
       5'h1B : ROW = 128'b0001111000010001000111100000111000000010000010001000100010001010100000010000001010000001000000100000000100000000100000000000000000
       5'h1C : ROW = 128'b000100000000101010001010000000000100000100000100010001000100010101000010100000010000001000000010000000001000000010000000000000000
       5'h1D : ROW = 128'b0001000000001001000010010000100010000010000010001000010100001101100010001000001000001000000001000000000010000001000000000000000
       5'h1E : ROW = 128'b0001000000000110100010001000011100000010000001110000001000001000100010001000001000001111000011100000000000001110000000000000 1111
       5'h1F : ROW = 128'b00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    endcase
```

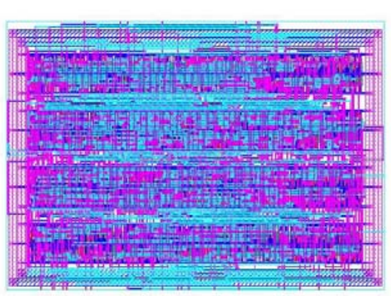
ROM vs. Verilog

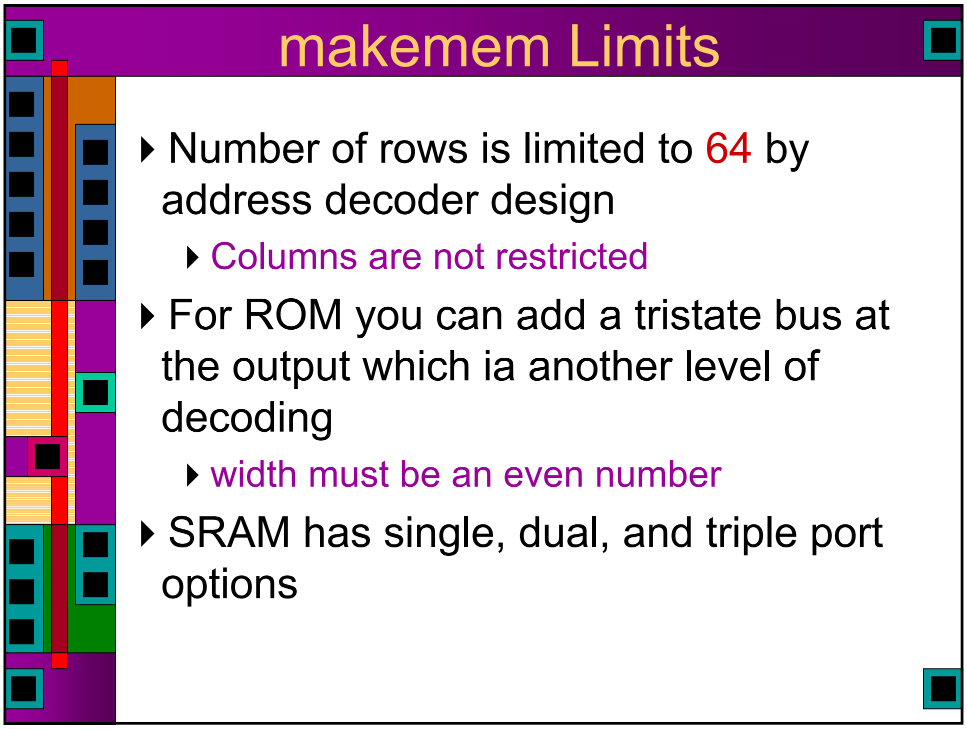


```
  assign      twist = {8{  nOE0}} & ROW[  7:  0]
                    | {8{  nOE8}} & ROW[ 15:  8]
                    | {8{ nOE16}} & ROW[ 23: 16]
                    | {8{ nOE24}} & ROW[ 31: 24]
                    | {8{ nOE32}} & ROW[ 39: 32]
                    | {8{ nOE40}} & ROW[ 47: 40]
                    | {8{ nOE48}} & ROW[ 55: 48]
                    | {8{ nOE56}} & ROW[ 63: 56]
                    | {8{ nOE64}} & ROW[ 71: 64]
                    | {8{ nOE72}} & ROW[ 79: 72]
                    | {8{ nOE80}} & ROW[ 87: 80]
                    | {8{ nOE88}} & ROW[ 95: 88]
                    | {8{ nOE96}} & ROW[103: 96]
                    | {8{nOE104}} & ROW[111:104]
                    | {8{nOE112}} & ROW[119:112]
                    | {8{nOE120}} & ROW[127:120];

  assign      T = {twist[0],twist[1],twist[2],twist[3],twist[4],twist[5],twist[6],twist[7]};

endmodule // char10
```

ROM vs. Verilog

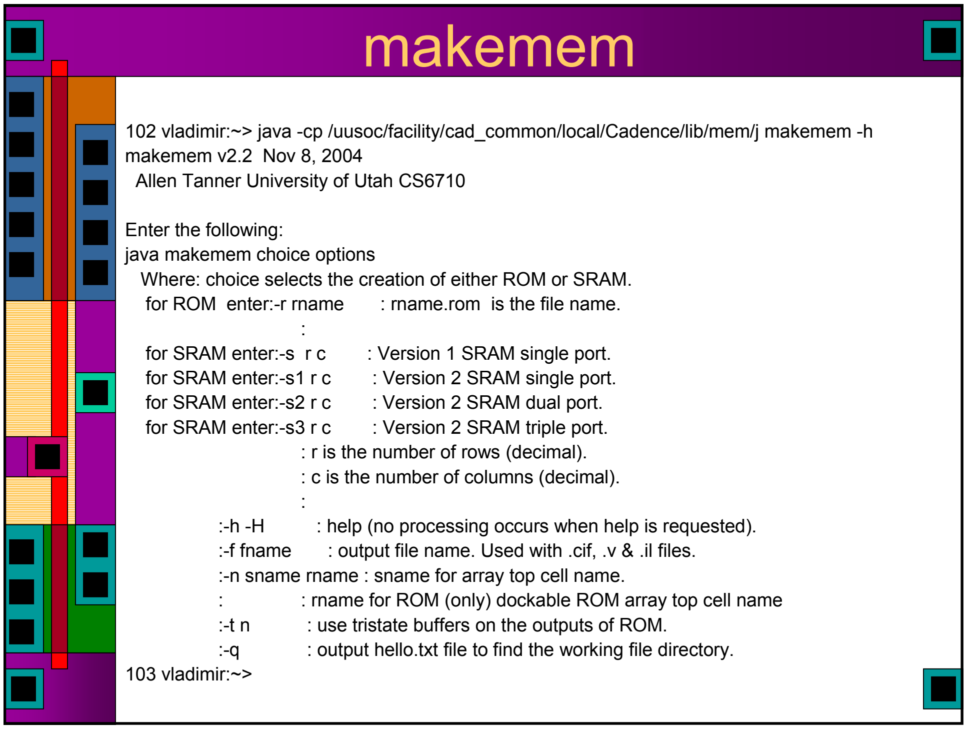






makemem Limits

- Number of rows is limited to 64 by address decoder design
  - Columns are not restricted
- For ROM you can add a tristate bus at the output which ia another level of decoding
  - width must be an even number
- SRAM has single, dual, and triple port options



makemem

```
102 vladimir:~> java -cp /uusoc/facility/cad_common/local/Cadence/lib/mem/j makemem -h
makemem v2.2  Nov 8, 2004
  Allen Tanner University of Utah CS6710

Enter the following:
java makemem choice options
   Where: choice selects the creation of either ROM or SRAM.
    for ROM  enter:-r rname       : rname.rom  is the file name.
                                  :
    for SRAM enter:-s  r c       : Version 1 SRAM single port.
    for SRAM enter:-s1 r c       : Version 2 SRAM single port.
    for SRAM enter:-s2 r c       : Version 2 SRAM dual port.
    for SRAM enter:-s3 r c       : Version 2 SRAM triple port.
                                  : r is the number of rows (decimal).
                                  : c is the number of columns (decimal).
                                  :
                  :-h -H          : help (no processing occurs when help is requested).
                  :-f fname       : output file name. Used with .cif, .v & .il files.
                  :-n sname rname : sname for array top cell name.
                  :               : rname for ROM (only) dockable ROM array top cell name
                  :-t n           : use tristate buffers on the outputs of ROM.
                  :-q             : output hello.txt file to find the working file directory.
103 vladimir:~>
```