Display Technology

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

Cathode Ray Tube

- Cathode Ray Tube
  - Anode
  - Cathode
  - High Voltage
  - Fluorescent Screen
Cathode Ray Tube

Raster Scanning
Color

Shadow Mask and Aperture Grille
Liquid Crystal Displays

Diagram showing the components of a liquid crystal display, including lightwave, polarized panels, electrode, liquid crystals, glass plates, vertical filter, crystal molecule, horizontal filter, and colour filter.
DLP Projector

Liquid Crystal on Silicon (LCoS)

- Put a liquid crystal between a reflective layer on a silicon chip
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 goes where
- Scan it with a laser for high light output
- 4000 pixel wide frame at 60Hz
Digistar 3 Dome Projector

- 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  - Now superseded by much higher resolution standards...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  
2: Green out  
3: Blue out  
4: Unused  
5: Ground

6: Red return (ground)  
7: Green return (ground)  
8: Blue return (ground)  
9: Unused  
10: Sync return (ground)

11: Monitor ID 0 in  
12: Monitor ID 1 in or data from display  
13: Horizontal Sync  
14: Vertical Sync  
15: Monitor ID 3 in or data clock

Raster Scanning
Raster Scanning

VGA Horizontal Timing

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal Dots</td>
<td>640</td>
</tr>
<tr>
<td>Vertical Scan Lines</td>
<td>480</td>
</tr>
<tr>
<td>Horiz. Sync Polarity</td>
<td>NEG</td>
</tr>
<tr>
<td>A (µs)</td>
<td>31.77</td>
</tr>
<tr>
<td>B (µs)</td>
<td>3.77</td>
</tr>
<tr>
<td>C (µs)</td>
<td>1.89</td>
</tr>
<tr>
<td>D (µs)</td>
<td>25.17</td>
</tr>
<tr>
<td>E (µs)</td>
<td>0.94</td>
</tr>
</tbody>
</table>

60Hz vertical frequency

640 pixels are displayed each time the beam traverses the screen

Stable current ramp: Information is displayed during this time

Total horizontal time

Retrace: No information is displayed during this time

Back porch

Front porch

HS

Horizontal sync signal sets the retrace frequency
### VGA Horizontal Timing

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal Dots</td>
<td>640</td>
</tr>
<tr>
<td>Vertical Scan Lines</td>
<td>480</td>
</tr>
<tr>
<td>Horiz. Sync Polarity</td>
<td>NEG</td>
</tr>
<tr>
<td>A (µs)</td>
<td>31.77</td>
</tr>
<tr>
<td>B (µs)</td>
<td>3.77</td>
</tr>
<tr>
<td>C (µs)</td>
<td>1.89</td>
</tr>
<tr>
<td>D (µs)</td>
<td>25.17</td>
</tr>
<tr>
<td>E (µs)</td>
<td>0.94</td>
</tr>
</tbody>
</table>

\[
\frac{25.17}{640} \times 10^6 = 39.33 \text{ns/pixel} = 25.4 \text{MHz pixel clock}
\]

### VGA Vertical Timing

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal Dots</td>
<td>640</td>
</tr>
<tr>
<td>Vertical Scan Lines</td>
<td>480</td>
</tr>
<tr>
<td>Vert. Sync Polarity</td>
<td>NEG</td>
</tr>
<tr>
<td>Vertical Frequency</td>
<td>60Hz</td>
</tr>
<tr>
<td>O (ms)</td>
<td>16.68</td>
</tr>
<tr>
<td>P (ms)</td>
<td>0.06</td>
</tr>
<tr>
<td>Q (ms)</td>
<td>1.02</td>
</tr>
<tr>
<td>R (ms)</td>
<td>15.25</td>
</tr>
<tr>
<td>S (ms)</td>
<td>0.35</td>
</tr>
</tbody>
</table>

\[
\frac{16.68}{480} \times 10^6 = 34.75 \text{ns/line} = 71.7 \text{MHz line clock}
\]
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 Horizontal Timing

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal Dots</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>Vertical Scan Lines</td>
<td>?</td>
<td>60Hz vertical frequency</td>
</tr>
<tr>
<td>Horiz. Sync Polarity</td>
<td>NEG</td>
<td></td>
</tr>
<tr>
<td>A (µs)</td>
<td>30.0</td>
<td>Scanline time</td>
</tr>
<tr>
<td>B (µs)</td>
<td>2.0</td>
<td>Sync pulse length</td>
</tr>
<tr>
<td>C (µs)</td>
<td>10.7</td>
<td>Back porch</td>
</tr>
<tr>
<td>D (µs)</td>
<td>12.8</td>
<td>Active video time</td>
</tr>
<tr>
<td>E (µs)</td>
<td>4.50</td>
<td>Front porch</td>
</tr>
</tbody>
</table>

\[
12.8/128 = 100\text{ns/pixel} = 10 \text{ MHz pixel clock}
\]

---

### VGA Relaxed Vertical Timing

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal Dots</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>Vertical Scan Lines</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>Vert. Sync Polarity</td>
<td>NEG</td>
<td></td>
</tr>
<tr>
<td>Vertical Frequency</td>
<td>60Hz</td>
<td></td>
</tr>
<tr>
<td>O (ms)</td>
<td>16.68</td>
<td>Total frame time</td>
</tr>
<tr>
<td>P (ms)</td>
<td>0.09</td>
<td>Sync pulse length (3x30µs)</td>
</tr>
<tr>
<td>Q (ms)</td>
<td>4.86</td>
<td>Back porch</td>
</tr>
<tr>
<td>R (ms)</td>
<td>7.65</td>
<td>Active video time</td>
</tr>
<tr>
<td>S (ms)</td>
<td>4.08</td>
<td>Front porch</td>
</tr>
</tbody>
</table>

---
VGA on Spartan3e Starter

Series resistors limit output voltage to 0-0.7v

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
    - This 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 location

VGA on Spartan3e Starter

<table>
<thead>
<tr>
<th>VGA_RED</th>
<th>VGA_GREEN</th>
<th>VGA_BLUE</th>
<th>Resulting Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Black</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Blue</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Green</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Cyan</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Red</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Magenta</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Yellow</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>White</td>
</tr>
</tbody>
</table>
More colors

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

VGA Breakdown

- vgaControl
  - Generate timing pulses at the right time
    - hSync, vSync, bright, hCount, vCount

- bitGen
  - Based on bright, hCount, vCount, turn on the bits
3 Types of bitGen

- Bitmapped
- Character/Glyph – based
- Hard-coded

Bitmapped
- Frame buffer holds a separate rgb color for every pixel
- bitGen just grabs the pixel based on hCount and vCount and splats it to the screen
- Chews up a LOT of memory
- This memory would have to be off-chip…
3 Types of bitGen

- Character/Glyph-based
  - Break screen into n×m pixel chunks (e.g. 8x8)
  - For each chunk, point to one of k n×m glyphs
  - Those glyphs are stored in a separate memory
  - For 8x8 case (for example)
    - glyph number is hCount and vCount minus the low three bits
    - glyph bits are the low-order 3 bits in each of hCount and vCount
    - Figure out which screen chunk you’re in, then reference the bits from the glyph memory

3 Types of bitGen

- Direct Graphics
  - Look at hCount and vCount to see where you are on the screen
  - Depending on where you are, force the output to a particular color
  - Tedious for complex things, nice for large, static things

```parameter BLACK = 3'b000, WHITE = 3'b111, RED = 3'b100;
// paint a white box on a red background
always@(*)
if (~bright) rgb = BLACK; // force black if not bright
// check to see if you’re in the box
else if (((hCount >= 100) && (hCount <= 300)) &&
          ((vCount >= 150) && (vCount <= 350))) rgb = WHITE;
else rgb = RED; // background color```
VGA Memory Requirements

- 640x480 VGA (bitmapped)
  - 307,200 pixels
  - 3 bits per pixel
  - Imagine using 24 bits per memory location (8 pixels)
  - 38.4 K-words with 24-bit words for 640x480
    - 115.2 K-bytes
  - FAR larger than you can put on your chip…
  - Not so bad with an off-chip RAM

- 320x240 VGA (bitmapped)
  - 76,800 pixels
  - Each stored pixel is 2x2 screen pixels
  - 3 bits per pixel
  - 8 pixels per 24-bit word (for example)
  - 9.6k 24-bit words needed
    - 28.8 K-bytes
  - Much more realistic…but still significant memory if you want to put it on-chip
VGA Memory Requirements

- 80 char by 60 line display (8x8 glyphs)
  - 4800 locations
  - Each location has one of 256 char/glyphs
  - 8-bits per location
    - 2 locations per 16-bit word?
    - 2400 words for the frame buffer
  - Each char/glyph is (say) 8x8 pixels
    - results in 640x480 display…
  - 8x8x256 bits for char/glyph table
    - 16kbits (1k words) for char/glyph table
    - Will this fit on your chip?

- 80 char by 60 line display (8x8 glyphs)
  - 4800 locations
  - Each location has one of 64 char/glyphs
  - 6-bits per location
    - 4 locations per 24-bit word?
    - 1200 words for frame buffer?
  - Each char/glyph is (say) 8x8 pixels
    - results in 640x480 display…
  - 8x8x64 bits for char/glyph table
    - 4kbits for char/glyph table (32 words, 128 b/word)
    - Will this fit on your chip?
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
- $A[4:3] = 2$ - third row
- $A[4:3] = 3$ - fourth row

The sixteen signals $nOE0$, $nOE9$, $nOE18$, $nOE24$, $nOE32$, $nOE40$, $nOE48$, $nOE56$, $nOE64$, $nOE72$, $nOE80$, $nOE88$, $nOE96$, $nOE104$, $nOE112$, $nOE120$ select one of the sixteen columns of four characters. These signals are active low and only one is asserted at any time. For instance, $nOE8=0$ selects the first column with the four characters " " in it and $nOE7=0$ selects " " in.

$A[2:0]$ 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 $T[7:0]$.

<table>
<thead>
<tr>
<th>Binary</th>
<th>Visible Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00011110</td>
<td>***</td>
</tr>
<tr>
<td>00100010</td>
<td>*</td>
</tr>
<tr>
<td>00100010</td>
<td>*</td>
</tr>
<tr>
<td>00111110</td>
<td>***</td>
</tr>
<tr>
<td>00100010</td>
<td>*</td>
</tr>
<tr>
<td>00100010</td>
<td>*</td>
</tr>
<tr>
<td>00100010</td>
<td>*</td>
</tr>
<tr>
<td>00000000</td>
<td>**</td>
</tr>
</tbody>
</table>
CharROM

Fit the charROM into a VGA system
- hVideo walks along the row
- vVideo picks which row to walk along

Character Function…
- … i.e. Frame Buffer
- 16 characters/line x 8 pixels/char = 128 pixels
- 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 8 years ago as his class project...

- makemem

Simple ROM arrays

(Don’t use the SRAM)

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 name : sname 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:~>
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 is another level of decoding
  - width must be an even number
- SRAM has single, dual, and triple port options
  - But, fabricated versions are very uneven…

ROM vs. Verilog
ROM vs. Verilog

```verilog
assign twist = ((n080) & ROM11:0);
assign twist = ((n088) & ROM15:4);
assign twist = ((n08c) & ROM11:10);
assign twist = ((n098) & ROM11:24);
assign twist = ((n09e) & ROM19:11);
assign twist = ((n0a0) & ROM47:40);
assign twist = ((n0a8) & ROM51:44);
assign twist = ((n0ac) & ROM55:48);
assign twist = ((n0b0) & ROM59:52);
assign twist = ((n0be) & ROM67:60);
assign twist = ((n0c0) & ROM105:96);
assign twist = ((n0c4) & ROM111:104);
assign twist = ((n0c8) & ROM115:112);
assign twist = ((n0d0) & ROM117:120);

module char10;

```

ROM vs. Verilog
ROM size comparison

Makemem also generates SRAM
- Three different variants: single, dual, triple port
- Each port is independent R/W
- But, no automatic arbitration, so make sure you're not using the same address on multiple ports

BUT! It's not working well
Use memCellsF09 instead!!!
module regfile #(parameter WIDTH = 8, REGBITS = 3)
  (input                clk, regwrite,
   input  [REGBITS-1:0] ra1, ra2, wa,
   input  [WIDTH-1:0]   wd,
   output [WIDTH-1:0]   rd1, rd2);
  reg  [WIDTH-1:0] RAM [(1<<REGBITS)-1:0];

  // read two ports (combinational)
  // write third port on rising edge of clock
  always @(posedge clk)
    if (regwrite) RAM[wa] <= wd;

  assign rd1 =  RAM[ra1];
  assign rd2 =  RAM[ra2];
endmodule

module SRAM #(parameter WIDTH = 8, REGBITS = 3)
  (input                clk, WE,
   input  [REGBITS-1:0] addr,
   input  [WIDTH-1:0]   wd,
   output [WIDTH-1:0]   data);
  reg  [WIDTH-1:0] RAM [(1<<REGBITS)-1:0];

  // on clk, write if WE is high
  always @(posedge clk)
    if (WE) RAM[addr] <= wd;

  // Read asynchronously from addr
  assign data =  RAM[addr];
endmodule
SRAM Cell, Transistors

6-transistor SRAM Cell

Tricky to get this right!

Multi-Port Register

Register file cell with single-ended read – makes a great register file
Register File

- Slightly larger cell, but with single-ended read – makes a great register file

SRAM Cell

Yet another cell – differential write, single-ended read
Array-Structured Memory

- Row Decoders
  - Select exactly one of the memory rows
  - Simple versions are just gates
Multiple levels of decoding can be more efficient layout.

Other circuit tricks for building row decoders...
Single-Port SRAM

module SRAM2 #(parameter WIDTH = 8, REGBITS = 3)
    (input    clk, WE,
     input [REGBITS-1:0] addr, raddr,
     input [WIDTH-1:0]  wd,
     output [WIDTH-1:0] data, rdata);

    reg [WIDTH-1:0] RAM [(1<<REGBITS)-1:0];

    // on clk, write if WE is high
    always @(posedge clk)
        if (WE) RAM[addr] <= wd;

    // Read asynchronously from addr & raddr
    assign data = RAM[addr];
    assign rdata = RAM[raddr];
endmodule

Two-Port SRAM/FF

module SRAM2 #(parameter WIDTH = 8, REGBITS = 3)
    (input    clk, WE,
     input [REGBITS-1:0] addr, raddr,
     input [WIDTH-1:0]  wd,
     output [WIDTH-1:0] data, rdata);

    reg [WIDTH-1:0] RAM [(1<<REGBITS)-1:0];

    // on clk, write if WE is high
    always @(posedge clk)
        if (WE) RAM[addr] <= wd;

    // Read asynchronously from addr & raddr
    assign data = RAM[addr];
    assign rdata = RAM[raddr];
endmodule
Two-Port SRAM/FF

Two-Port SRAM
Conclusions

- Try out memCellsF09 for SRAM
  - Details on the class web page
  - But, as you can see, you can’t fit much on a chip

- ROMs are very useful for tables of data
  - I’d use Verilog case-statements…

- If you’re using VGA
  - Check out the mini-project from 2005
  - Again, on the class website