Display Technology

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

Cathode Ray Tube

- Anode
- Cathode
- High Voltage
- Fluorescent Screen

(C) COSMOS 2002
Cathode Ray Tube

Raster Scanning
Electron Gun

Beam Steering Coils
Color

Shadow Mask and Aperture Grille
Liquid Crystal on Silicon

- 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 where
- Scan it with a laser for high light output
- 4000 pixel wide frame ever 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  - Now superseded by much higher resolution standards...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**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red out</td>
</tr>
<tr>
<td>2</td>
<td>Green out</td>
</tr>
<tr>
<td>3</td>
<td>Blue out</td>
</tr>
<tr>
<td>4</td>
<td>Unused</td>
</tr>
<tr>
<td>5</td>
<td>Ground</td>
</tr>
<tr>
<td>6</td>
<td>Red return (ground)</td>
</tr>
<tr>
<td>7</td>
<td>Green return (ground)</td>
</tr>
<tr>
<td>8</td>
<td>Blue return (ground)</td>
</tr>
<tr>
<td>9</td>
<td>Unused</td>
</tr>
<tr>
<td>10</td>
<td>Sync return (ground)</td>
</tr>
<tr>
<td>11</td>
<td>Monitor ID 0 in</td>
</tr>
<tr>
<td>12</td>
<td>Monitor ID 1 in or data from display</td>
</tr>
<tr>
<td>13</td>
<td>Horizontal Sync</td>
</tr>
<tr>
<td>14</td>
<td>Vertical Sync</td>
</tr>
<tr>
<td>15</td>
<td>Monitor ID 3 in or data clock</td>
</tr>
</tbody>
</table>

**Raster Scanning**

Diagram showing raster scanning pattern.
<table>
<thead>
<tr>
<th>VGA Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Horizontal Dots</strong></td>
</tr>
<tr>
<td><strong>Vertical Scan Lines</strong></td>
</tr>
<tr>
<td><strong>Horiz. Sync Polarity</strong></td>
</tr>
<tr>
<td><strong>A (µs)</strong></td>
</tr>
<tr>
<td><strong>B (µs)</strong></td>
</tr>
<tr>
<td><strong>C (µs)</strong></td>
</tr>
<tr>
<td><strong>D (µs)</strong></td>
</tr>
<tr>
<td><strong>E (µs)</strong></td>
</tr>
</tbody>
</table>

![VGA Timing Diagram]

\[
\frac{25.17}{640} = 39.33\text{ns/pixel} = 25.4\text{MHz pixel clock}
\]
VGA Timing

- Horizontal 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)

VGA Timing Summary

- 60 Hz refresh and 25MHz pixel clock
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

- **Horizontal Dots**: 128
- **Vertical Scan Lines**: ?
- **Horiz. Sync Polarity**: NEG
- **A (µs)**: 30.0
- **B (µs)**: 2.0
- **C (µs)**: 10.7
- **D (µs)**: 12.8
- **E (µs)**: 4.50

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

- Horizontal 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

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 location.

VGA on Spartan3e Starter

Series resistors limit output voltage to 0-0.7v

Figure 6-1: VGA Connections from Spartan-3E Starter Kit Board
### VGA on Spartan3e Starter

#### Table 6-1: 3-Bit Display Color Codes

<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>

#### Raster Scanning

![Raster Scanning Diagram](image)
### More colors

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

### VGA on Spartan3e Starter

#### Table 6-2: 640x480 Mode VGA Timing

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Vertical Sync</th>
<th>Horizontal Sync</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Clocks</td>
<td>Lines</td>
</tr>
<tr>
<td>$T_g$</td>
<td>Sync pulse time</td>
<td>16.7 ms</td>
<td>416,800</td>
</tr>
<tr>
<td>$T_{disp}$</td>
<td>Display time</td>
<td>15.36 ms</td>
<td>384,000</td>
</tr>
<tr>
<td>$T_{pw}$</td>
<td>Pulse width</td>
<td>64 μs</td>
<td>1,600</td>
</tr>
<tr>
<td>$T_{fp}$</td>
<td>Front porch</td>
<td>320 μs</td>
<td>8,000</td>
</tr>
<tr>
<td>$T_{bp}$</td>
<td>Back porch</td>
<td>928 μs</td>
<td>25,280</td>
</tr>
</tbody>
</table>

#### Figure 6-3: VGA Control Timing

The signal timings in Table 6-2 are derived for a 640-pixel by 480-row display using a 25 MHz pixel clock and 60 Hz a 1 refresh. Figure 6-3 shows the relation between each of the timing symbols. The timing for the sync pulse width ($T_{pw}$) and front and back porch intervals ($T_{fp}$ and $T_{bp}$) are based on observations from various VGA displays. The front and back porch intervals are the pre- and post-sync pulse times. Information cannot be displayed during these times.
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**
  - 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 nxm pixel chunks (e.g. 8x8)
  - For each chunk, point to one of k nxm 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

- 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

```verilog
parameter BLACK = 3'b 000, 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
  - 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
  - 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 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 “*#%$&’()**+,-./”

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 four characters. These signals are active low and only one is asserted at any time. For instance, $nOE0$ selects the first column with the four characters “*#%$” in it and $nOE7=0$ selects “\]_}"".

$A[2:0]$ 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></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0011100</td>
<td>***</td>
</tr>
<tr>
<td>001</td>
<td>00100010</td>
<td>*</td>
</tr>
<tr>
<td>010</td>
<td>00100010</td>
<td>*</td>
</tr>
<tr>
<td>011</td>
<td>00111110</td>
<td>*****</td>
</tr>
<tr>
<td>100</td>
<td>00100010</td>
<td>*</td>
</tr>
<tr>
<td>101</td>
<td>00100010</td>
<td>*</td>
</tr>
<tr>
<td>110</td>
<td>00100010</td>
<td>*</td>
</tr>
<tr>
<td>111</td>
<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

Two Lines of Text
- Character Function...
- 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 6 years ago as his class project...

- makemem

Simple ROM arrays (Don’t use the SRAM)

```
102 vladimir:~> java -cp /uusoc/facility/cad_common/local/Cadence/lib/mem/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.
      :-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

```verilog
module myword(input [4:0] addr);
output reg [5:0] char;
always @ (addr)
begin
  case(addr)
    '100 : char = 'h00 ; //
    '101 : char = 'h01 ; //
    '102 : char = 'h02 ; //
    '103 : char = 'h03 ; //
    '104 : char = 'h04 ; // L
    '105 : char = 'h05 ; // L
    '106 : char = 'h06 ; // O
    '107 : char = 'h07 ; //
    '108 : char = 'h08 ; //
    '109 : char = 'h09 ; //
    '110 : char = 'h0a ; //
    '111 : char = 'h0b ; //
    '112 : char = 'h0c ; //
    '113 : char = 'h0d ; //
    '114 : char = 'h0e ; //
    '115 : char = 'h0f ; //
    '116 : char = 'h10 ; //
    '117 : char = 'h11 ; //
    '118 : char = 'h12 ; //
    '119 : char = 'h13 ; //
    '11a : char = 'h14 ; //
    '11b : char = 'h15 ; //
    '11c : char = 'h16 ; //
    '11d : char = 'h17 ; //
    '11e : char = 'h18 ; //
    '11f : char = 'h19 ; //
    '120 : char = 'h1a ; //
    '121 : char = 'h1b ; //
    '122 : char = 'h1c ; //
    '123 : char = 'h1d ; //
    '124 : char = 'h1e ; //
    '125 : char = 'h1f ; //
    '126 : char = 'h20 ; //
    '127 : char = 'h21 ; //
    '128 : char = 'h22 ; //
    '129 : char = 'h23 ; //
    '12a : char = 'h24 ; //
    '12b : char = 'h25 ; //
    '12c : char = 'h26 ; //
    '12d : char = 'h27 ; //
    '12e : char = 'h28 ; //
    '12f : char = 'h29 ; //
    '130 : char = 'h2a ; //
    '131 : char = 'h2b ; //
    '132 : char = 'h2c ; //
    '133 : char = 'h2d ; //
    '134 : char = 'h2e ; //
    '135 : char = 'h2f ; //
    '136 : char = 'h30 ; //
    '137 : char = 'h31 ; //
    '138 : char = 'h32 ; //
    '139 : char = 'h33 ; //
    '13a : char = 'h34 ; //
    '13b : char = 'h35 ; //
    '13c : char = 'h36 ; //
    '13d : char = 'h37 ; //
    '13e : char = 'h38 ; //
    '13f : char = 'h39 ; //
    '140 : char = 'h3a ; //
    '141 : char = 'h3b ; //
    '142 : char = 'h3c ; //
    '143 : char = 'h3d ; //
    '144 : char = 'h3e ; //
    '145 : char = 'h3f ; //
  endcase
endmodule // myword
```
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!!!
### SRAM vs FF-registers

```verilog
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
```

### SRAM vs FF-registers

```verilog
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
```
Single-Port SRAM/FF

8x8

16x16

32x32

Single-Port SRAM

8x8

16x16

32x32
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
Conclusions

- Try out the makemem program for ROM
- 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
- If you’re using VGA
  - Check out the mini-project from 2005
  - Again, on the class website