Display Technology

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

Cathode Ray Tube

Electron Gun

Beam Steering Coils

Raster Scanning
Color

Shadow Mask and Aperture Grille

Liquid Crystal Displays

DLP Projector

LCoS

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

- Glass Substrate
- ITO Electrode
- Alignment Layer
- Third Metal (Reflective Electrode)
- Second Metal (Wiring Shield)
- First Metal (Wiring)
- Silicon Substrate
- Source
- Drain
- Capacitor Poly Silicon
- 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 every 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  - Now superseded by much higher resolution standards...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

Raster Scanning

| 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 |
VGA Timing

Horizontal Dots: 640
Vertical Scan Lines: 480
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

60Hz vertical frequency

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

12.8/128 = 100ns/pixel = 10 MHz 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. retraceing)
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!

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

---

Raster Scanning
More colors

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

Table 6-2: 480x480 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>Tl</td>
<td>Start pulse time</td>
<td>16.7 µs</td>
<td>405.000</td>
</tr>
<tr>
<td>Twp</td>
<td>Pulse width</td>
<td>15.8 µs</td>
<td>340.000</td>
</tr>
<tr>
<td>Tpp</td>
<td>Pulse peak II</td>
<td>12.8 µs</td>
<td>480.000</td>
</tr>
<tr>
<td>Tpp</td>
<td>Back ground</td>
<td>12.0 µs</td>
<td>230.000</td>
</tr>
</tbody>
</table>

Figure 6-3: VGA Control Timing

The signal timings in Table 6-2 are derived for a 480 by 480 display using a 20 MHz pixel clock and 60 Hz by 1 frame. Figure 6-3 shows the relationship between each of the timing parameters. The timing for the sync pulse width (Twp) and (Tpp) and back ground intervals (Tpp) and (Tpp) are based on observations from various VGA displays. The front and back ground intervals are the pre-scan and post-scan 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)
  - 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

```vhdl
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
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

- 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
    - 16bits (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?
Fit the charROM into a VGA system
- hVideo walks along the row
- vVideo picks which row to walk along

---

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)
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 size comparison

SRAM vs FF-registers

module regfile #(parameter WIDTH = 8, REGBITS = 3)
  (input clk, rewrite,
   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 (rewrite) 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

Makemem also generates SRAM
- Three different variants: single, dual, triple port
- Each port is independent of other ports
- 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!!!
Single-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

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