Digital Systems Design

Video Signal Generation for the Altera DE Board

VGA Video Display Generation

- A VGA signal contains 5 active signals
  - Two TTL compatible signals for synchronization
    - HSYNC – horizontal synchronization
    - VSYNC – vertical synchronization
  - Three analog (0.7-1.0 V peak-to-peak) for color control
    - Red
    - Green
    - Blue
- All other color combinations are generated by changing the analog levels of the three RGB signals
VGA Video on the DE0-CV Board

- There is a 4-bit DAC resistor network used to produce the analog data signals (red, green, and blue)

VGA Video Display Technology

- In standard VGA format, the screen contains 640x480 pixels
  - 640 pixels in a row
  - 480 rows
- The standard refresh rate for a screen is \( \approx 60 \) Hz
  - The entire screen is refreshed 60 times per second
VGA Clock Information

• Each VGA monitor uses a clock that determines when each pixel is updated
• This clock operates at the VGA-specified frequency of 25.175 MHz
• Basis for the 25.175 MHz clock
  – Includes pixel processing time, horizontal and vertical synchronization times and guardband (front and back porch, analog voltage level stabilization) times
  – \([640 \text{ (pixels/row)} + 160 \times 480 \text{ (rows)} + 45 \times 60 \text{ (refreshes/second)}\)
  – 25,200,000 ‘pixels’ processed/second
• The 25.175 MHz clock can be generated from a Phase Lock Loop (PLL) with the 50 MHz clock on the DE board as an input clock

VGA Clock Information (continued)

• General characteristics
  – Clock frequency 25.175 MHz
  – Line frequency 31469 Hz
  – Field frequency 59.94 Hz
• One line
  – 8 pixels front porch
  – 96 pixels horizontal sync
  – 40 pixels back porch
  – 8 pixels left border
  – 640 pixels video
  – 8 pixels right border
  – 800 pixels total per line
• One field
  – 2 lines front porch
  – 2 lines vertical sync
  – 25 lines back porch
  – 8 lines top border
  – 480 lines video
  – 8 lines bottom border
  – 525 lines total per field
VGA Screen Refresh Process

- The monitor refreshes the screen in a prescribed manner that is partially controlled by the horizontal and vertical synchronization signals
  - The monitor starts each refresh cycle by updating the pixel in the top left-hand corner of the screen, which can be treated as the origin of an X–Y plane
  - After the first pixel is refreshed, the monitor refreshes the remaining pixels in the row
  - When the monitor receives a pulse on the horizontal synchronization, it refreshes the next row of pixels.
  - This process is repeated until the monitor reaches the bottom of the screen
  - When the monitor reaches the bottom of the screen, the vertical synchronization pulses, causing the monitor to begin refreshing pixels at the top of the screen (i.e., at [0,0])

Horizontal Refresh Cycle

<table>
<thead>
<tr>
<th>Parameters</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>31.77 μs</td>
<td>3.77 μs</td>
<td>1.89 μs</td>
<td>25.17 μs</td>
<td>0.94 μs</td>
</tr>
</tbody>
</table>

- A: Total Horizontal Refresh Time
- B: Horizontal Synchronization Time
- C, E: Guardband Time
- D: Pixel Refresh Time
Vertical Refresh Cycle

- **O**: Total Vertical Refresh Time
- **P**: Vertical Synchronization Time
- **Q, S**: Guardband Time
- **R**: Horizontal Refresh Cycles

<table>
<thead>
<tr>
<th>Parameters</th>
<th>O</th>
<th>P</th>
<th>Q</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>16.6 ms</td>
<td>64 µs</td>
<td>1.02 ms</td>
<td>15.25 ms</td>
<td>0.35 ms</td>
</tr>
</tbody>
</table>

VHDL VGA Synchronization Example

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-- Module Generates Video Sync Signals for Video Monitor Interface
-- RGB and Sync outputs tie directly to monitor connector pins

entity vga_sync is
  port(
    clock_50mhz, red, green, blue, red_out, green_out, blue_out, horiz_sync_out, vert_sync_out, video_on, pixel_clock, pixel_row, pixel_column : in std_logic;
  horiz_sync_out, vert_sync_out, video_on, pixel_clock, pixel_row, pixel_column : out std_logic;
  video_on, pixel_clock : out unsigned(3 downto 0));
end vga_sync;
```
VHDL VGA Synchronization Example

architecture rtl of vga_sync is

signal horiz_sync, vert_sync, pixel_clock_int : std_logic;
signal video_on_int, video_on_v, video_on_h : std_logic;
signal h_count, v_count : unsigned(9 downto 0);

-- Horizontal Timing Constants
constant h_pixels_across : natural := 640;
constant h_sync_low : natural := 664;
constant h_sync_high : natural := 760;
constant h_end_count : natural := 800;

-- Vertical Timing Constants
constant v_pixels_down : natural := 480;
constant v_sync_low : natural := 491;
constant v_sync_high : natural := 493;
constant v_end_count : natural := 525;

component video_pll
port
( refclk : in std_logic := '0';
  rst : in std_logic := '0';
  outclk_0 : out std_logic );
end component;

begin

-- PLL below is used to generate the pixel clock frequency
-- Uses 50Mhz clock for PLL's input clock
video_pll_inst : video_pll port map ( refclk => clock_50mhz, rst => '0',
  outclk_0 => pixel_clock_int);

-- video_on is high only when RGB pixel data is being displayed
-- used to blank color signals at screen edges during retrace
video_on_int <= video_on_h and video_on_v;

-- output pixel clock and video on for external user logic
pixel_clock <= pixel_clock_int;
video_on <= video_on_int;

end;
process
begin
  wait until(pixel_clock_int'event) and (pixel_clock_int='1');
  -- Generate Horizontal and Vertical Timing Signals for Video Signal
  -- H_count counts pixels (#pixels across + extra time for sync signals)
  --
  -- Horiz_sync -------------------------
  -- H_count  0               #pixels sync low end
  --
  if (h_count = h_end_count) then
    h_count <= "0000000000";
  else
    h_count <= h_count + 1;
  end if;
  --Generate Horizontal Sync Signal using H_count
  if (h_count <= h_sync_high) and (h_count >= h_sync_low) then
    horiz_sync <= '0';
  else
    horiz_sync <= '1';
  end if;

  -- V_count counts rows of pixels (#pixel rows down + extra time for V sync signal)
  --
  -- Vert_sync -------------------------
  -- V_count  0               last pixel row V sync low end
  --
  if (v_count >= v_end_count) and (h_count >= h_sync_low) then
    v_count <= "0000000000";
  elsif (h_count = h_sync_low) then
    v_count <= v_count + 1;
  end if;
  -- Generate Vertical Sync Signal using V_count
  if (v_count <= v_sync_high) and (v_count >= v_sync_low) then
    vert_sync <= '0';
  else
    vert_sync <= '1';
  end if;
end process;
VHDL VGA Synchronization Example

-- Generate Video on Screen Signals for Pixel Data
-- Video on = 1 indicates pixel are being displayed
-- Video on = 0 retrace - user logic can update pixel
-- memory without needing to read memory for display

if (h_count < h_pixels_across) then
  video_on_h <= '1';
  pixel_column <= h_count;
else
  video_on_h <= '0';
end if;

if (v_count <= v_pixels_down) then
  video_on_v <= '1';
  pixel_row <= v_count;
else
  video_on_v <= '0';
end if;

VHDL VGA Synchronization Example

-- Put all video signals through DFFs to eliminate any small timing delays
-- that cause a blurry image

horiz_sync_out <= horiz_sync;
vert_sync_out <= vert_sync;

-- Also turn off RGB color signals at edge of screen during vertical
-- and horizontal retrace
red_out <= red and video_on_int;
green_out <= green and video_on_int;
blue_out <= blue and video_on_int;

end process;
end rtl;
VGA Video Example

- If the vga_sync.vhd design had the connections shown on the input:
  - The design will change the entire screen from red to black when KEY[0] is pressed

Character Based Video Design (Font Data)

- For a video display that contains textual data
- A pixel pattern (font) is needed for each character to be displayed
- Font data can be stored in a ROM inside the Cyclone II device
  - A *.mif file can be used to initialize the ROM contents
  - tcgrom.mif from Hamblen text will suffice
### Char Array Data

<table>
<thead>
<tr>
<th>CHAR</th>
<th>ADDRESS</th>
<th>CHAR</th>
<th>ADDRESS</th>
<th>CHAR</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>å</td>
<td>00</td>
<td>P</td>
<td>20</td>
<td>Space</td>
<td>40</td>
</tr>
<tr>
<td>A</td>
<td>01</td>
<td>Q</td>
<td>21</td>
<td>!</td>
<td>41</td>
</tr>
<tr>
<td>B</td>
<td>02</td>
<td>R</td>
<td>22</td>
<td>&quot;</td>
<td>42</td>
</tr>
<tr>
<td>C</td>
<td>03</td>
<td>S</td>
<td>23</td>
<td>#</td>
<td>43</td>
</tr>
<tr>
<td>D</td>
<td>04</td>
<td>T</td>
<td>24</td>
<td>$</td>
<td>44</td>
</tr>
<tr>
<td>E</td>
<td>05</td>
<td>U</td>
<td>25</td>
<td>%</td>
<td>45</td>
</tr>
<tr>
<td>F</td>
<td>06</td>
<td>V</td>
<td>26</td>
<td>&amp;</td>
<td>46</td>
</tr>
<tr>
<td>G</td>
<td>07</td>
<td>W</td>
<td>27</td>
<td></td>
<td>47</td>
</tr>
<tr>
<td>H</td>
<td>10</td>
<td>X</td>
<td>30</td>
<td>(</td>
<td>50</td>
</tr>
<tr>
<td>I</td>
<td>11</td>
<td>Y</td>
<td>31</td>
<td>)</td>
<td>51</td>
</tr>
<tr>
<td>J</td>
<td>12</td>
<td>Z</td>
<td>32</td>
<td>*</td>
<td>52</td>
</tr>
<tr>
<td>K</td>
<td>13</td>
<td>[</td>
<td>33</td>
<td>+</td>
<td>53</td>
</tr>
<tr>
<td>L</td>
<td>14</td>
<td>Dn Arrow</td>
<td>34</td>
<td>,</td>
<td>54</td>
</tr>
<tr>
<td>M</td>
<td>15</td>
<td>Up Arrow</td>
<td>36</td>
<td>.</td>
<td>56</td>
</tr>
<tr>
<td>N</td>
<td>16</td>
<td>/</td>
<td>37</td>
<td>/</td>
<td>57</td>
</tr>
<tr>
<td>O</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td>F</td>
</tr>
</tbody>
</table>

### Character Based Video (Display Data)

- **Given that a 640x480 display is available**
  - A maximum of 80x60 8x8 characters could be displayed
  - 80x60=4800 characters
    - Would require 38400 bits of memory (M10K blocks)

- **For an 800x600 display**
  - A maximum of 100x75 characters could be displayed
  - 100x75=7500 characters
    - Would require 60000 bits of memory
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity char_rom is
  generic
    (address_width : integer := 9;
     data_width : integer := 8);
  port
    (clock : in std_logic;
     character_address : in std_logic_vector(address_width-1 downto 0);
     font_row, font_col : in std_logic_vector(2 downto 0);
     rom_mux_output : out std_logic);
end char_rom;

architecture behavior of char_rom is
  type rom is array(0 to 2**address_width-1) of std_logic_vector(data_width-1 downto 0);
  signal rom_block : rom;
  attribute ram_init_file : string;
  attribute ram_init_file of rom_block : signal is "tcgrom.mif";
  signal rom_data : std_logic_vector(data_width-1 downto 0);
  signal rom_address : std_logic_vector(address_width-1 downto 0);
begin
  -- small 8 by 8 character generator rom for video display
  -- each character is 8 8-bits words of pixel data
  rom_address <= character_address & font_row;
  -- mux to pick off correct rom data bit from 8-bit word for on screen character generation
  rom_mux_output <= rom_data(to_integer(unsigned(not font_col(2 downto 0))));
  process (clock)
  begin
    if rising_edge(clock) then
      rom_data <= rom_block(to_integer(unsigned(rom_address)));
    end if;
  end process;
end behavior;
Character ROM Display Example

- Note that Pixel_column[0] and Pixel_row[0] bits are not used in FONT_COL and FONT_ROW.
  - The effect is that each bit from the character ROM is displayed in a 2x2 pixel area.
- Since Pixel_row[3..0] bits are not used for the character address, a new character will be displayed every 16th pixel row.

Character ROM Display Example

Output
Graphics Memory Design

• The previous designs considered text only displays
• Consider graphics memory options
  – For a two color display, a pixel on the display would correspond to one bit in a graphics memory
    • 640x480=307200
    • 512x512=262144
    • 256x256=65536
    • 256x256x3=196608 (for an eight color display)

Video Data Options

• Simple video games have a background color with a few moving images
  – No need to store the (default) background color in video memory
  – Hardware comparators track the row and column counts as video signal is generated and detect when another image, other than the background, is to be displayed
    • When the comparator signals that the row and column count matches the image location, the image color data is switched into the RGB output using a multiplexer
  – The image can be made to move if its current row and column location is stored in registers and used as comparator input
  – Additional logic can change the image’s location producing movement
  – Multiple comparators can be used to support several fixed and moving images (often referred to as sprites)
Updating Video Data

- When RGB data is being displayed, pixel memory should be in ‘read’ mode
  - To avoid flicker and memory access conflicts, designs should update pixel RAM (or other hardware that produces the RGB output) during the time RGB data is not being displayed
- Use the horizontal and vertical guardband times to update pixel data