Assignment 9  
Cs342  
Spring 2005  
Chapter 9 (VGA Video Signal Generation)

Please e-mail your design description, testing, documentation, (try Quartus II to verify that they are consistent) waveform files; VHDL files to cs342@cslab

Due Date: Place Date Here

Purpose:

In this assignment you are to generate several video images by using the Altera UP 1 board. You are asked to complete the following tasks:

- Display the color red on the VGA screen. How many colors can you display in total (add this to your report)?
- Implement the character ROM to display the ‘A’ character or any other character you would like.
- Display the vertical bouncing ball on the VGA screen. Afterwards change the motion of the ball from vertical to horizontal.
- Extra points will be rewarded for students how can implement a simple ping-pong game.

VGA Video Display Generation

To understand how it is possible to generate a video image using the Altera UP 1 Board, it is first necessary to understand the various components of a video signal. A VGA video signal contains 5 active signals. Two signals compatible with TTL logic levels, horizontal sync and vertical sync, are used for synchronization of the video. Three analog signals with 0.7 to 1.0-Volt peak-to-peak levels are used to control the color. The color signals are Red, Green, and Blue. They are often collectively referred to as the RGB signals. By changing the analog levels of the three RGB signals all other colors are produced.

Video Display Technology

The major component inside a VGA computer monitor is the CRT or Cathode Ray Tube shown in Figure 9.1. The electron beam must be scanned over the viewing screen in a sequence of horizontal lines to deflect the electron beam.

In standard VGA format, as seen in Figure 9.2, the screen contains 640 by 480 picture elements or pixels. The video signal must redraw the entire screen 60 times per second to provide for motion in the image and to reduce flicker at refresh rates less than 30 Hz.
The onboard clock on the Altera UP 1 board produces a fixed 60 Hz refresh rate. The color of each pixel is determined by the value of the RGB signals when the signal scans across each pixel. In 640 by 480-pixel mode, with a 60 Hz refresh rate, this is approximately 40 ns per pixel. A 25 MHz clock has a period of 40 ns.

**Video Refresh**

The screen refresh process seen in Figure 9.2 begins in the top left corner and paints 1 pixel at a time from left to right. At the end of the first row, the row increments and the column address is reset to the first column. Each row is painted until all pixels have been displayed. Once the entire screen has been painted, the refresh process begins again.

![Color CRT and Phosphor Dots on Face of Display](image)

**Figure 9.1** Color CRT and Phosphor Dots on Face of Display.
Many VGA monitors will shut down if the two sync signals are not the correct values. Most PC monitors have an LED that is green when it detects valid sync signals and yellow when it does not lock in with the sync signals.

**Using a CPLD for VGA Video Signal Generation**

In Figure 9.5, a 25.175 MHz clock, that is the 640 by 480 VGA pixel data rate of approximately 40 ns is used to drive counters that generate the horizontal and vertical sync signals. Additional counters generate row and column addresses. In some designs, pixel resolution will be reduced from 640 by 480 to a lower resolution by using a clock divide operation on the row and column counters. The row and column addresses feed into a pixel RAM for graphics data or a character generator ROM when used to display text. The required RAM or ROM is also implemented inside the CPLD chip.
The UP1core function, VGA_SYNC can be used to generate the timing signals needed for a VGA video display. Although VGA_SYNC is written in VHDL, like the UP1core function it can be used as a symbol in a design created with any entry method.

To turn off RGB data when the pixels are not being displayed the video_on signals are generated. Video_on is gated with the RGB inputs to produce the RGB outputs to force them to the zero state. VGA_SYNC also puts all of the video outputs through a final register to eliminate any timing differences in the video outputs. VGA_SYNC outputs the pixel row and column address.

Final Output Register for Video Signals

The final video output for the RGB and sync signals in any design should be directly from a flip-flop output. Even a small time delay of a few nanoseconds from the logic that generates the RGB color signals will cause a blurry video image. Since the RGB signals must be delayed a 25 MHz clock period to eliminate any possible timing delays, the sync signals must also be delayed by clocking them through a D flip-flop. If the outputs all come directly from a video image is produced. The last few lines of VHDL code in the UP1core VGA_SYNC design generate this final output register.

Required Pin Assignments for Video Output

The UP 1 board requires the following FLEX chip pins be defined in the project’s *.gdf, *.acf file, or elsewhere in the design in order to display the video signals:

<table>
<thead>
<tr>
<th>Pin</th>
<th>INPUT_PIN</th>
<th>OUTPUT_PIN</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>91</td>
<td></td>
<td>--25 MHz Clock Input Pin</td>
</tr>
<tr>
<td>Red</td>
<td>236</td>
<td></td>
<td>--Red Data Signal Output Pin</td>
</tr>
<tr>
<td>Blue</td>
<td>238</td>
<td></td>
<td>--Blue Data Signal Output Pin</td>
</tr>
<tr>
<td>Green</td>
<td>237</td>
<td></td>
<td>--Green Data Signal Output Pin</td>
</tr>
<tr>
<td>Horiz_Sync</td>
<td>240</td>
<td></td>
<td>--Horizontal Sync Signal Output Pin</td>
</tr>
<tr>
<td>Vert_Sync</td>
<td>239</td>
<td></td>
<td>--Vertical Sync Signal Output Pin</td>
</tr>
</tbody>
</table>

These pins are hard wired on the UP 1 board to the VGA connector and cannot be changed.

Video Examples

The following schematic is a video simulation of a red LED. When the PB1 pushbutton is hit, the color of the entire video screen will change from black to red.
VGA_SYNC outputs the pixel row and column address. Pixel_row and Pixel_column are normally inputs to user logic that in turn generates the RGB color data. Here is a simple example that uses the pixel_column output to generate the RGB inputs. Bits 7, 6, and 5 of the pixel_column count are connected to the RGB data. Since bits 4 through 0 of pixel column are not connected to the RGB data. Since bits 4 through 0 of pixel column are not connected, RGB color data will only change once every 32 pixels across the screen. This turns generates a sequence of color bars in the video output. The color bars display the eight different colors that can be generated by the three digital RGB outputs.

A Character Based Video Design

A memory initialization file, *.mif, can be used to initialize the ROM contents during download. Given the memory limitations inside the FLEX CPLD, one option that fits is a display of 40 characters by 30 lines.

Each letter, number, or symbol is a pixel image from the 8 by 8 character font. To make the characters larger, each dot in the font maps to a 2 by 2 pixel block so that a single character requires 16 by 16 pixels. This was done by dividing the row and column counters by 2.

Character Selection and Fonts

The screen is constantly being refreshed and the video image is being generated on-the-fly as the beam moves across the video display, it is necessary to use other registers, ROM, or RAM inside the CPLD to hold and select the characters to be displayed on the screen. Each location in this character ROM or RAM contains only the starting address of the character font in font ROM; using two levels of memory results in a design that is more compact and used far less memory bits. This technique was used on early generation computers before the PC.
Here is an example implementation of a character font used in the UP1core function, char_ROM is shown in your Rapid Prototyping of Digital Systems. To display an “A” the character ROM would contain only the starting address 000001 for the font table for “A”.

A 3-bit font column address can be used with a multiplexer to select appropriate bit from the ROM output word to drive the RGB pixel color data. The Character font ROM and the multiplexer are contained in the UP1core char_ROM as shown below (see chapter 5 for information on CHAR_ROM). The VHDL code for CHAR_ROM declares the memory size using LPM_ROM function and the tcgrom.mif contains the initial values or font data for the ROM.

**Figure 9.7 Accessing a Character Font Using a ROM.**

**VHDL Character Display Design Examples**

The UP1cores VGA_SYNC and CHAR_ROM are designed to be used together to generate a text display. CHAR_ROM contains an 8 by 8 pixel character font. In the following schematic, a test pattern with 40 characters across with 30 lines down is displayed. Examining the RGB inputs on the VGA_SYNC core you can see that characters will be white (111 = RGB) with a red (100 = RGB) background. Each character uses a 16 by 16 pixel area in the 640 by 480 display area. Since the low bit in the pixel row and column address is skipped in the font row and font column ROM inputs, each data bit from the font is displayed in a 2 by 2 pixel area. Since pixel row bits 9 to 4 are used for the character address a new character will be displayed every 16\(^{th}\) pixel row or character line; division by 16 without any logic since the low four bits are not connected.

![Character Font Access Diagram](image-url)
Normally, more complex user designed logic is used to generate the character address. The video example shown in Figure 9.8 is an implementation of the MIPS RISC processor core. The values of major buses are displayed in hexadecimal and it is possible to single step through instructions and watch the values on the video display. This example includes both constant and variable character display areas. The video setup is the same as the schematic, but additional logic is used to generate the character address.

![Character Test Design Example](image)

**Figure 9.8 MIPS Computer Video Output.**

Pixel row address and column address counters are used to determine the current character column and line position on the screen. They are generated as the image scans across the screen with the VGA_SYNC core by using the high six bits of the pixel row and pixel column outputs. Each character is a 16 by 16 block of pixels. The divide by 16 operation just
requires truncation of the low four bits of the pixel row and column. The display area is 40 characters by 30 lines.

Constant character data for titles in the left column is stored in a small ROM called the character format ROM; this section of code sets up the format ROM that contains the character addresses for the constant character data in the left column of the video image for the display.

```
-- Character Format ROM for Video Display
-- Displays constant format character data
-- on left side of Display area

format_rom: lpm_rom
   GENERIC MAP (  
      lpm_widthad  => 6,  
      lpm_numwords =>"60",  
      lpm_outdata  =>'UNREGISTERED',  
      lpm_address_control =>'UNREGISTERED',  
                        "Reads in mif file for data display titles"
   lpm_file     =>'format.mif',  
   lpm_width    => 6)
```

Each 25 MHz clock cycle, a process containing a series of nested CASE statements is used to select the character to display as the image scans across the screen. The CASE statements check the row and column counter outputs from the sync unit to determine the exact character column and character line that is currently being displayed. The CASE statements then output the character address for the desired character to the char_ROM UP1core.

Hexadecimal variables in the right column in Figure 9.8 are generated by using 4-bit data values from the design to index into the character font ROM. As an example, the value “11” & PC(7 DOWNTO 4), when used as the character address to the UP1core, char_ROM, will map into the character font for 0..9 and A..F. The actual hex character selected is based on the current value of the 4 bits in the VHDL signal, PC. As seen in the last column of Table 9.1, the letters, A..F, appear again after decimal numbers in the font ROM to simplify this hexadecimal mapping conversion.

**A Graphics Memory Design Example**

For another example, assume the display will be used to display only graphics data. The FLEX 10K20 EABs contain 12K bits of memory. If only two colors are used in the RGB signals, one bit will be required for each pixel in the video RAM. If a 64 by 64 pixel video RAM was implemented in the FLEX chip is would use 4K bits of the chip’s 12K-bit memory. For full color RGB data of three bits per pixel, a 64 by 64 pixel RAM would be left for the remainder of the design.
When the scan of each horizontal line is complete there are around 160 clock cycles before the next RGB value is needed, as seen in Figure 9.9.

In most cases, calculations that change the video image should be performed during this off-screen period of time to avoid memory conflicts with the readout of video RAM or other registers which are used to produce the RGB video pixel color signals. Since pixel memory is limited, complex graphic designs with higher resolutions will require another approach.

**VHDL Graphics Display Design Example**

Below is a simple graphics example that will generate a ball that bounces up and down on the screen. As seen in Figure 9.10, the ball is red and the background is white. This example requires the VGASYNC design from Section 9.4 to generate the video sync and the pixel address signals. The pixel_row signal is used to determine the current row and the pixel_column signal determines the current column. Using these current row and column addresses, the process RGB_Display generates the red ball on the white background and produces the ball_on signal which displays the red ball using the logic in the red, green, and blue equations. Ball_X_pos and Ball_y_pos are the current address of the center of the ball. Size is the size of the square ball.

![Figure 9.9 Display and Compute clock cycles available in a single Video Frame.](image-url)
The process Move_Ball moves the ball a few pixels every vertical sync and checks for bounces off of the walls. Ball_motion is the number of pixels to move the ball at each vertical sync clock. The VGA_SYNC process is also used to generate sync signals and pixel addresses but is not shown in the code below.