Lab 6: VGA Interface

EEL 4712 – Spring 2020

Objective:

The objective of this lab is to study the generation of the control signals required by a VGA monitor by creating a component VGA_sync_gen that generates those signals. Using VGA_sync_gen and additional components, a simple color raster image will be displayed on a VGA monitor.

Required tools and parts:

QuartusII software package, ALTERA DE10-lite board, and VGA Monitor.

IP Used:

An altsyncram IP component will be used in this lab. All output bits of this lab should be directed to the appropriate pins of the ALTERA DE10-lite VGA connector. Also, a memory initialization file (brom.mif) will be used to initialize the memory values of the ROM.

Introduction:

The VGA monitor connector on the ALTERA DE10-lite board consists of five signals, RED(4 bits), GREEN(4 bits), BLUE(4 bits), HORIZ_SYNC, and VERT_SYNC. Look into the user manual for the header pin outs for these 5 signals. The timing relationships among this signal are shown in **Figures 1 and 2** at the end of this lab document. The generation of signals needed for the raster on the VGA monitor begins with dividing the frequency of the 50 MHz clock on the ALTERA DE0 board down to a 25 MHz pixel clock which is further divided down to the horizontal and vertical sync frequencies. This means your design will contain a ***horizontal counter*** and a ***vertical counter***. The horizontal and vertical sync pulses of appropriate lengths are then produced from the two counters.

Horizontal and vertical synchronization. A video display consists of 640 pixels in the horizontal direction and 480 lines of pixels in the vertical direction. 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 (0,0) 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 ***HORIZ_SYNC*** pin, it refreshes the next row of pixels. The time required for the sweep, the horizontal sweep period, is nominally 31.77 μs. This process is repeated until the monitor reaches the bottom of the screen. When the monitor reaches the bottom of the screen, a 64 μs pulse applied to the ***VERT_SYNC*** pin causes the monitor to begin refreshing pixels at the top of the screen (i.e., at [0,0]). As shown in Figure 2, the VERT_SYNC pulse must be repeated every 16.6 ms (vertical sweep period). A complete screen of information is traced by the electron beam every 16.6 ms for a refresh rate of 60 Hz.

Blanking intervals. In order to accommodate the time required to move the electron beam back to the left side of the screen (horizontal retrace period) the electron beam must be turned off during the retrace time or the return trace would be visible. This is accomplished by two blanking intervals during the horizontal refresh cycle marked by B and C at the beginning of the trace and E at the end of the trace in Figure 1. Similar blanking intervals are required during vertical refresh (P, Q, and S in Figure 2). The color beams are turned on only when the horizontal counter is in the period D and the vertical counter is in period R.

Hint: Blanking intervals can be easily implemented by creating a video "gate" signal (Video_On) that is true only when the color beams can be active. Refer to the figure 3 at the end of this lab write-up for a graphical look at the VGA display signals.

Pre-lab requirements:

1. Pixel Clock Generate Component

Create a behavioral component to generate a 25 MHz pixel clock (Input: clk, Output: pixel_clock). You need a 25 MHz clock because the VGA interface expects a new pixel every cycle at this frequency. Use the pixel clock for every other component in this lab. **DO NOT use the 50 MHz clock in any other component.**

2. VGA Sync Component (VGA_sync_gen)

Create a ***behavioral VHDL*** file for a component called VGA_sync_gen.

- Be sure to include the IEEE.NUMERIC_STD package.
- Be sure to include the work.VGA_LIB package, which is a custom package for this project. Some useful constants are provided in vga_lib.vhd (provided on the website). **Depending on your exact implementation, you might need to modify these constants by a small amount.**
- The entity should have a clock and reset input and 5 outputs: Hcount, Vcount, Horiz_Sync, Vert_Sync, and Video_On.
- Create two counters. Both counters should be declared as 10-bit std_logic/unsigned signals.
  - **Hcount**
    - Continually counts up to the horizontal period (H_MAX – See vga_lib.vhd) and then starts over at 0, using the 25 MHz pixel clock.
    - A value of zero on Hcount corresponds to the beginning of section D in Figure 1.
  - **Vcount**
    - Counts up to the vertical period (V_MAX – See vga_lib.vhd). It will increment at a particular point in the horizontal counters count (Hcount = H_VERT_INC – See vga_lib.vhd).
    - A value of zero on Vcount corresponds to the beginning of section R in Figure 2.
- The values of Video_On, Horiz_Sync and Vert_Sync are determined by decoding the values of Hcount and Vcount and comparing the counter values to the constants provided in vga_lib.vhd.

Create a testbench that illustrates the correct behavior and timing of all outputs. First focus your simulations on the horizontal refresh cycle, since it can be simulated in its entirety easily. Then try to measure the length of VERT_SYNC (64 µs) in simulation, and do a full simulation of the vertical refresh cycle. Make sure the actual timings are as close as possible to the illustrated timings when using a 25 MHz clock.

Turn in all vhdl files and annotated simulation results showing the correct behavior of each VGA_sync_gen output.

3. Display of Raster Picture

In this part of lab, you will partition the screen into 2x2 pixel-sized blocks, each of which displays a color made from a combination or red, green, and blue. There will be a total of 4096 blocks arranged as a 64x64 grid, which forms a 128x128 image. VGA resolution is 640x480, so you must make sure the pixels not used by the image are black. Your circuit should allow the image to be placed in five different locations: centered (when no buttons are pressed), the top left corner (when the top left button is pressed), the top right corner (when the top right button is pressed), etc.

You will need to utilize **three additional parts** to implement this color raster picture: a ROM that contains the RGB values for each block, logic that generates the current block row, and logic that generates the current block column. They are described as follows:

a) **Block row address logic**: This takes the Vcount signal and generates a 6-bit row address that identifies one of the 64 rows of the 64x64 grid. The block row address will be used as a part of the

address input to the ROM discussed below. Note that this logic should take into account the position of the image based on which buttons are pressed. I recommend using an enable output that forces the color outputs to 0 during an inappropriate row.

b) **Block column address logic**: This takes the Hcount signal and generates a 6-bit column address that identifies one of the 64 columns of the 64x64 grid. The block column address will be used as a part of the address input to the ROM discussed below. Note that this logic should take into account the position of the image based on which buttons are pressed. I recommend using an enable output that forces the color outputs to 0 during an inappropriate row.

c) **ROM**: The ROM (BROM) contains the RGB values for each block of the 64x64 grid. Thus, the size of ROM is 4096x12, with an 12-bit address (2^{12} = 4096) and a 12-bit output. You should combine the block row and column addresses to form a 12-bit address, for which the ROM will output the RGB colors (12 bits, 4 bits each) for that block.

The BROM component is made from the altsyncram component provided by Quartus II. Use the Megawizard Plugin Wizard to create a 1-Port ROM (in the memory compiler section).

- Tools -> MegaWizard Plug-In Manager
- Select “Create a new custom megafunction variation”, click “Next”.
- Click on the triangle next to “Memory Compiler”.
- Click on “ROM: 1 PORT”.
- Specify that the output should be VHDL (if it is not checked already).
- Browse to the desired folder and name your file vga_rom.vhd.
- Click “Next”.

On the next page of the wizard, spccify that:

- ‘q’ output bus should be 12 bits.
- There should be 4096 words.
- Leave everything else as default.
- Click “Next”.

On the next page:

- ‘q’ output should not be registered.
- Leave everything else as default.
- Click “Next”.

On the next page:

- Browse to the provided file brom.mif.

Leave everything else as default and finish.

The generated file (vga_rom.vhd) can now be used as a component in your design. **IMPORTANT: the vga_rom file will reference brom.mif. You will likely need to modify the generated code to specify the full path to brom.mif, since the generated code will likely use a relative path that might not be correct for your Modelsim project.**

A memory initialization file (brom.mif) will contain the actual memory values of the ROM and can be edited by using the Memory Editor in Quartus (or using a text editor). The brom.mif can be downloaded from the class website and can be edited to modify the ‘picture’ being displayed on the VGA monitor. Using the original brom.mif file, you should see a color gradient changing from black on the top left corner to white on the bottom right corner.

Design and test each part using your own test benches (simulate each component individually) with limited simulation runs.

Turn in the design files and annotated simulation results for the above parts. The simulation should show the basic functionality of each entity

4. Final test setup

Create a top_level VGA entity that connects together your VGA_sync_gen, ROM, column address logic, and row address logic to make a circuit that will display a raster picture as previously described. Add any additional components or logic that may necessary (hint: enables to turn off pixels outside of the image). The top level entity has a 3 bit vector as input and generates the VGA signals as outputs. Assign the input vector to the push button switches. If the input signal is 0, show the image at the center of the screen; if it is 1, at the top left, if 2, at the top right, if 3, bottom left, if 4, bottom right (See figure 4). Create a test bench and simulate the entity to prove that your design works.

Turn in the design files and annotated simulation results for the final circuit.

Pre-lab turn in instructions:

To submit your pre lab please create a folder that is named your UFID. Inside create another folder P1,P2,P3... for each part that contains the VHDL files for that part. If a report is needed then please include it in the UFID folder. Then zip and upload the folder which is named your UFID.

In-lab procedure:

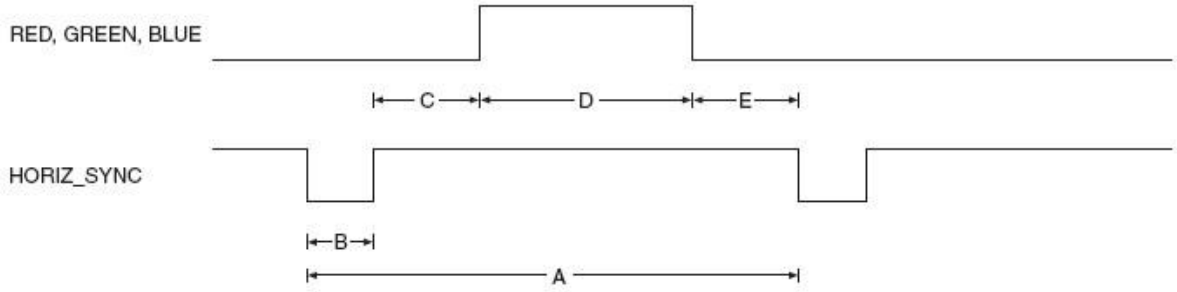
1. Program your board with the final test setup from Prelab step 3 and connect a monitor to your board's VGA connector. Confirm that a picture is displayed on your VGA monitor and show your TA.
2. Modify your code to display a 128x128 image. Use the brom128x128.mif file provided on the website.
3. Modify the brom.mif file to show a picture of your choosing. You can be creative! Some suggestions:
    - Change your brom.mif file and your Row/Column block address generators to display an image larger than 128x128 and possibly not square (eg: 200x150 pixels)
    - Animate the displayed image. Animation could be based on offsetting the address generated and using modular arithmetic or changing the video memory.
    - Extra credit will be given based on images with more complexity. A different 128x128 image is required. Anything more complex will get extra credit.
4. Your TA may ask you to modify your design in some way, so save some in-lab time for a new task. **Be prepared to answer simple questions or to make simple extensions that your TA may request. If you have done the pre-lab exercises, these questions should not be difficult.**

Common problem w/ old board, could be similar problem with current board: Many students get the error "multiple pins assigned to pin location Pin_K22" when they try and compile their top-level design. This error occurs because the vga blue[0] pin uses pin K_22 but by default pin K_22 is used for the nCEO signal as a programming pin for chaining multiple systems together. This error can be fixed by doing the following from within Quartus then recompiling:

Assignments -> Device -> Device and Pin Options -> Dual-Purpose Pins -> Set nCEO as "Use as regular I/O"

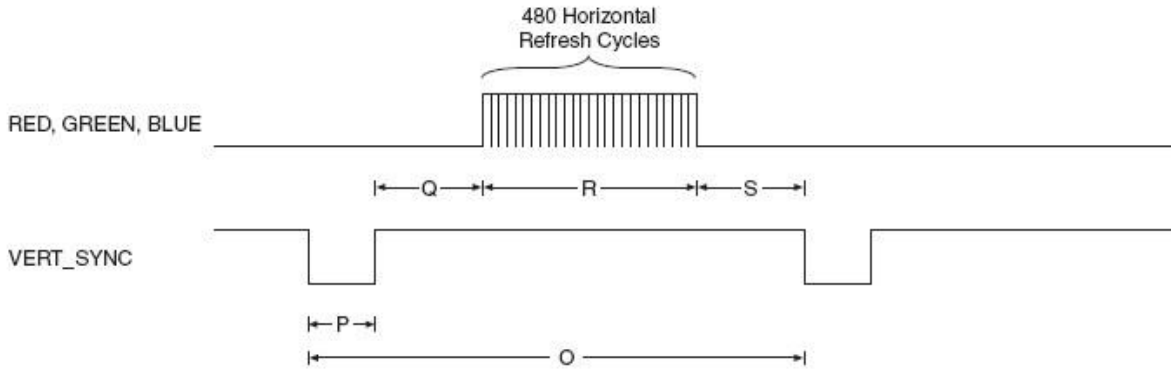
In-lab Option:

In order to test your VGA_sync_gen component you may want to design a simple component, Colorbars, which displays three vertically-aligned Red, Green, and Blue bars across the screen. This component takes Hcount as an input, and gives Red, Green, and Blue signals as outputs. It can be easily made using an IF statement. This component is not required, but will allow students to get partial credit if they cannot get their full design working.



Parameters	A	B	C	D	E
Time	31.77 µs	3.77 µs	1.89 µs	25.17 µs	0.94 µs

Figure 1. Horizontal Refresh Cycle.



Parameters	O	P	Q	R	S
Time	16.6 ms	64 µs	1.02 ms	15.25 ms	0.35 ms

Figure 2. Vertical Refresh Cycle.

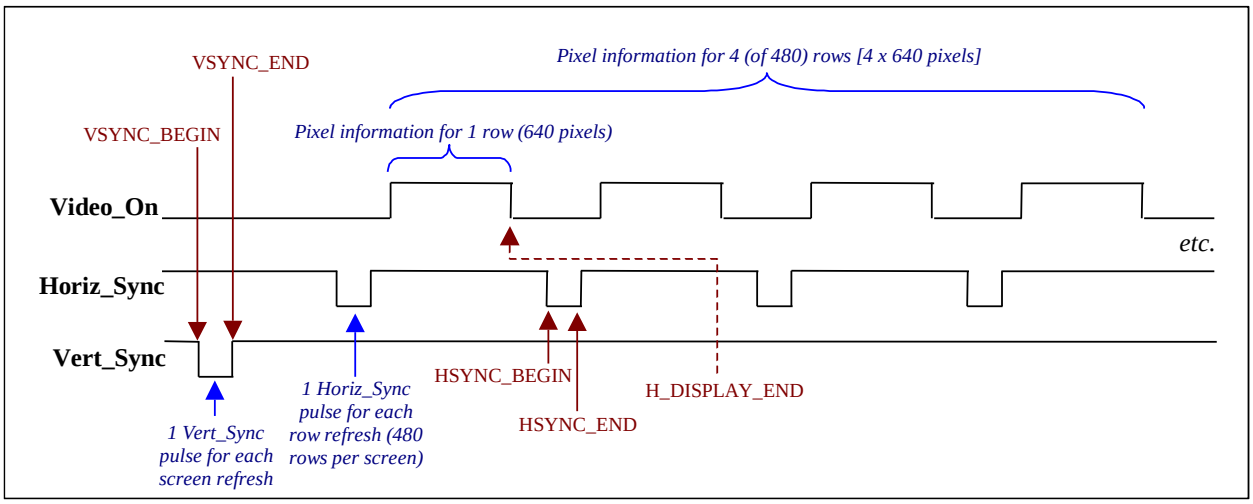


Figure 3. Timing Diagram for four rows of a VGA Display.

Figure 4. Display positions for different button combinations. Exactly one of these can be active at a time.