CSE100 Lab 3 – Screensaver

Contents

Copyright © 2024 Ethan Sifferman.

All rights reserved. Distribution Prohibited.

1 Submission

- Due: Thursday 10/24/2024 at the end of your lab section.
- Submit your Prelab answers to Gradescope.
- Submit your code to the Gradescope autograder. (You have unlimited submission attempts).
- Demonstrate your implementation to a TA.

2 Goals

You will implement a VGA interface with a counter built from adders and flip-flops. Then, you should display the contents of a ROM over VGA.

Once you finish your implementation, you will program it to a Basys 3 FPGA Board.

3 Prelab (ungraded)

3.1 VGA controller

You are provided the `vga_timer` module inside the file `"rtl/vga_timer.sv"`. You will have to implement it according to the VESA standard for “640 × 480 @ 60 Hz”.

Read the section on the VGA Port in the BASYS3 Board Reference Manual. But do not use the circuit design shown there: it is asynchronous.

To control the monitor you must generate two control signals, `hsync` and `vsync`, as well as the 12 RGB data signals (`vgaRed[3:0]`, `vgaBlue[3:0]`, `vgaGreen[3:0]`) for each of the screen’s 640 × 480 pixels. The value of these 12 signals is sent one at a time for each pixel, row by row from left to right and top to bottom using one cycle of the 25.175MHz clock (provided to you) for each pixel. There is also some time between rows and between frames (after all 480 rows) which allows the cathode ray to be re-positioned for the next row or frame. The `hsync` and `vsync` signals are used by the monitor to “synchronize” the start of each row and frame; they are low at fixed times between rows and frames. (Yes, the monitors we will use are LCD displays, not cathode ray tubes. But the protocol used to communicate with these monitors is a standard that lives on.)

One way to think of this is to imagine that you have an 800 × 525 grid of pixels as shown below (instead of the 640 × 480 pixels which correspond to the area you see on the monitor). This grid is traversed starting at the top left, location row 0, column 0. Each row is traversed from left to right followed by the row immediately below it and so on. The region of dimension 640 × 480 at the top left is the Active Region: the pixels in this region correspond to pixels on the screen. The pixels outside this region correspond to time where the cathode ray would be off the screen. So, the L-shaped region outside the active region is not part of the screen but represents the time needed between rows and frames in terms of pixels. The value of the RGB data signals determine the color displayed for pixels in the Active Region, with one cycle of the 25.175MHz clock corresponding to a pixel. For a pixel outside the Active Region the 12 RGB data signals should be low. The horizontal synchronization signal (`hsync`) should be low for the 96 pixels in each row starting with the 656th pixel, and high for the rest. The vertical synchronization signal (`vsync`) should be low for all the pixels in the 490th and 491st rows, and high for all pixels in all other rows. So `hsync` and `vsync` are low in only the regions shaded pink and blue below, respectively.

The frame is continuously transmitted to the monitor to refresh the image. Transmitting one frame takes 800 × 525 × 40ns = 16,800,000ns = 16.8ms, so the monitor is being refreshed roughly 60 times per second: at 60Hz.

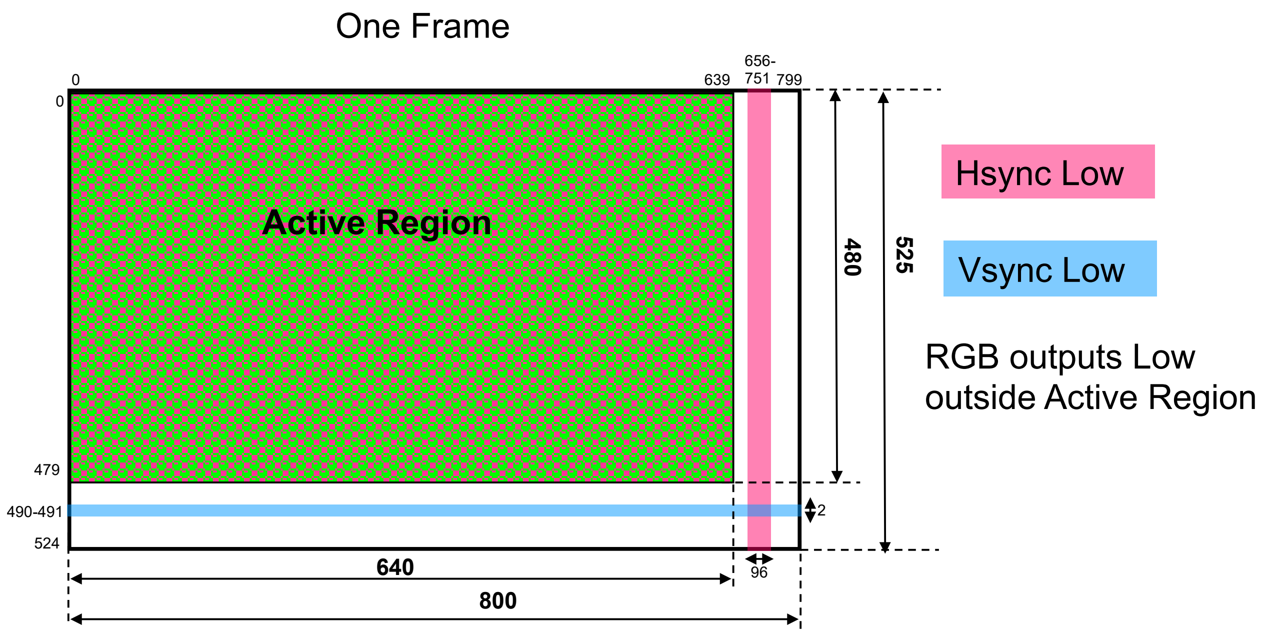

Figure 1: VGA Timing

3.2 `images` module

You are provided the `images` module in `"rtl/images.sv"`. Every cycle, the `images` module produces image data, retrieved from internal ROMs. You do not need to modify this module, (although you may change which images are stored for fun).

`"scripts/generate_image_rom.py"` is automatically run to generate the `.memh` files for the 4 different image ROMs. Each pixel is 12 bits, 4 bits for red, green, and blue respectively.

`images` will take in a ROM address, and will produce the corresponding pixel value 1 cycle later. Be sure to account for this delay in your design.

Each row of pixels is stored sequentially after each other. For example, the pixel value at x=5, y=2 would be found at address=325. There are several methods to generate an address from a coordinate, however the simplest is likely with the expression `y*SCREEN_WIDTH + x` where `SCREEN_WIDTH` is the constant value of 640.

Questions

1. Draw a circuit schematic that implements the expression `y*SCREEN_WIDTH + x`. Show how the constant multiplication can be converted to a series of additions.

3.3 `screensaver` module

You are provided the starter code to the `screensaver` module in `"rtl/screensaver.sv"`. The `screensaver` module should instantiate `vga_timer`, and display the correct pixel values from the `images` module.

The `screensaver` module should have a register to store which image should currently be displayed:

Button(s) Pressed	Image to Display
`image_select==4'b0001`	`IMAGE0`
`image_select==4'b0010`	`IMAGE1`
`image_select==4'b0100`	`IMAGE2`
`image_select==4'b1000`	`IMAGE3`
else	keep previous image

Note that the `images` module stores 160 × 120 images, though your VGA timer expects 640 × 480 images. You will need to stretch the image to fill the entire VGA monitor. You can do this by truncating off the bottom 2 bits of the VGA x and y positions before calculating the ROM address.

Finally, recall that the `images` module will convert an address to a pixel value after a 1-cycle delay. Ensure that your sync pulses properly match that 1-cycle delay. You should add a flip-flop after `hsync`, `vsync`, and `visible` so they are synchronized with your pixel values.

Questions

2. Draw a block diagram for the `screensaver` module. You may see all module inputs and outputs in`"rtl/screensaver.sv"`. You should use the following cells in your schematic:
   - `vga_timer`
   - `images`
   - Coordinate to Address Cell (`y*SCREEN_WIDTH + x`)
   - `current_image` register
   - `hsync` register
   - `vsync` register
   - `visible` register
   - Anything else you need

4 Lab

You need to complete and submit the following files:

- `"rtl/screensaver.sv"`
- `"rtl/vga_timer.sv"`
- `"synth/basys3/Basys3_Master.xdc"`

1. Finish the `vga_timer` module inside the file `"rtl/vga_timer.sv"`. It should implement the VESA standard for "640 × 480 @ 60 Hz" according to the pre-lab. There should be a port denoting the "hsync" and "vsync" pulses, a port denoting whether the signal is in the visible region, and a port showing the coordinate of what pixel should be currently drawn. You may add or remove ports as you see fit. You should regularly simulate your design and look at the generated waveforms as specified in the README. You may temporarily comment out the `first_row_pixels_check` block in `"dv/tb.sv"` so that only your `hsync` and `vsync` pulses are tested.

2. Finish the `screensaver` module inside the file `"rtl/screensaver.sv"` according to the pre-lab. You should regularly lint and simulate your design. Be sure to uncomment the `first_row_pixels_check` block in `"dv/tb.sv"`. Note that passing the simulation will not guarantee that your implementation is fully correct.

3. The Basys3 constraints file `"synth/basys3/Basys3_Master.xdc"` was downloaded from Digilent's GitHub: https://github.com/Digilent/Basys3/blob/master/Resources/XDC/Basys3_Master.xdc. Complete it to match your `basys3` module. Refer to previous labs and the Basys3 Reference Manual if you have questions: https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual.

4. Lint and simulate your design using commands specified in the `"README.md"`. Submit to the autograder until you get 100%. Note that passing the autograder will not guarantee that your implementation is correct.

5. Synthesize your design and program it to the Basys3 using commands specified in the `"README.md"`. Your design may not work the first time. Review the waveforms in Gtkwave for any issues that the autograder did not catch.

6. Get checked off by a TA.