RUTGERS School of Engineering

Embedded Systems Design

14 : 332 : 493 : 03 / 16 : 332 : 579 : 05

Lab 4: "Now You See It, Now You Don't"

Introduction

In this lab, we are going to utilize the VGA analog video standard in order to produce a static image on a display. The timing signals and ROM addressing will all be driven by a combination of counters, which prove to be one of the most versatile components in digital design.

Prelab - Wires with Electron Guns

Background

In order to understand VGA graphics, one must first understand the hardware that it was intended to be used with. Back in the days of Cathode Ray Tube monitors (CRT), generating a color image was a relatively simple process. The front of the monitor consisted of a plate of glass that had a coating of phosphorescent particles on the back. By firing an electron beam at the particles, they would become energized and emit light. The color of the light would depend on the chemical composition of the particle, and the intensity on the level of energy imparted by the beam. In order to direct the beam on the screen, a set of electromagnetic coils are used to deflect it to various angles on the x and y axes.

In this way, the question then becomes how we can generate multiple colors using this technology. The answer is simple; three types of particles that glow red, green, and blue. By creating combinations of these three colors and exploiting some glitches in the human visual system, we can create a large number of colors.

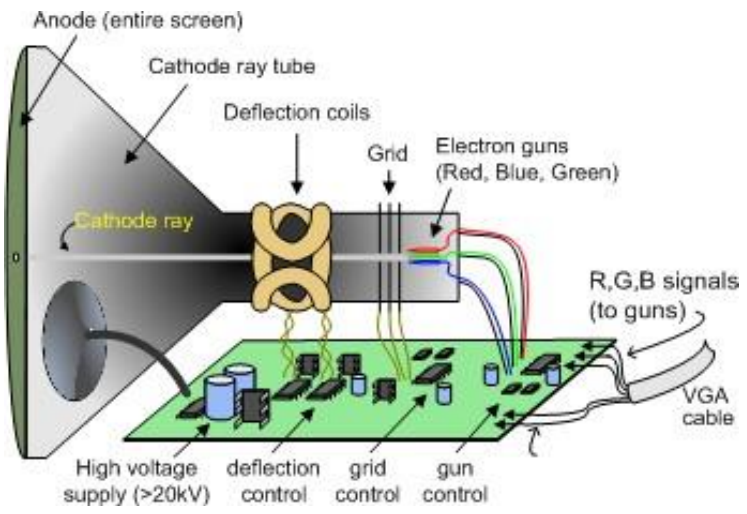

Figure 4.1: CRT Monitor Hardware [3]

Obviously, the CRT has gone the way of the Dodo bird but the VGA standard has remained as the lowest common denominator. Almost all displays supported the standard, but it has begun to be slowly phased out. Even, newer Zybo boards do not have VGA ports available. However, while it is old and has been superseded by a myriad of options, it is still a very useful method for getting images out of an FPGA. Moreover, the techniques and theory behind video generation in the context of VGA are the foundation of the other methods (DVI and HDMI for 640x480 simply encode the VGA RGB components using TMDS and serialize them out at 250MHz using differential signals with an explicit clock).

Knowing all about the technology is wonderful, but if it is not going to be used to do something meaningful then it is all for naught. In order to do so, we must first prepare something to display. For this purpose, a simple Matlab script has been created. It takes an input image and downsamples it to an 8-bit color space while also creating a Xilinx COE file that we will use later to create a ROM to hold our image data.

In order to use the script, please download an image of your choosing in JPG format and save it to a folder. When the script is run, it will ask for you to enter the **absolute** path to that folder. It will then ask for you to enter the name of the output file you wish to create. **It will then take the first JPG in that folder and convert it**. The output file will be stored in the same folder as the image that was used for input. As an aside, the script is designed to be tolerant of all resolutions and targets a 480x480 output resolution due to memory limitations on the Zybo. If images are too big, they will be cropped automatically. If they are too small, they will be padded with black. Since the script may modify the image, save the Matlab version of the image so you can later compare it.

Prelab Task

- Read section 11 on VGA in the Digilent reference manual for the Zybo board
- Run the provided Matlab script with an image of your choosing. Keep the output file for use later in the lab

1 Timing is Everything

1.1 Background

Now that you understand how the technology for the monitors worked back in the day, you have some preliminary idea about the signals that were used to control those monitors. In the end, VGA can be reduced to 5 signals. Keep in mind that for modern monitors, they interpret these signals differently (they don't use electron guns any more) but the principles are the same.

- R: An analog intensity for the "red" electron gun
- G: An analog intensity for the "green" electron gun
- B: An analog intensity for the "blue" electron gun
- HS (Horizontal Sync): A signal that initiates the "newline" for the electron guns
- VS (Vertical Sync): A signal that initiates the "new page" for the electron guns

Through these 5 signals (and no explicit clock, as it is recovered by the monitor via doing some math on the HS and VS signals), we output color intensities to a continuously moving set of electron beams.

While the R, G, and B signals are relatively obvious in what they are, the same cannot be said for HS and VS. In order to understand those, we have to look at several tables and diagrams.

The first thing that we realize when we look at these diagrams is that even though our screen resolution is 640x480, we actually base our signals off an 800x525 "screen". This is because there is off screen space both below and to the right of the image. In the old days, this existed to allow the electron guns time to move over when we told them to go the next line or next screen.

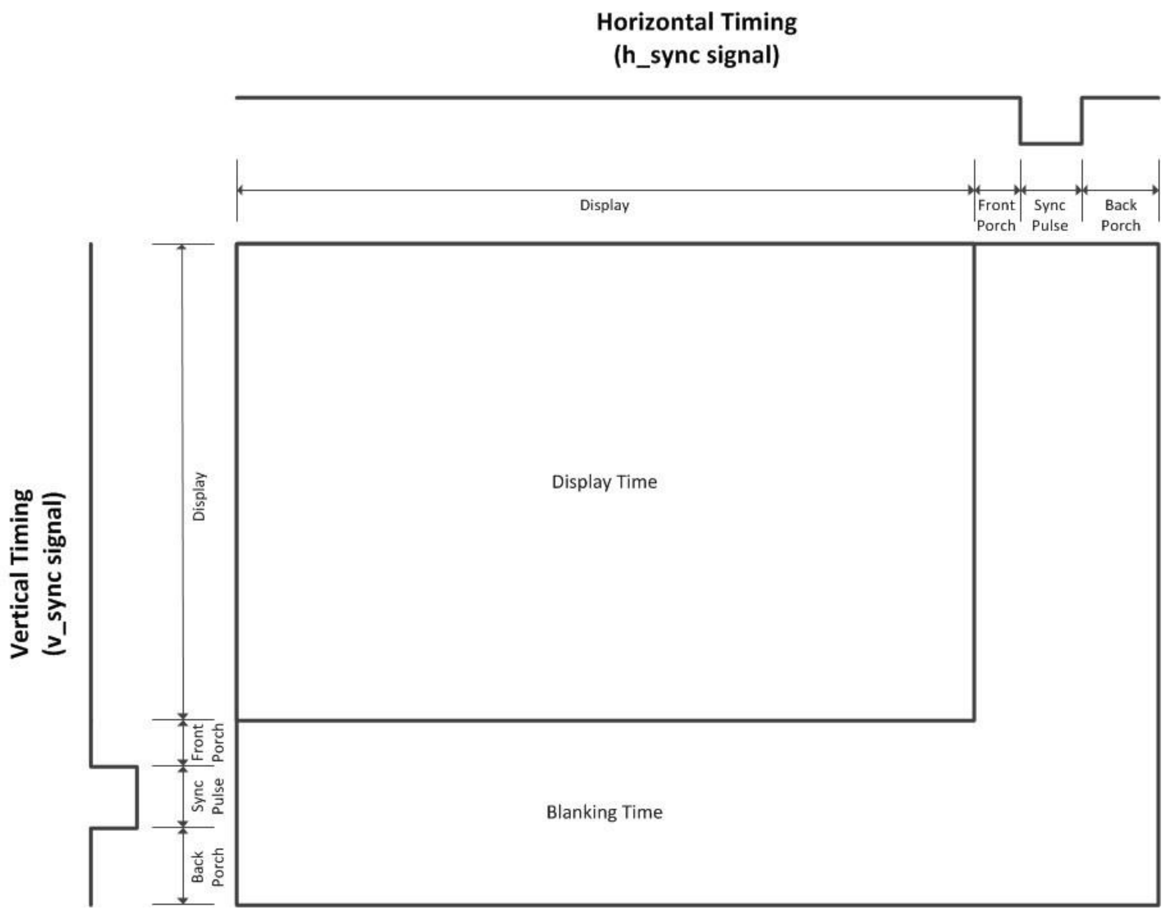

Figure 4.2: Off Screen Space and Sync Pulse Visualization [2]

Now that we understand why the sync pulses (HS and VS) exist, and that they exist off screen, the question then becomes when they are asserted. In order to find that out, we can look at this handy table [1]:

Horizontal Timing (line)			**Vertical Timing (frame)**		
Scanline Part	**Pixels**	**Time** (μs)	**Frame Part**	**Lines**	**Time** (ms)
Visible Area	640	25.422045680238	Visible Area	480	15.253227408143
Front Porch	16	0.63555114200596	Front Porch	10	0.31777557100298
Sync Pulse	96	3.8133068520357	Sync Pulse	2	0.063555114200596
Back Porch	48	1.9066534260179	Back Porch	33	1.0486593843098
Whole Line	800	31.777557100298	Whole Frame	525	16.683217477656

Note: The porches just mean space before and after the sync pulse during which the output is held black (R=G=B=0)

From these tables, and the info above, we can ascertain the following:

- Our timing generator is composed of two counters:
  1. A horizontal counter that counts from [0, 799]. It increments every clock tick when enable is **1**.
  2. A vertical counter that counts from [0, 524]. It increments every clock tick when enable is **1** and the horizontal counter has reset to **0**.
- During the time that our Horizontal counter is between [0, 639], and our Vertical counter is between [0, 479], our display is ON and we output color data. Otherwise we output black (R = G = B = **0**).
- During the time that our Horizontal counter is between [656, 751], our 'HS' signal is **0**. Otherwise it is **1**.
- During the time that our Vertical counter is between [490, 491], our 'VS' signal is **0**. Otherwise it is **1**.

1.2 Tasks

- Create an entity called "vga_ctrl" that takes as input a clock and a clock enable and produces the following outputs. It should behave according to the description above.
  - hcount: a 10-bit std_logic_vector that is the value of the horizontal counter
  - vcount: a 10-bit std_logic_vector that is the value of the vertical counter
  - vid: a 1-bit signal that is **1** when the display should be on, otherwise it is **0**.
  - hs: a 1-bit signal that is the 'HS' pulse
  - vs: a 1-bit signal that is the 'VS' pulse
- Run some basic simulation tests to check the behavior of the counters and the hs and vid signals. Do not worry about the vs signal, your simulation would have to go a very long time to witness changes on it. However, if your experience issues in the following sections, you may need to go back to this component to validate a whole frame through simulation.

2 Pixel Pusher

2.1 Background

In order to read our picture onto the screen, we need to store the pixel information into a memory block. The easiest way to do so is using the Xilinx Block Memory IP, which using hard memory elements embedded in the FPGA fabric.

In order to do so, click "IP Catalog" under the Project Manager tab. Then enter "block" into the search bar, and double click the "Block Memory Generator" IP to start the customization process.

Change the Component Name to "picture", and the Memory Type to "Single Port ROM". Then navigate to the "Port A Options" tab. Change the width to **8** and the depth to 230400. Change the Enable Port Type to "Always Enabled" and un-tick any output register boxes. In the "Other Options" tab, tick the "Load Init File" box and use the browse button to direct it to the COE file generated during the prelab. Then click the 'OK' button. The tool will bring up a window asking for permission to generate the required files for the IP instance. Click the

'Generate' button. Wait patiently; the process may take quite some time (around 10 minutes) depending on the speed of your machine.

Once Vivado has finished generating the IP, expand it in the project manager tab with the drop down arrow and open the main VHDL file. Using the Entity declaration found in the VHDL file, we can later instantiate it as a component in our top level design.

2.2 Tasks

- Create a block memory according to the description above with the COE file generated during the prelab section
- Create an entity called "pixel_pusher" that takes as input a clock, a clock enable, a 1-bit VS, an 8-bit pixel signal, a 10-bit hcount signal, and a vid signal. It should output two 5-bit R and B signals and a 6-bit G signal, as well as an 18-bit addr. It should have the following behavior:
  - It contains an internal 18 bit counter called addr with the following behavior:
    - Every clock tick when enable is **1**, vid is **1**, and hcount is less than 480, it increments. It resets synchronously when VS is **0**.
  - Every clock tick when enable is **1**, vid is **1**, and hcount is less than 480...
    - R <= pixel(7 downto 5) & "00"
    - G <= pixel(4 downto 2) & "000"
    - B <= pixel(1 downto 0) & "000"

    Otherwise, R, G, and B are **0**.
- Draw a block diagram for a top level design that contains instances of your block memory "picture", "pixel_pusher", "vga_ctrl", and a "clock_div" modified to produce a 25 MHz clock enable. It should take as input a clk signal and produce as output single bit vga_hs and vga_vs signals, 5-bit vga_r and vga_b signals, and a 6-bit vga_g signal.
- Create a top level entity called "image_top" that implements your block diagram.
- Create an appropriately modified XDC file (need VGA interface) and load the design onto the board to test it.
- Create a testbench that forces a 125 MHz clock to the clk input of "image_top" and simulate a few frames. Simulation should not run too long as we are using a 25 MHz clock enable. You should verify the following:
  - HS, VS signals behave as expected
  - R, G, B signals output data while video is on, and is always zero when video is off.
  - Timing should be precise. Otherwise, this can cause offset images, rows or columns on the VGA screen.

Below is a simplified system block diagram to help with the creation of the top level design. Please flesh it out with all of the proper signal wires and bit widths for your drawn version.

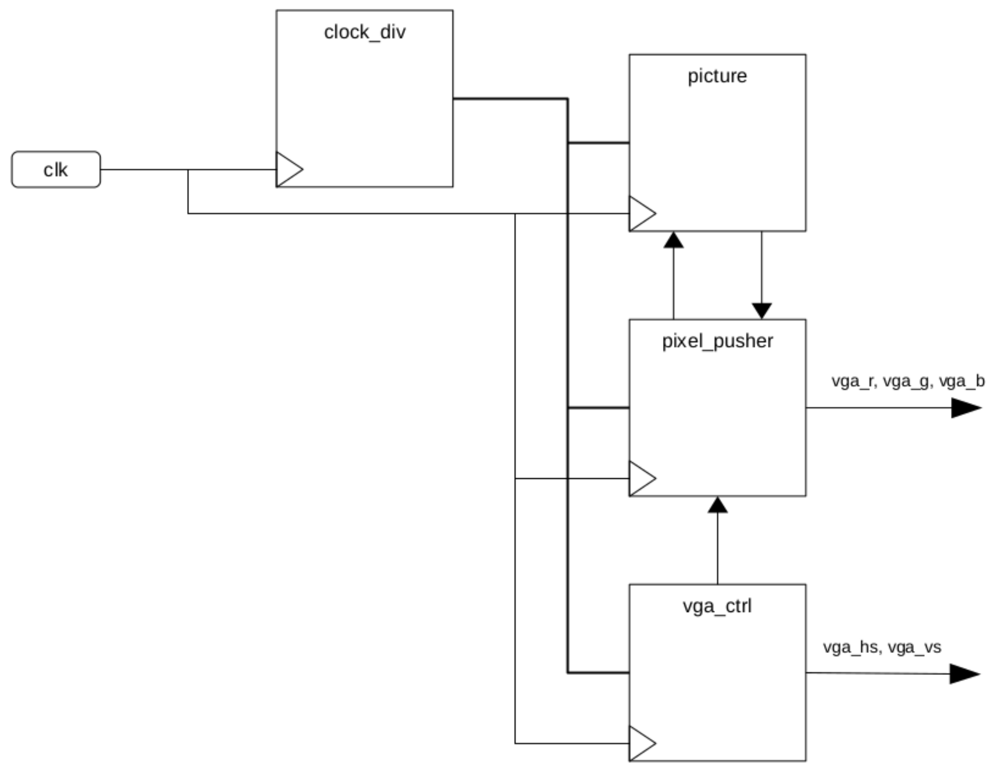

Figure 4.3: Simplified Block Diagram

2.3 Testing

Once your design has been loaded onto the board, you will need to connect a VGA cable from the port on the Zybo to a computer monitor. Because the monitors in EE103 do not have VGA inputs, we will be using ones not part of the normal lab setup.

When your board is connected, you should see your image used during the prelab Matlab conversion displayed on the screen, positioned at the top left area of the monitor.

3 Extra Credit

The VGA protocol is being phased out of all electronic components including the latest versions of the Zybo board. In this section we will be looking at generating an HDMI interface for this project:

- Using the "IP Catalog" as before, create an IP block instance of the RGB to DVI Video Encoder "rgb2dvi". We have all the clocks and signals already available to us from prior sections. Make sure you use both PixelClk and SerialCLK as inputs – understand how they relate to each other to connect to existing signals we have already created. The reset signal will be tied appropriately so the block is always enabled. For the Zybo Rev. B board, an hdmi_out_en signal is needed, refer to the Zybo reference manual for an explanation. For the Zybo Z7, you have two HDMI ports (one for input - RX, one for output - TX) so select the appropriate one for this implementation.
- Modify or create a new instance of the "pixel_pusher" design block, where you need to resize the RGB output signals to be 8 bits each. Think of how to fill in the missing bits, and/or factor the bit information sizes to maintain the graphical information (clarity, brightness etc…).
- Create an "hdmi_top" design block that instantiates the rgb2dvi IP, modified pixel_pusher and connect all of these elements appropriately. You will benefit by creating a block diagram that shows these interconnections.
- As before, run a simple testbench for the hdmi_top block, showing all the TMDS protocol outputs.
- Program the Zybo board, connect the HDMI port to one of the screens in the lab and demo by displaying your image on the screen.

Hint: No blocks are being removed from what we created before. For more helpful information, you can read the rgb2dvi IP core user guide posted on the link below,
https://github.com/Digilent/vivado-library/blob/master/ip/rgb2dvi/docs/rgb2dvi.pdf

4 Credits

Sources

[1] *Horizontal and Vertical Timing. url: http://tinyvga.com/vga-timing/640x480@60Hz.*

[2] *VGA Timing. url: https://i.stack.imgur.com/jURiy.jpg.*

[3] *ZyBo Reference Manual. Feb. 27, 2017. url: https://reference.digilentinc.com/reference/programmable-logic/zybo/reference-manual.*

Acknowledgments

Phil Southard, Milton Diaz
Part Time Lecturers

Gregory Leonberg, Original Author, Fall 2017

Steve DiNicolantonio
Updates/Modifications, Fall 2018