VGA

Introduction

In this mini project, you will be implementing a display unit and connect it to the monitor using VGA port. The display unit is commonly used in computer systems having anything to do with display. This includes computers, laptops, cellphones, and game boxes and so on. In this mini lab, you will drive the image from a ROM programmed with a specific pattern.

The main objectives of this mini project are

1. Learn about video signals that may aid you in finding a good project
2. Get an important reusable module implemented in an efficient way
3. Allow you to work as a team to help you choose your teams better

VGA Design

VGA functional description

This section defines the blocks that need to be implemented in this mini project. Figure 1 shows the block diagram of the project. The initialization (init) file will be provided and should be present in the mini project 2 starter files. In addition, the UCF file is also provided, but you may need to update it based on the signal names used in your project.

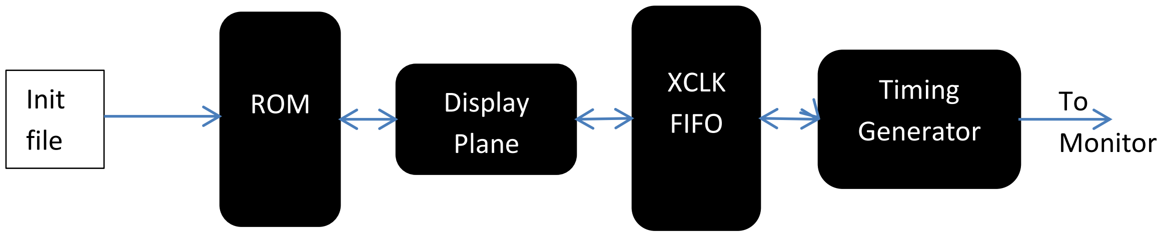


Figure 1: Block diagram of the project.

The output video is fixed at a standard VGA resolution of 640x480 @ 60Hz. The provided init file has the data as

Red = Bit 23 through 16

Green = Bit 15 through 8

Blue = Bit 7 through 0

ROM

Implement ROM using CoreGen and initialize it with the file provided. If we have 3 bytes per pixel and 640x480 pixels, we will require 921600 bytes (307200x24bits) of memory storage. The provided init file only provides initialization for a ROM of size 4800 entries. Each pixel is 24 bits (8 bits per Red, green and blue). To generate the output resolution of 640x480 pixels from the provided 80x60 frame, you may do one of the following:

1. Pixel multiply (repeat pixels horizontally and vertically for 8 times. This can be done in the display plane or the timing generator or both – depending on how you want to implement it. Contact your TA if you need more clarity on this.
2. Generate your own ROM init file for the full size (compilation may take a long time) and implement the logic without any pixel replication logic.

Display Plane

Display plane is responsible for generating the addresses for pixels to fetch and putting them in the XCLK FIFO. This logic block has to run at 100MHz. Make sure you take care of the XCLK FIFO's full and empty signals to start or stop fetching data from the ROM. Increase the idle time of this unit to save power.

XCLK FIFO

The cross clock fifo acts as a buffer between the 100MHz core clock domain and 25MHz display clock domain. Although much design of the size of FIFO is not required in this case, you have to take care of the minimum size of the XCLK FIFO when the clock frequencies may differ (100MHZ core clock and 80MHz display clock). In this project, we will just use a 16 entry buffer. Use CoreGen to generate this block.

Timing Generator

Timing generator generates the required sync and blank signals and aligns pixels according to these signals. The outputs to the VGA port are

1. Red
2. Green
3. Blue
4. Hsync
5. Vsync

The blanks are used to read the data from FIFO and put them on the Red, green and blue data lines. The description of these signals is as below. Figure 2 &3 show the timing diagrams required for VGA.

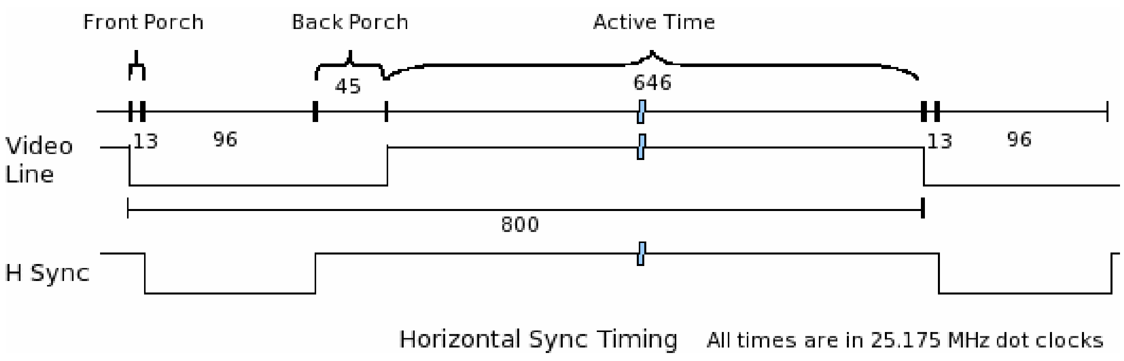


Figure 2: Horizontal Sync timing, © 2008, Mac A. Cody

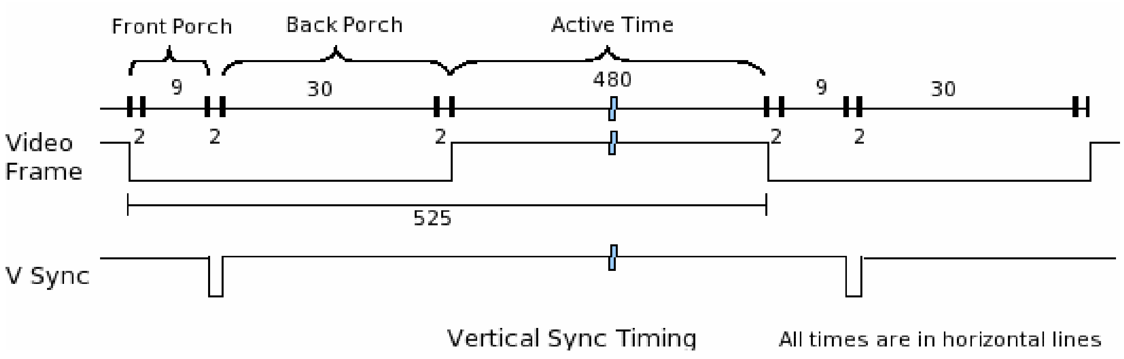


Figure 3: Vertical Sync Timing, © 2008, Mac A. Cody

More Information

VGA resolution is very well documented and widely used standard. You may reference them, but the implementation has to be your own.

Testing

Unit testing and full pipe level testing is expected from this project. You should annotate the simulation results in the report.

For the demo, you need to show the TA that your code compiles and you can burn the bit file on to FPGA and the screen should display the test image (or the image that you chose to display, if you want you create and use your own bit file).

Report:

Your report should have -

1. Well commented Verilog code,
2. Description of your implementation and test bench
3. Problems encountered and solutions for those problems (if you solved them)

Mini Project 2–Starter Guide

September 15, 2015

Contents

In this mini assignment we need a total of 5 Verilog files. We also need the .ucf file for pin assignment. These files are listed below. Note, they exclude the two IP cores (FIFO and ROM) that should be generated using the core generator.

1. dvi_ifc.v
2. vga_logic.v
3. vgamult.v
4. main_logic.v
5. display_plane.v
6. vgamult.ucf

In the above list, files #1 and #2 can be directly copied from the tutorial files; they don't need to be modified. To access the tutorial file, see "Design Tutorial for Virtex 5" under Topic 5 on the course website.

Files #3 and #4 may be copied from the tutorial but need to get modified for this mini project.

File #5 may be written from scratch. Alternatively, the file draw_logic.v from the tutorial file can be modified. We recommend to implement a variation of #5 that can read from ROM, scale the data for the higher resolution, and then write into FIFO.

File #6 is the .ucf file and available on the course website under mini project.

The total lines of code that needs to be written is less than mini project 1. So it may be worth-while to go through the tutorial files, and make modifications to the tutorial files as suggested above to complete mini project 2.

The main objective of mini project 2 is to learn creating IP modules using core generator as well as making use of FIFO to interface two clock domains with different frequencies, i.e., 100MHz and 25MHz in our case.

Kai Zhao
John Roy
ECE554 Miniproject 2 Report

Implementation

Our top level module for our design is called vgamult. Vgamult contains a clock divider to generate a 25 MHz signal for VGA from the 100 MHz chip signal, a 16 deep FIFO to release pixel data into the VGA 25 MHz clock domain from the 100 MHz clock domain, a main logic unit, a module for the logic for VGA, and a module for logic to convert VGA to DVI.

The VGA logic inside vgamult was provided for us. It gives vgamult the coordinates of the pixel it is preparing to send along with signals necessary for VGA, vsync, hsync, and blank. In return, vgamult gives the VGA logic unit the color values for the pixel it requests. The DVI unit was also provided for is. It translates the VGA signals into DVI signals for display.

The main logic unit is just a container for the display plane. The display plane has a ROM in it containing an image file. The unit takes in the pixel_x and pixel_y values from the VGA logic unit and translates them into an address to fetch the RGB values for that pixel from the ROM. The data width in the ROM is 24 bits which gives each of red, green, and blue 8 bits of depth. There are 4800 entries in the ROM, giving an 80x60 image.

In order to convert the 80x60 image into a 640x480 image, we take the pixel_x and pixel_y values that correspond to a screen pixel and only use the top 7 of 10 bits of each to create an address, ensuring that each color value in the ROM is repeated for 8 screen pixels in the x direction and 8 screen pixels in the y direction.

Problems

The first problem we had was that our storage wasn't initialized with the .coe file, so when we tried to display the image we only got a blank screen. In order to fix this, we ensured that our storage was ROM instead of RAM. When we did this the initialization worked and we were able to see an image.

The second problem we had was that we could get the image to appear when we bypassed the FIFO and wrote directly from ROM to VGA, but we couldn't get an image to appear when we passed the data through the FIFO. We made sure that the proper clocks were going to the proper modules in order to fix this. We could then see an image.

The third problem we had was that a single vertical line of incorrect data appeared on the screen. We found that this problem originated from the fact that we weren't writing any data to the screen when VGA's blank signal was asserted. Our signal for blank was coming from the 25 MHz clock domain, but it was being applied to the side of the FIFO in the 100 MHz clock domain. Once we gated the signal to be in the 100 MHz clock domain, the vertical line of incorrect data disappeared.

The fourth problem we had was that the image was shifted to the right. We only had about 34 instead of 48 pixels after the right of the flower. We incremented the pixel_x signal by 14 to obtain the offsetted ROM address.

The fifth problem we had was that the first few columns of the first row was not displaying the image. We fixed this by modifying the pixel_y that goes into the ROM address to 0 if it will go out of the visible range.

One final problem we faced was that some data from the right side of the screen was being displayed on the left side of the screen. We tried delaying the pixel_x and pixel_y signals being sent to the VGA to account for the delay through the FIFO. We also tried reading ahead in the ROM to account for the delay through the FIFO, but this has not worked either. We had to reset the FIFO on negative edge of blank, and enable write when the pixel_x he about to wrap around, which effectively will have the data in the fifo by the time the pixel wraps around to the 1st column.

Testbenching

In order to testbench our design, we created an image file that displays a spectrum of colors on the screen. This image allowed us to clearly see when pixels are in incorrect places on the screen as they will stand out against the color spectrum. The spectrum consists of 3 bands. The top band has a red value of 40, the middle has a red value of 120, and the bottom has a red value of 200. Within each band, the green increases continuously from top to bottom and the blue increases continuously from left to right.

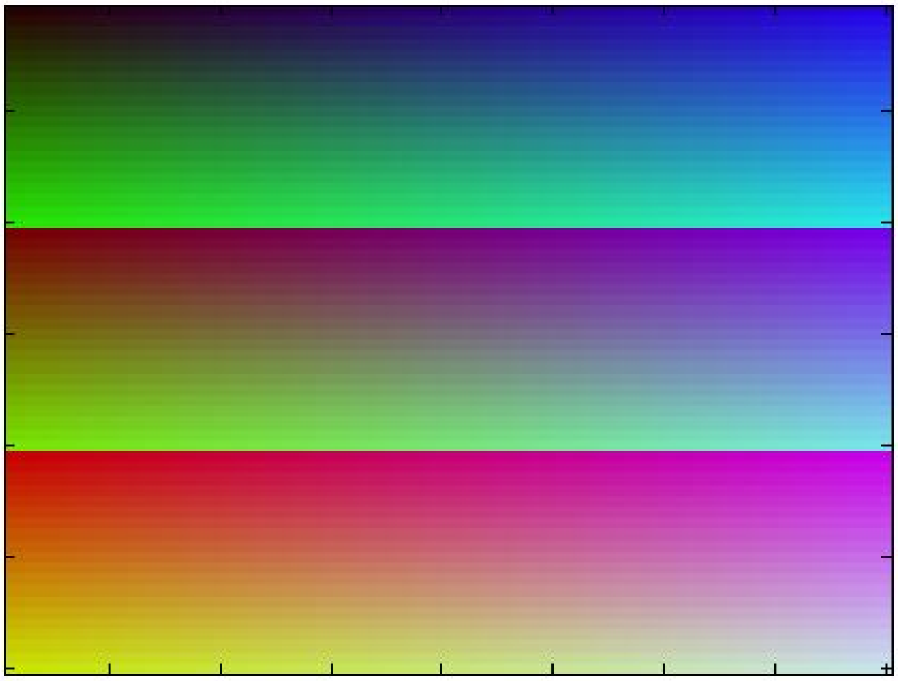
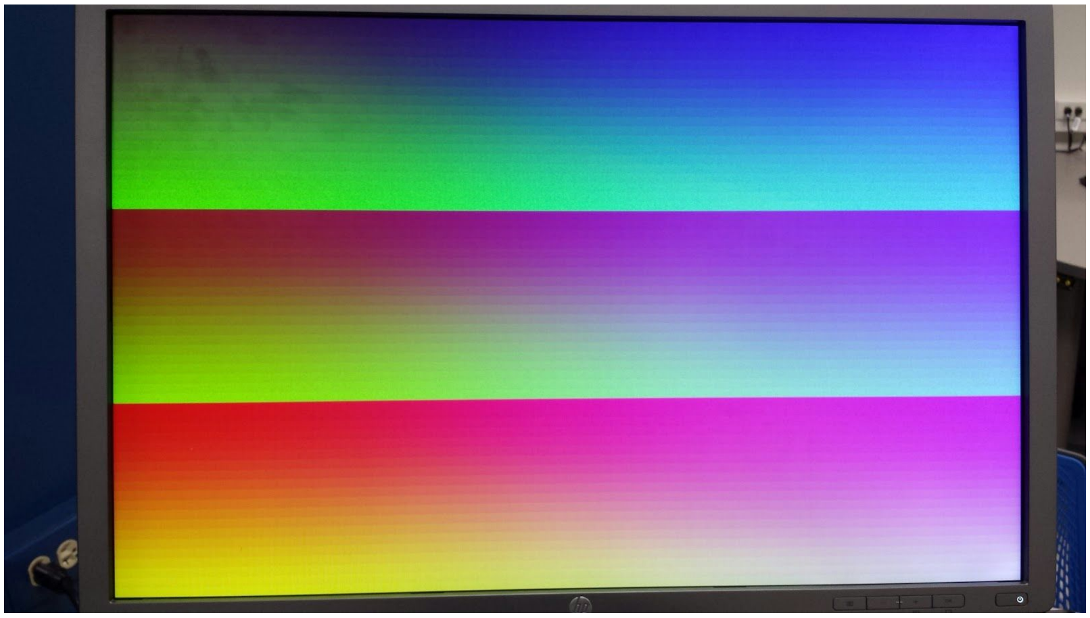
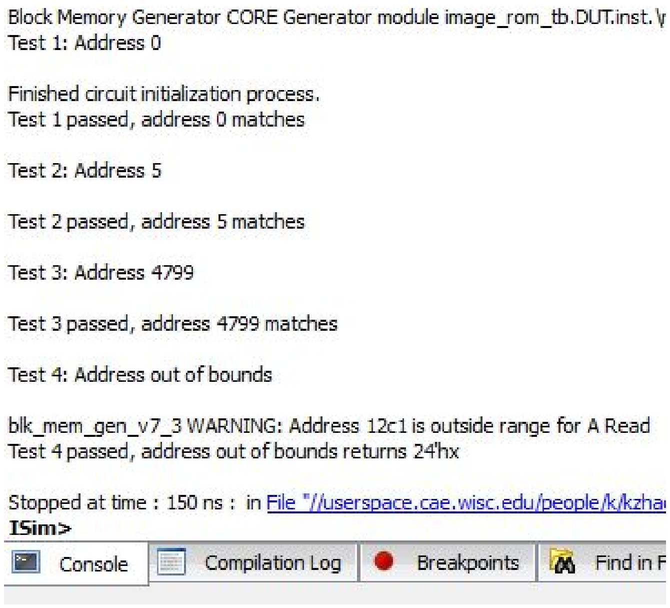


Figure 1. Image of spectrum testbench



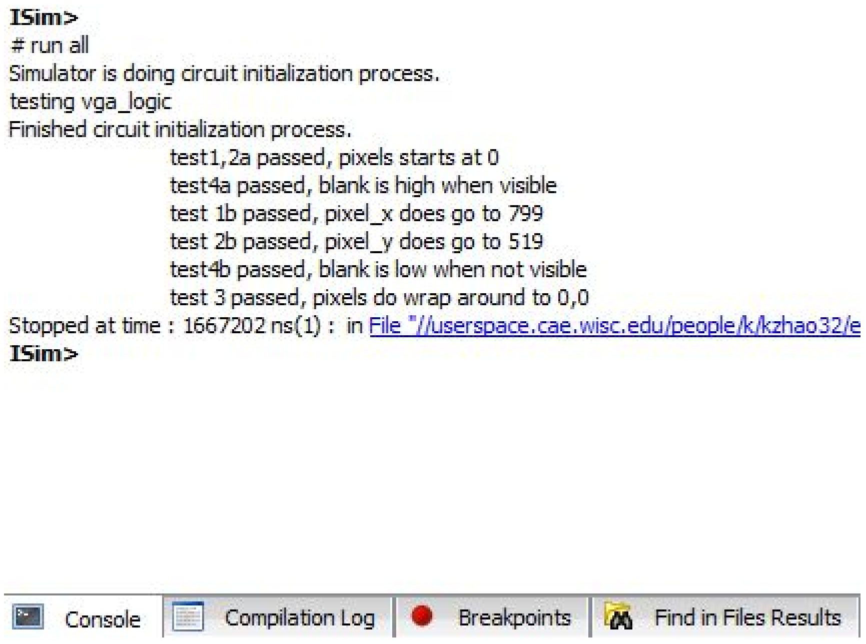
Another form of testbenching we did was viewing the waveforms and outputs of unit test benches inside the ISE module. This was particularly useful in determining whether each individual module was behaving correctly signals from the VGA unit should go to the FIFO so that it can be read from or written to.

image_rom_testbench test address 0, address 5, address 4799, and address out of range



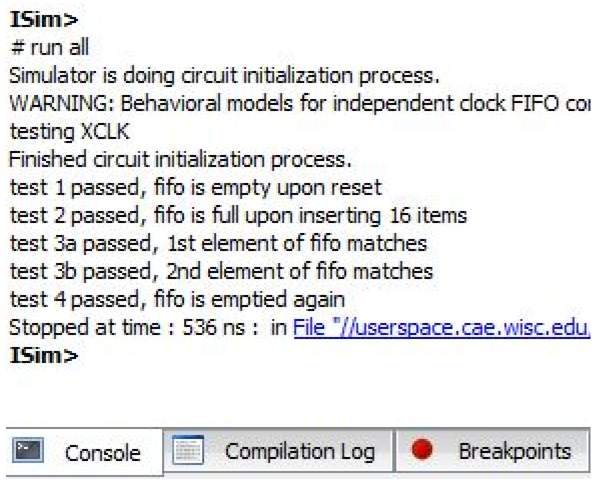
```
Block Memory Generator CORE Generator module image_rom_tb.DUT.inst.\
Test 1: Address 0

Finished circuit initialization process.
Test 1 passed, address 0 matches

Test 2: Address 5

Test 2 passed, address 5 matches

Test 3: Address 4799

Test 3 passed, address 4799 matches

Test 4: Address out of bounds

blk_mem_gen_v7_3 WARNING: Address 12c1 is outside range for A Read
Test 4 passed, address out of bounds returns 24'hx

Stopped at time : 150 ns :  in File "//userspace.cae.wisc.edu/people/k/kzha
ISim>
```

VGA_logic_testbench tests that pixel_x and pixel_y loops around and blank signal is high when the pixel is visible



```
ISim>
# run all
Simulator is doing circuit initialization process.
testing vga_logic
Finished circuit initialization process.
                    test1,2a passed, pixels starts at 0
                    test4a passed, blank is high when visible
                    test 1b passed, pixel_x does go to 799
                    test 2b passed, pixel_y does go to 519
                    test4b passed, blank is low when not visible
                    test 3 passed, pixels do wrap around to 0,0
Stopped at time : 1667202 ns(1) :  in File "//userspace.cae.wisc.edu/people/k/kzhao32/e
ISim>
```

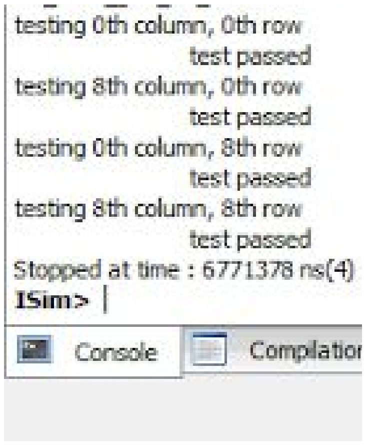
XCLK_tb tests that the FIFO is emptied upon reset, full after putting in 16 elements, the data matches when popping from the FIFO, and the FIFO empties again



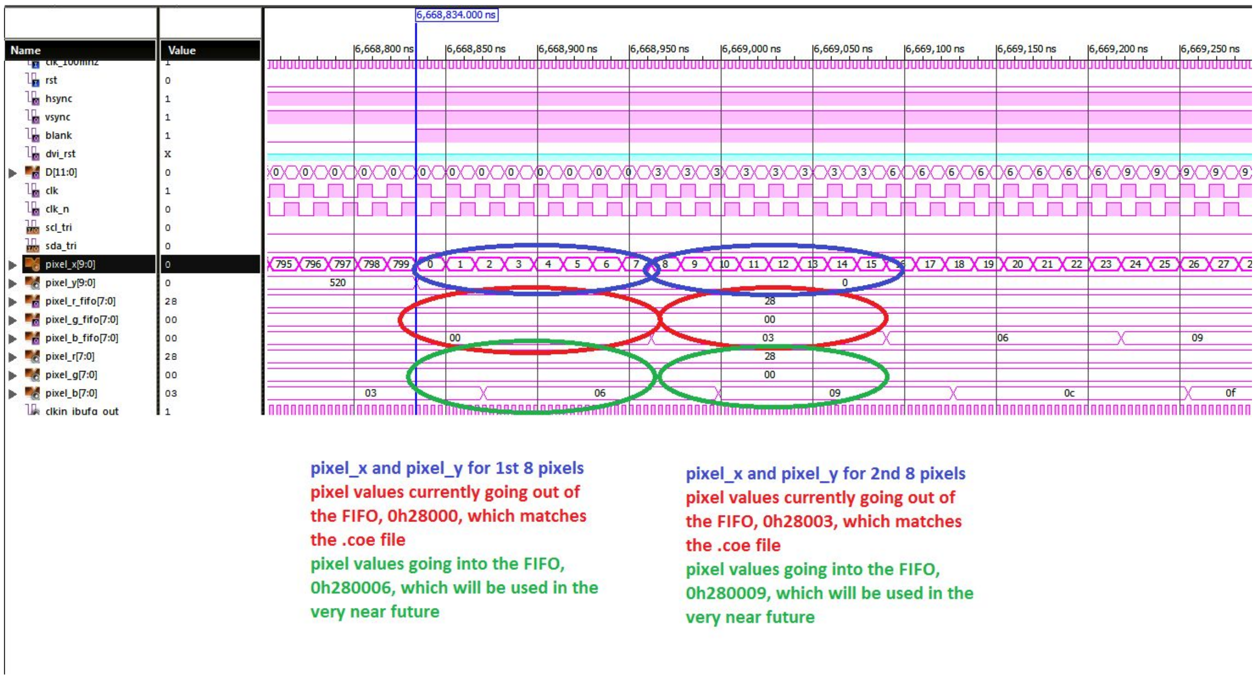
```
ISim>
# run all
Simulator is doing circuit initialization process.
WARNING: Behavioral models for independent clock FIFO co
testing XCLK
Finished circuit initialization process.
test 1 passed, fifo is empty upon reset
test 2 passed, fifo is full upon inserting 16 items
test 3a passed, 1st element of fifo matches
test 3b passed, 2nd element of fifo matches
test 4 passed, fifo is emptied again
Stopped at time : 536 ns :  in File "//userspace.cae.wisc.edu
ISim>
```

vgamult testbench results and annotated waveform to check the RGB values matches the .coe file

Finally, we wrote a Verilog testbench to verify that correct color values corresponding to a pixel location were being sent to the VGA unit. This testbench works by instantiating VGAmult, then running through an entire screen until vsync is asserted, then checking that pixel values match what they should be in the COE file that initializes the ROM. Running through the screen once ensures that any initialization effects on the first frame after reset are not noted.



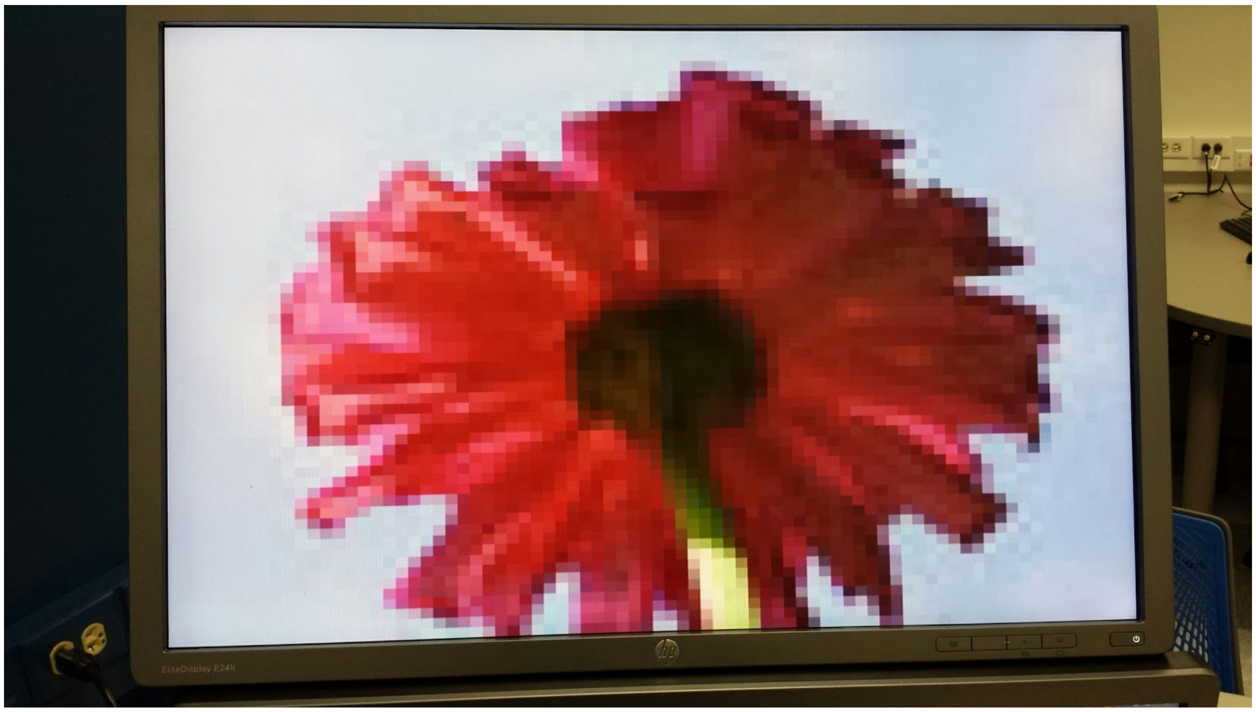
```
testing 0th column, 0th row
                    test passed
testing 8th column, 0th row
                    test passed
testing 0th column, 8th row
                    test passed
testing 8th column, 8th row
                    test passed
Stopped at time : 6771378 ns(4)
ISim>
```



pixel_x and pixel_y for 1st 8 pixels
pixel values currently going out of the FIFO, 0h28000, which matches the .coe file
pixel values going into the FIFO, 0h280006, which will be used in the very near future

pixel_x and pixel_y for 2nd 8 pixels
pixel values currently going out of the FIFO, 0h28003, which matches the .coe file
pixel values going into the FIFO, 0h280009, which will be used in the very near future

The final testing is implementing our design and check if it produces a flower output from the given tiny_image.coe file



Verilog Code

Our Verilog code is attached inside the .zip file, and it should be considerably well commented. The comments explain how to handle pixel_x and pixel_y with FIFO and ROM with wrap around so that every pixel displays correctly.

- vgamult_tb is the top_level testbench
  - vgamult is the top_level to display the image ROM to DVI
  - XCLK_tb is the individual testbench of the FIFO
    - XCLK is the FIFO for storing pixel data
  - vga_logic_tb is the individual testbench to check pixel_x and pixel_y
    - vga_logic, used to generate the pixel_x and pixel_y signals
  - vga_clk generates the 25mhz signal from the 100mhz signal
  - main_logic is used to interface between vgamult and display_plane
    - display_plane is used to fetch data from the ROM
      - image_rom_tb is used to test the ROM
      - image_rom is the ROM generated from the .coe file
  - dvi_ifc is used to connect visual output to DVI interface