CS 223 Term Project Assignment

Fall 2024

Video Graphics Array Driver and Drawing Canvas Design and Implementation

Report Due: December 16, 2024 08:00

Introduction

You are going to implement a Video Graphics Array (VGA) module in SystemVerilog, which will enable your Basys 3 board to generate high-quality graphics on a monitor. Your implementation will include designing a controller that can handle the timing and synchronization of the VGA signal, as well as controlling the pixel output on the screen. This project will allow you to demonstrate your knowledge in digital design, state machines, read/write control logic, and memory operations, all while working with a complex, high-speed peripheral like the VGA interface. Your implementation will enable the Basys3 board to generate graphics on the screen in full color and resolution, and will be fully compatible with common video monitors, such as the ones present in the lab.

Stage 1: VGA Controller [35]

Implementing a VGA screen driver requires manipulating two digital synchronization pins and three analog color pins `(RED, GREEN, BLUE)`. On the rising edge of the pixel clock, the device transmits an 8-bit color value to be displayed at the currently selected pixel. This process continues with subsequent rising edges, transmitting color information for successive pixels until reaching the end of the current row. At that point, the synchronization pin `HSYNC` signals the screen to advance to a new row of pixels. The other synchronization pin, `VSYNC`, tells the screen when to initiate a new frame, indicating the completion of an entire frame transmission. In the original specification, each VGA frame consists of 640×480 pixels and 60 frames are displayed each second. There are blank intervals called the back porch and the front porch where the pixel drawing is separated to allow for horizontal and vertical syncing of frames. This operation is controlled by the pixel clock `PCLK`, which provides the pixel clocking information. The protocol governing this process is outlined below and in Fig. 1.

1. The VGA pixel clock runs at 25.175 MHz (you can probably get away with 25 MHz)
2. `HSYNC` and `VSYNC` start with logic-level high
3. `HSYNC` remains in active mode for 640 pixel clock cycles (i.e., one row of the VGA display)
4. For each of the 640 clock cycles, the voltages on the `RED`, `GREEN`, and `BLUE` lines are varied between 0 and $0.7V$, with each voltage representing the intensity of that particular color for a particular pixel.
5. After 640 clock cycles, the `RED`, `GREEN`, and `BLUE` lines are set to 0, and the `HSYNC` line remains high through its front porch (16 pixel clock cycles).
6. `HSYNC` is set to logic-level low for 96 pixel clock cycles (this is the horizontal sync pulse)
7. `HSYNC` is set to logic-level high through its back porch (48 pixel clock cycles).
8. `HSYNC` then returns to the start of active mode (step 2, above), and the process is repeated for the next row of pixels. Each row of pixels is a line on the display.
9. `VSYNC` remains logic-level high for 480 lines.
10. After 480 lines, the voltages on the `RED`, `GREEN`, and `BLUE` lines are set to 0, and the `VSYNC` line remains high through its front porch (10 lines).
11. `VSYNC` is set to logic-level low for 2 lines (this is the vertical sync pulse).
12. `VSYNC` is set to logic-level high through its back porch (33 lines).
13. `VSYNC` then returns to the start of logic-level high mode (step 2, above), and the process is repeated for the next frame.

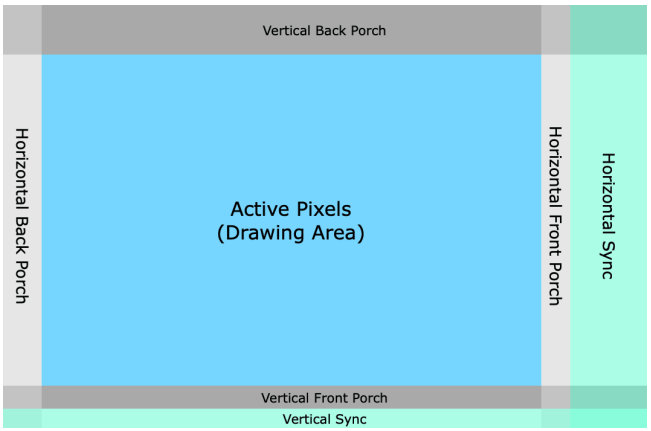

Figure 1: VGA display timings.

The sizes of pixel drawing and blank intervals for 640×480 @ 60Hz VGA signal are given to you in Tables 1 and 2. For more information on VGA specification and connector pinout, you may also check this resource [2]. You are expected to conduct your own research on the VGA specification for use in your implementation.

Table 1: Horizontal timing [1].

Scanline part	Pixels	Time (μs)
Visible area	640	25.422045680238
Front porch	16	0.63555114200596
Sync pulse	96	3.8133068520357
Back porch	48	1.9066534260179
Whole line	800	31.777557100298

Table 2: Vertical timing [1].

Frame part	Lines	Time (μs)
Visible area	480	15.253227408143
Front porch	10	0.31777557100298
Sync pulse	2	0.063555114200596
Back porch	33	1.0486593843098
Whole frame	525	16.683217477656

VGA Controller Implementation

In your VGA controller design, use the standard VGA specification of 640×480 pixel area at 60 Hz. Create a checkerboard pattern of different colors for testing purposes.

For the first stage of the project, your implementation should have a functioning VGA controller module capable of rendering a simple test pattern on the display. After you verify that you can display a test image, use the four buttons, `BTNU, pin T18`, `BTND, pin U17`, `BTNR, pin T17`, and `BTNL, pin W19` to start scrolling of the image in the corresponding directions. For example, pressing `BTNU` will cause the image to slowly "move" upwards, with disappearing pixels an the top wrapping around and reappearing at the bottom.

At the end of stage 1:

- **[25]** You can display a simple checkerboard pattern on the display
- **[10]** You can change the direction of scrolling by pressing directional buttons at any time
- You may choose to display a stationary image at the start, or select a default scrolling directions. Both approaches are acceptable.

Important Note: You can verify the functionality of your implementations by connecting a VGA cable between your Basys3 board and the available monitors in the lab.

Stage 2: Drawing Canvas Application [40]

Now that you have a functioning VGA controller, it is time to develop your FPGA-based drawing canvas application. Begin by implementing a "+"-shaped cursor that can be controlled using directional buttons. Each button press should move the cursor by one pixel in the corresponding direction. Once the cursor is working, assign **8** colors of your choice to the rightmost switches on the Basys3. Select a color by setting the switch to logic-level high. Then, use the center button `BTNC`, to "draw" the selected color at the pixel under

the cursor. Next, utilize the leftmost switch to assign a "brush" size mode. When this switch is logic-level low, the brush should draw on a single pixel. Conversely, when it's logic-level high, the brush should draw on a 3×3 area centered around the cursor. Experiment with creating basic shapes using button movement, and set the starting canvas to a purely white background.

At the end of this stage:

- You have a blank canvas with a white background on display
- You have a plus-shaped cursor that can be controlled with directional buttons
- You selected eight colors and assigned them to individual switches for selection
- You set a brush size switch that selects the paint area as either 1×1 or 3×3
- You color the canvas by pressing the `BTNC`

Stage 3: PS/2 Mouse Interface [15]

You utilized your VGA controller device to create a drawing application, now you are able to color individual pixels on a pixel canvas. However, using directional buttons to draw on the canvas can be cumbersome. It is preferable to use a more flexible device like a standard USB mouse to create your fancy drawings. The USB (type A) connector labeled J2 and the adjacent auxiliary function microcontroller (PIC24FJ128) drives several signals into the FPGA, two of which are used to implement a standard PS/2 interface for communication with a mouse or keyboard.

You already have logic in place to move a cursor around your canvas. Now, communicate with a standard USB mouse with the PS/2 interface to control your cursor's position. Create a PS/2 driver that adjusts the cursor's (x, y) coordinates based on input from the mouse. Use the left-click button instead of `BTNC` to color the pixel canvas. Color switches function the same as before.

At the end of this stage:

- **[10]** You control your cursor using wired USB mouse movements
- **[5]** You color the canvas by pressing the left-click button

Remarks

At the end, you successfully implemented and verified a drawing canvas application using a VGA controller, seamlessly incorporating fundamental digital design concepts into your work. Congratulations! Here are a few remarks that you should consider:

- You can directly implement the whole project in one go, skipping the intermediate steps. However, stages allow you to incrementally implement the project, ensuring you can easily fix issues as you encounter them. You can pinpoint and fix the problem faster if you have less components to worry about.
- You may receive partial points if your design has partial functionality; for instance, if it functions correctly only up to stage 2. In that case, you might want to include proper simulations, and testbenches for each stage in your report. This is another reason you might want to follow the project stages.
- Please remember that you will be demonstrating your designs in the assigned lab session. It is preferable if you test everything beforehand. Lab hours should be reserved solely for the demo, so it is advised to complete your work before coming to the lab.
- Do not forget to check the VGA cable for any defects while testing your project!
- **While you may choose not to, it is strongly advised that you treat Vivado warnings as errors. More often than not, they indicate potential issues. Vivado only raises errors when it encounters a fatal error, i.e., when your design is not synthesizable or when it cannot generate a bitstream. Possible logical errors might still be implementable; thus, Vivado does not consider them as errors, only as warnings.**
- **[10]** Be ready for any questions your TA may ask during and after your demo.

Project Report

In the project report, you need to submit the following:

- Cover page with your name, surname, ID, and section number
- Detailed explanation of your design and implementation along with the VGA specification
- RTL schematics and state diagram for VGA controller, drawing logic, cursor control, PS/2 mouse control
- Block diagram of each module you implement
- Testbenches (optional, mainly for partial points if everything does not work properly)
- References (if you have any)
- **Appendix containing all of your Verilog code as text, *NOT* image. This will be checked and points will be deducted if you submit *ANY* portion of your code in image format.**

References

[1] Vga signal 640 x 480 @ 60 hz industry standard timing — 60hz. `http://www.tinyvga.com/vga-timing/640x480@60Hz`. [Accessed 15-11-2024].

[2] Javier Valcarce. Vga signal format timing and specifications. `http://javiervalcarce.eu/html/vga-signal-format-timming-specs-en.html`. [Accessed 15-11-2024].