San Jose State University
Department of Electrical Engineering
EE178, Spring 2006, Crabill

Laboratory Assignment #6

Objectives

The objective of this lab is to understand and implement a simple character-based raster video display. As a result of completing this lab, you should have some appreciation for how a raster video display works. Your design will display a 50x40 grid of 8x8 characters on a standard raster display and accept input to change the characters displayed under user control. Figure 1 shows a block diagram of the project. The omnipresent clock and reset signals have been omitted.

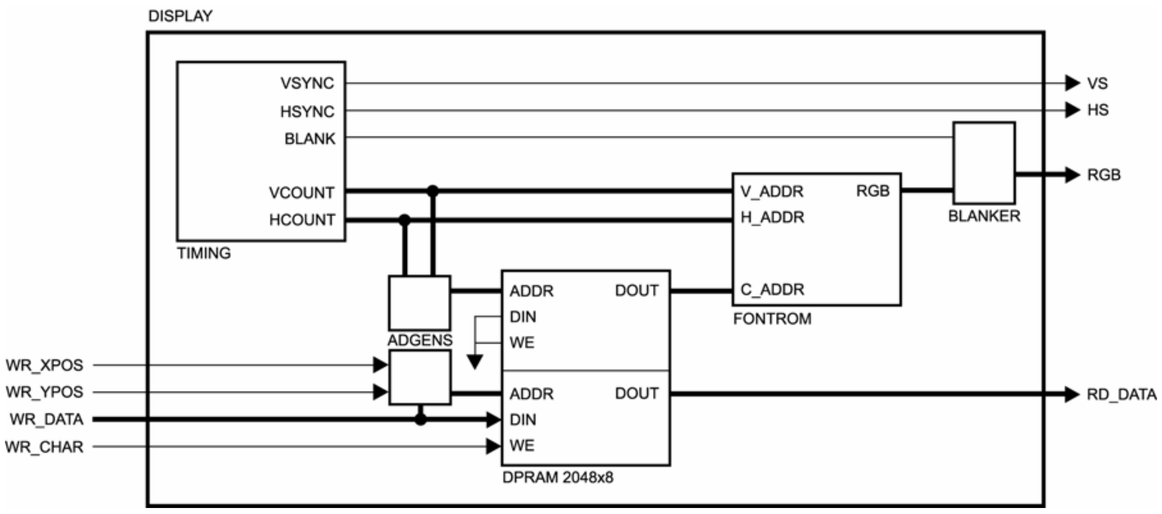


Figure 1: Project Block Diagram

The character buffer will be implemented as a dual-port RAM located on the FPGA. To access the dual-ported RAM resources, you will use vendor specific primitives. The font buffer will be implemented as a ROM located on the FPGA. When you successfully complete this lab, you will have developed a piece of intellectual property that you might be able to re-use in the future.

Bibliography

The look and feel of the 8x8 character font used in this lab is reminiscent of a mid-1980's video arcade game. I would like to give credit to Atari Games Incorporated for this inspiration. Completion of this lab requires the use of an instructor-provided font design utility. Portions of the test bench provided for this lab are based on the *TIFF Revision 6.0* specification from Adobe Systems Incorporated. The figures used in this document to illustrate video timing are reproduced from the Texas Instruments *TMS34010 Graphics System Processor User Guide*.

Video Generation

Video typically consists of a sequence of images, or frames. Figure 2 shows how a non-interlaced raster display is created by scanning a phosphor-coated screen in a regular pattern with an amplitude modulated electron beam. During the active, or visible, portion of the frame, the electron beam excites the phosphor coating on the screen, generating light. After scanning the active portion of the frame, the electron beam is returned to its starting place and the process is repeated.

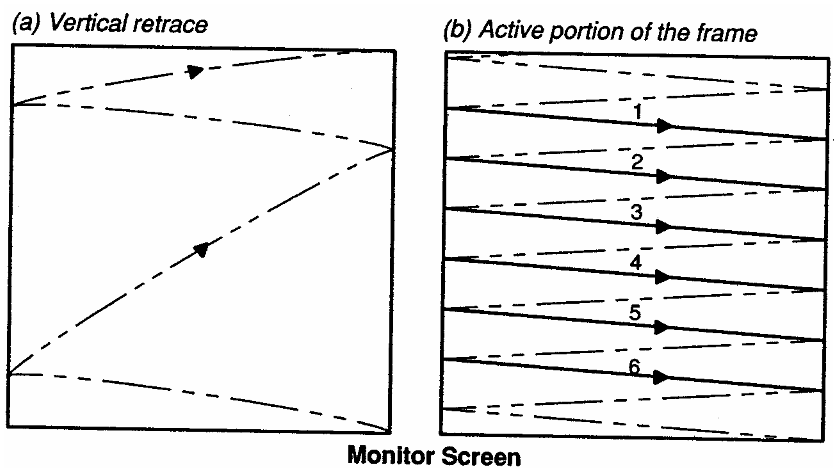


Figure 2: Non-Interlaced Raster Scanning Pattern

Common raster displays operate based on a set of inputs from a video timing controller. In addition to color information, the display needs synchronization signals, as shown in Figure 3. A complete interface may consist of five signals: R, G, B, /HSYNC, and /VSYNC.

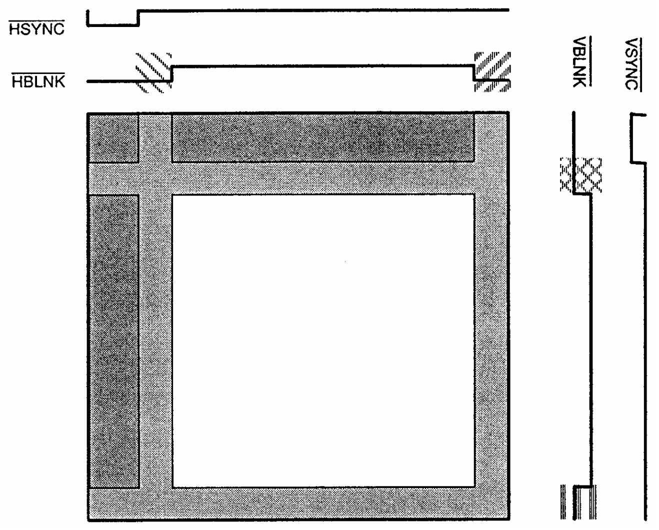


Figure 3: Sample Display Scan (Spatial)

The /VSYNC, or vertical synchronization signal forces the display to return the beam to the top edge of the display. The /HSYNC, or horizontal synchronization signal forces the display to return the beam to the left edge of the display. The two blanking signals, /VBLNK and /HBLNK, turn off the electron beam while it is retracing so that it does not perturb the active display area. The same effect may be achieved by simply turning off the R, G, and B signals during beam retrace. For this reason, blanking signals are typically not present on an interface to a raster display. Instead, the image generation circuit takes the responsibility to ensure that the R, G, and B signals are turned off during beam retrace.

Figure 3 shows a spatial representation of the synchronization signals. The white area in the middle represents time spent drawing the visible portion of the display. The actual interface is temporal in nature, and the raster display effectively performs a temporal to spatial conversion. Figure 4 shows a temporal representation of the display interface.

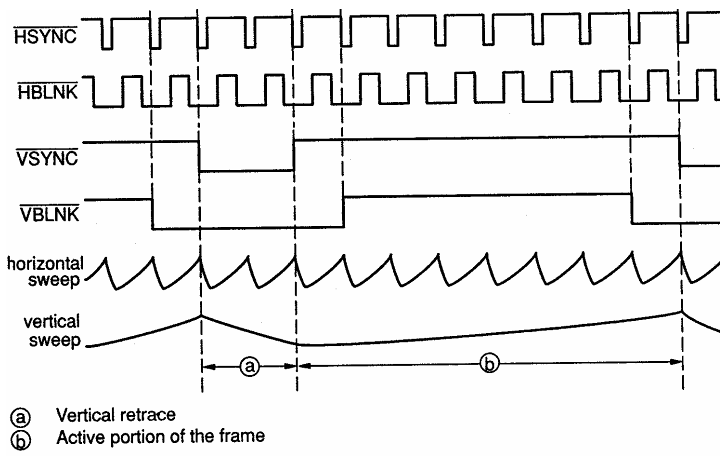


Figure 4: Sample Display Scan (Temporal)

A video timing controller is included with the downloadable support package for this lab. The video timing controller was designed for use with a 50 MHz pixel clock to create a VESA compliant 800x600 display, with a 72 Hz refresh rate. This is a common display mode supported by most monitors. Your design will make use of a 400x320 region in the center of the visible display area.

The video timing controller module relies on two counters and some associated logic. The first counter counts pixels on a given line at the pixel clock rate. The output of this counter is called the horizontal count, which is an indicator of the current beam position on the horizontal axis. Comparators watch the horizontal count output and determine the value for /HSYNC, /HBLNK, and when to reset the counter.

The second counter counts lines on a given frame. The output of this counter is called the vertical count, which is an indicator of the current beam position on the vertical axis. Comparators watch the vertical count output and determine the value for /VSYNC, /VBLNK, and when to reset the counter.

The video timing controller makes the vertical count, the horizontal count, and a single blanking signal available as outputs. The blanking signal is the logical OR of the horizontal and vertical blanking signals.

The horizontal and vertical counts, taken together at any instant in time, may be interpreted as an {x, y} coordinate of the current pixel being drawn on the display. The upper left pixel is {0, 0} and the lower right pixel is {399, 319}. The bulk of your logic design for this project involves building circuits that evaluate the coordinate of the current pixel to decide what color should be displayed. On the Spartan-3 Starter Kit board, with a single bit each for red, green, and blue, there are eight possible colors. Each pixel may be one of these eight colors. The colors are:

R	G	B	Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

Project Description and Requirements

In this design, you are not allowed to use latches. You are allowed to use only one clock and only one asynchronous reset signal. The clock must be the 50 MHz clock signal available from the oscillator on the Spartan-3 Starter Kit board. **You will receive zero points if you do not follow these requirements.**

As shown in Figure 1, the design has a number of inputs. In addition to the clock and reset, there are inputs that allow the user to change the characters to be displayed.

clk	clock signal, 50 MHz from oscillator
rst	reset signal
wr_data [7:0]	eight-bit common data input
wr_xpos	write enable for user character x position (legal wr_data values from 0 to 49)
wr_ypos	write enable for user character y position (legal wr_data values from 0 to 39)
wr_char	write enable for user character select (legal wr_data values from 0 to 255)

Also shown in Figure 1 are the outputs. One output is for feedback to the user, while the other five are for the display interface. The display outputs should be interfaced to the VGA connector on the Spartan-3 Starter Kit board.

rd_data [7:0]	eight-bit data output
vs	vertical sync
hs	horizontal sync
r	digital red output
g	digital green output
b	digital blue output

The user interface consists of a common eight-bit data input, an eight-bit data output, and three write enable signals. The desired behavior of this interface is as follows:

- Assertion of wr_xpos causes the design to store the value on wr_data in a register (xpos).
- Assertion of wr_ypos causes the design to store the value on wr_data in a register (ypos).
- Assertion of wr_char causes the design to store the value on wr_data into the character buffer at the address determined by the current xpos and ypos values.
- At all times, the rd_data output should indicate the contents of the character buffer at the address determined by the current xpos and ypos values.

Display updates are performed by writing to xpos and ypos, and then writing to the character buffer. Subsequent writes to the same row of characters do not require ypos to be updated. Likewise, subsequent writes to the same column of characters do not require xpos to be updated. The user can read back the character buffer contents by setting xpos and ypos as desired and then inspecting the rd_data output.

The video outputs generated by the design must properly drive the display and generate a solid image without flicker, snow, or other distortions.

Project Design

There is a downloadable support package for this lab. In it, you are provided with a test bench, the video timing module, and tools to generate the font ROM module. You must design everything else. Use the block diagram as a tool to understand the data flow through the design. The major sections are described below, followed by some general observations.

Video Timing Module

You are provided with a video timing module. Do not modify this module. The module provides 9-bit vertical and horizontal count outputs, in addition to sync and blank signals as previously described in the video generation section.

Dual Ported Character Buffer

The character display is a 50x40 grid of characters. Each of the 2,000 characters on the display is controlled by a unique byte stored in the character buffer. The character buffer must be constructed from dual-port BlockRAM primitives available in the unisim library. Since each BlockRAM will store 2,048 bytes, only one BlockRAM will be required, configured as a 2,048x8 dual ported RAM. The characters are to be stored in the first 2,000 bytes of the character buffer. The remaining bytes are unused.

One port of the character buffer is read-only. This port is read by a process that is looking up the correct character to be displayed based on the horizontal and vertical count values. The other port of the character buffer is read-write. This port is read and written by the user to observe and control the contents of the character buffer.

Font ROM

Each character is an 8x8 grid of pixels. Each of the 64 pixels in a given character is controlled by data stored in the font ROM. The font ROM is organized as 16,384x3, single-ported, and read-only. The 14-bit address bus for the font ROM is broken into three sub-addresses. These are:

- 8-bit character select (character image to be displayed)
- 3-bit horizontal select (column of the character image)
- 3-bit vertical select (row of the character image)

The 14-bit address selects a specific pixel in a specific character image and the output of the font ROM is a 3-bit piece of data that represents a color for the pixel to be displayed. The font ROM is read by a process that is looking up the correct pixel data to be displayed based on the character to be displayed and the horizontal and vertical count values.

You will not create the font ROM from scratch; a utility provided in the downloadable support package enables you to create/edit your own characters and generate a structural description of a ROM. When you are ready to create the ROM description, use “Export RTL Color ROM” from the file menu.

Address Generators

The address generators are responsible for generating addresses for the character buffer. On the read-only port of the character buffer, one address generator must look at the horizontal and vertical counts to determine what character should be displayed. On the read-write port of the character buffer, the other address generator must look at the xpos and ypos registers that are set by the user to determine what character should be read or written. You will find multiplication to be a useful function in the design of the address generators.

Blanker

This function is quite simple; the video output must be blanked (that is, set to zero) when the video timing controller indicates the display should be blanked.

General Observations

Before you begin writing any code, you must sit down with scratch paper and draw block diagrams of the circuits that will satisfy the design requirements. Once you have a possible solution, write a description of it in Verilog-HDL and proceed to test it in simulation.

When designing synchronous systems, it can be important for asynchronous inputs to be synchronized to the system clock. This can be done using a circuit called a synchronizer, as you saw in a previous lab. Mechanical switches, in addition to being asynchronous, also exhibit a behavior called "bounce". When contact is made in the switch, it is possible for the mechanical contacts to bounce a few times. Outside the switch, from an electrical point of view, this can look like repeated closing and opening of the switch. For this lab, do not include synchronizers or de-bouncers on the input signals. This will make it easier to re-use your design as a piece of intellectual property in future projects. We can safely ignore these behaviors because they will be difficult to observe in the video output.

You will notice that certain blocks – notably the ROM and RAM – are synchronous. As a result, the propagation of signals through them incurs a clock cycle delay. Consider them to be pipelined. As a result, you need to carefully evaluate Figure 1 to decide where else you must insert registers on the data path to make sure the design, as a whole, is pipelined properly. Print a copy of Figure 1 and pipeline it using the techniques discussed in class, making sure that one pipeline stage cuts through the character buffer and another pipeline stage cuts through the font ROM. You are advised to add one additional pipeline stage, at the output. This will help you meet the performance requirements stipulated in the implementation section. Then describe your design with the pipeline registers. This ensures the design will run at 50 MHz and the signals propagating through different paths in the design will remain synchronized with each other.

Project Verification

You must perform some minimal functional simulation of the design. This is important for two reasons. First, it will give you confidence your design is working properly before you implement it. Second, if the design does not behave as expected when you download it, you will have a mechanism to quickly create additional test cases to help debug the problem. The instructor will not help you debug logic problems (incorrect design behavior) unless you have a block diagram and are able to run a simulation.

A simple test bench is included with the downloadable support package for this lab. The test bench can be used for functional simulation as well as timing simulation after the implementation step. Feel free to enhance the basic test bench as you see fit.

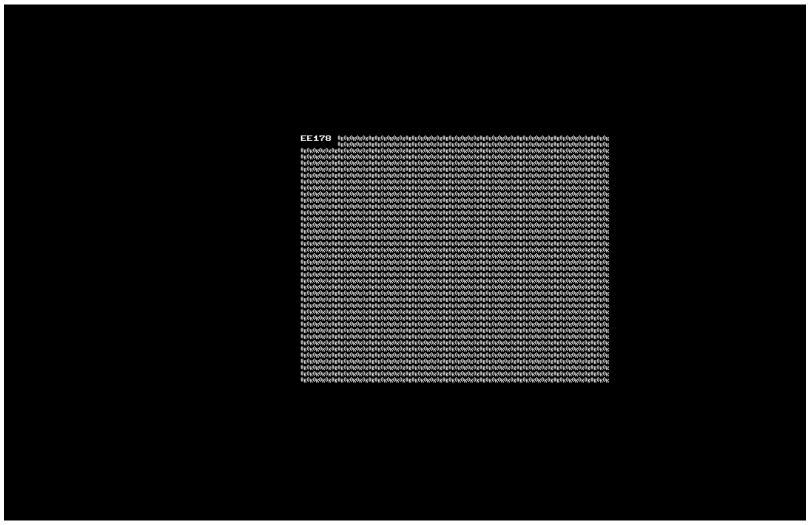


Figure 5: Sample Simulation Results

The provided test bench automatically generates TIFF output files that show your simulation results as they would appear on the display. Inspect the results *carefully*. Figure 5 shows the expected output of the unmodified test bench that is provided to you. The black bands around the four sides represent the display blanking time and are only partially visible on an actual display. Each character is displayed properly. The test bench is a great way to verify your design in simulation, although it can take a long time to simulate a full display frame. After running the simulation, you will need to look in the project directory to locate the TIFF files that are created.

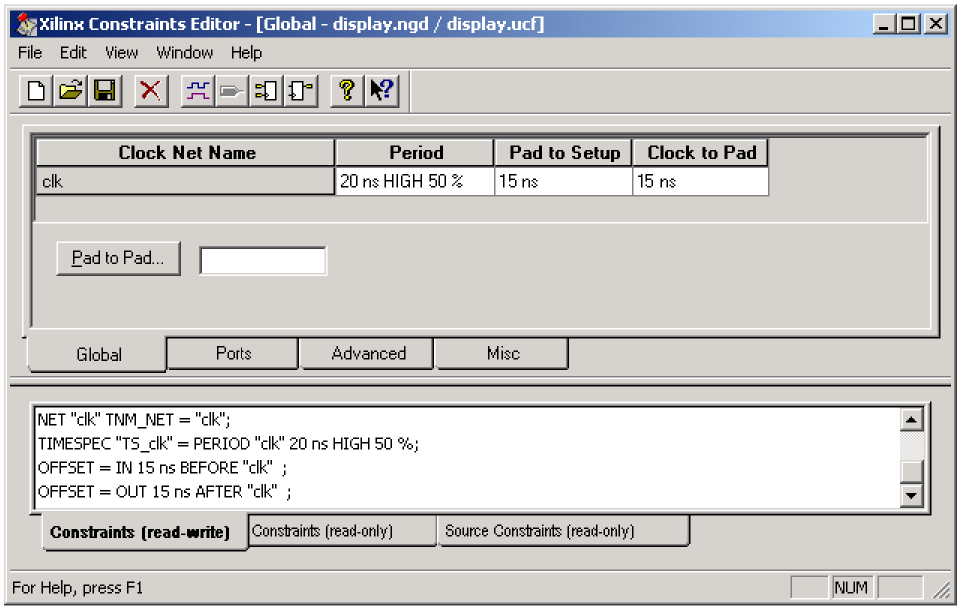
Project Synthesis

Synthesize your design exactly as you have done before. Do not forget to check the synthesis report. This report will tell you how many clocks exist in your design, under the "clock information" summary. If you have more than one clock, you need to go back and correct your design. Also check to see if any latches were used, you should not have any. These can be found in the "cell usage" summary. If you see anything starting with LD (Latch, D-type) then you need to go back and correct your design.

Project Implementation

Before you implement your design, you will need to add a constraints file and edit the I/O locations and properties. You must properly constrain the clock input. Use button 3 for reset, button 2 for wr_xpos, button 1 for wr_ypos, and button 0 for wr_char. The wr_data input should come from the switches and the rd_data output should go to the LEDs. The display outputs should be properly constrained so that they are interfaced to the VGA connector on the Spartan-3 Starter Kit board. For all inputs and outputs, do not forget to properly specify the I/O standard as LVCMOS33.

For this design, you are required to add timing constraints. After selecting the constraint file in Project Navigator, use one of the available processes to launch the constraints editor. Enter the constraints as shown in Figure 6. These constraints tell the implementation tools that your design must run at 50 MHz and that the input and output timing is bounded. The specific values used for the input and output timing are arbitrary in this instance, but represent reasonable (less than one full clock period) values.


Xilinx Constraints Editor - [Global - display.ngd / display.ucf]
File Edit View Window Help
Pad to Pad...
Global
Ports
Advanced
Misc
Constraints (read-write)
Constraints (read-only)
Source Constraints (read-only)
For Help, press F1
NUM

Clock Net Name	Period	Pad to Setup	Clock to Pad
clk	20 ns HIGH 50 %	15 ns	15 ns

```
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 20 ns HIGH 50 %;
OFFSET = IN 15 ns BEFORE "clk" ;
OFFSET = OUT 15 ns AFTER "clk" ;
```

Figure 6: Timing Constraint Entry

Verify your design in hardware. When you are satisfied with the results, generate a programming file for the PROM and then load your design into the PROM.

Laboratory Hand-In Requirements

This lab requires a written report. The report must be submitted as a professional-looking document in a single electronic file. A Microsoft Word DOC file is preferred, although an Adobe PDF file is acceptable. Paper submissions are not accepted. The body of the report must be written in English and contain the following sections:

- Title page containing your name, the lab title, and the date.
- Introduction containing a brief summary of the problem you set out to solve and your final results. For example, "The objective of Lab 1 was to implement a two-input XOR gate. I described the XOR gate, implemented it, and successfully demonstrated its operation in hardware."
- Design details documenting how you achieved your result. This is the most important part of the lab report. Illustrate that you understood the problem and explain how you solved it. In this section, you should consider using block diagrams, simple waveforms, FSM state diagrams, tables, figures, and other forms of graphical communication to clarify and strengthen your textual description.
- Simulation strategy and results documenting how you verified your design. You may include simple waveforms or other relevant test bench output.
- Final results. Describe the ultimate result of your efforts. Here, you should also include information such as maximum frequency and resource usage of your implementation.
- Conclusion containing a brief summary and what you thought of the lab. Evaluate what worked well, what did not, and how you might do it again, if you had the chance to start over. Please provide feedback on your perception of the lab difficulty and how you suggest the lab could be improved for next semester. For example, "I successfully described, implemented, and demonstrated the two-input XOR gate. My description of the logic using a continuous assignment was too terse. Next time I would describe the behavior with a verbose procedural assignment. This lab was tedious and boring, next semester you should use a two-input AND gate instead, that would really spice it up!"

Do not include project source code listings in the report. If you wish to refer to specific portions of the source code, cite a file name and a line number. This allows the reader to refer to specific portions of the source code without unnecessarily bloating the body of the report.

Once you have completed a working design and written your lab report, prepare for the submission process. You are required to demonstrate a working design which has been programmed into the PROM. Within six hours of your demonstration, you are required to submit your entire project directory and lab report in the form of a compressed ZIP archive. Your lab report must be in the project directory with the file name l6_yourlastname.doc or l6_yourlastname.pdf. Use WinZIP to archive the entire project directory, and name the archive l6_yourlastname.zip. For example, if I were to make a submission, it would be l6_crabill.zip, containing a report file l6_crabill.doc. Then email the archive to the instructor. Only WinZIP archives will be accepted. If your archive is too large, you may remove:

- The xst subdirectory (temporary synthesis files)
- The work subdirectory (temporary simulation files)
- Any file with a .wlf extension (simulation waveform files)

Demonstrations must be made on or before the due date. If your circuit is not completely functional by the due date, you should still write a report documenting what you have accomplished and turn in what you have to receive partial credit.