Lab 5: HDMI

EE478 Fall 2019

Objective

In this lab, you will learn about basic video signaling and the theory of the High Definition Multimedia Interface (HDMI). You will then display a simple shape on the monitor and use a finite state machine to change its color every one second. **It will be very helpful if you bring your own HDMI cable to the lab!** The TAs will have several cables for everyone to share, but you might be waiting around for quite a while.

You will be starting from an existing codebase that has an HDMI controller implemented for you. This controller outputs pixel coordinates and accepts red, green, and blue data as input. You will start by drawing a square on the display. You will then implement a finite state machine that changes color of the square every 1 second between four distinct colors of your choice. No board inputs will be needed for this lab.

Deliverables

Lab start: Beginning of lab session on 10/28

Demo: Working demo of a square on the screen of at least 200x200 pixels, changing colors between four colors every one second. No testbench simulation is required for this lab.

Demo due: End of the week (by 11/1) (1 week lab, flexible) [You can take more time if you need, but this lab should be simple enough to finish in one week, and you should start working on the final project directly afterwards]

Design Document due:

- On UBLearns before the end of the day on 11/8. *Submit one per team*.

Lab Contents

Part A: Existing Lab5 code

In this lab, you will be provided with an existing codebase that is already set up to control the HDMI output on the Zybo board. HDMI control is complicated because it requires the use of high frequency clock signals and a special digital signaling technique called TMDS (see Appendix A and Appendix B).

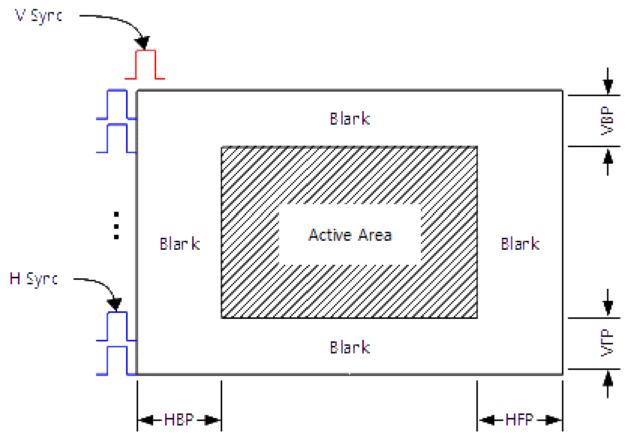
Open up Lab5.vhd and take a look at the interface for the hdmi_controller component. The following inputs and outputs are declared.

Inputs/Outputs:

- sys_clk – 100MHz system clock
- reset_btn – reset button connected to the UP momentary pushbutton
- red_data – 8 bit red color intensity, 0 = off, 255 = max
- green_data - 8 bit green color intensity, 0 = off, 255 = max
- blue_data - 8 bit blue color intensity, 0 = off, 255 = max
- TMDS, TMDSB – digital signals sent to the monitor using TMDS, including pixel clock, red, green, and blue channels
- hcount – horizontal coordinate of the next displayed pixel, counts from 0 to 1747; display data while counter is less than 1280.
- vcount – vertical coordinate of the next displayed pixel, counts from 0 to 749, display data while counter is less than 720.

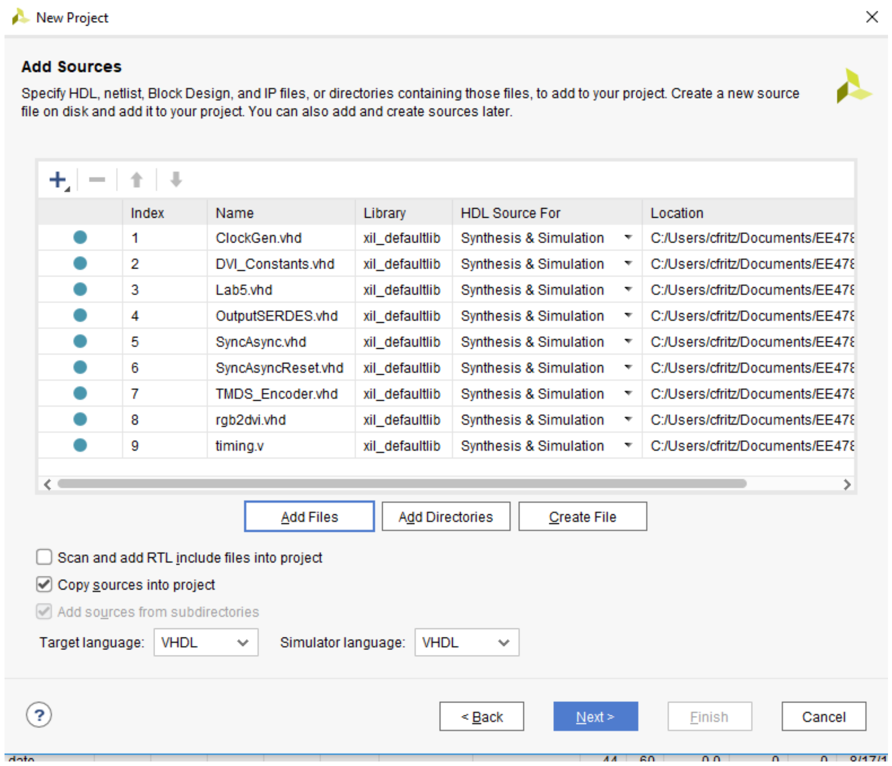
A 74.25MHz pixel clock is input to this component, and the hcount and vcount signals update at this rate. We will be outputting a 720p signal to the monitor, with a resolution of 1280x720 pixels. The output starts at the top left corner and paints one pixel at a time across each row, then moves down to the next row, and restarts from the top left after painting all pixels. (hcount, vcount) gives the x and y coordinates from the top left corner.

On each row, there are blanking regions – coordinates that do not correspond to any pixel on the display. The blanking region before the active area is called the back porch; the blanking region after the active area is called the front porch. Therefore, the hcount coordinate will go beyond 1280 and the vcount coordinate will go beyond 720. We should only display information when the coordinates are inside the display region.



Part B: Lab assignment overview

In this lab, you will be drawing a square on the display of a size of at least 200x200 pixels. You will then implement a 1Hz slow clock and a finite state machine as in lab 4 that changes the color of the square every second. Create a new Vivado project and add all of the provided vhd and v files to it:


New Project

Add Sources

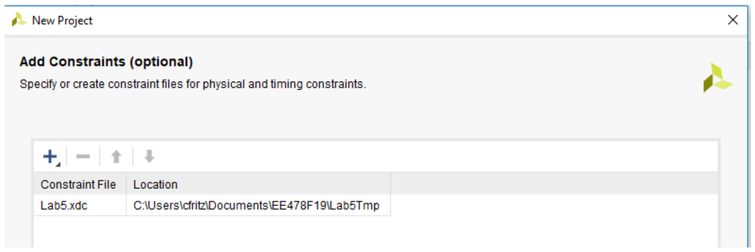
Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

	Index	Name	Library	HDL Source For	Location
●	1	ClockGen.vhd	xil_defaultlib	Synthesis & Simulation	C:/Users/cfritz/Documents/EE478
●	2	DVI_Constants.vhd	xil_defaultlib	Synthesis & Simulation	C:/Users/cfritz/Documents/EE478
●	3	Lab5.vhd	xil_defaultlib	Synthesis & Simulation	C:/Users/cfritz/Documents/EE478
●	4	OutputSERDES.vhd	xil_defaultlib	Synthesis & Simulation	C:/Users/cfritz/Documents/EE478
●	5	SyncAsync.vhd	xil_defaultlib	Synthesis & Simulation	C:/Users/cfritz/Documents/EE478
●	6	SyncAsyncReset.vhd	xil_defaultlib	Synthesis & Simulation	C:/Users/cfritz/Documents/EE478
●	7	TMDS_Encoder.vhd	xil_defaultlib	Synthesis & Simulation	C:/Users/cfritz/Documents/EE478
●	8	rgb2dvi.vhd	xil_defaultlib	Synthesis & Simulation	C:/Users/cfritz/Documents/EE478
●	9	timing.v	xil_defaultlib	Synthesis & Simulation	C:/Users/cfritz/Documents/EE478

☐ Scan and add RTL include files into project
☑ Copy sources into project
☑ Add sources from subdirectories

Target language: VHDL Simulator language: VHDL

Also, add the provided xdc file as well:


New Project

Add Constraints (optional)

Specify or create constraint files for physical and timing constraints.

Constraint File	Location
Lab5.xdc	C:\Users\cfritz\Documents\EE478F19\Lab5Tmp

Assignment 1 – Creating a PLL

In Lab5.vhd, there is a signal called “pclk” that is used to control the timing of the HDMI control. Note that this signal has no driver. Create a PLL as in Lab4 that takes in the sys_clk 125MHz input clock, and connect the PLL output to the pclk signal. Make sure to configure the output clock frequency to 74.25MHz. You can name the PLL whatever you want in the PLL wizard. Make sure you refer to the right name when you instantiate it in your code. For example, if I called the PLL pxl_clk_gen, then in the code I would instantiate it as

```
pixel_clock_gen : entity work.pxl_clk_gen port map (
```

You will need to connect to the clk_in1, clk_out1, locked, and reset pins on this instance; see the codec from Lab4 for an example.

Assignment 2 – Drawing a Square

First, calculate the coordinates of the top left and bottom right corners of the square on the display, keeping in mind that the top left corner is (0,0) and the bottom right corner is (1279, 719). Then you can use a combinational process (already started for you) to drive the red_data, green_data, and blue_data to nonzero when the hcount and vcount are within the square's region, and 0 otherwise, using **if** statements.

Before moving on, you may want to synthesize the design and make sure you see the square on the board as expected. The HDMI TX connection on the Zybo board is next to the ethernet jack; be sure not to plug in to either HDMI RX connection.

Assignment 3 – State Machine

We will implement a finite state machine that works as follows. First, create a slow clock as we did in Lab 4. Then, declare constants for red, green, and blue values for four different colors. You may find this list of hexadecimal colors useful:

http://cloford.com/resources/colours/500col.htm

For example, "yellow1" is 0xFFFF00 so if the first of our four colors was yellow, we could declare constants for this color as

```
constant COLOR1_RED   : std_logic_vector(7 downto 0) := x"FF";
constant COLOR1_GREEN : std_logic_vector(7 downto 0) := x"FF";
constant COLOR1_BLUE  : std_logic_vector(7 downto 0) := x"00";
```

Once you have four colors, implement a state machine with four states that automatically moves around the states on the rising edge of your slow clock. Then, change the color based on the state:

```
process(hcount, vcount, state)
```

If you use a combinational process, don't forget to assign values for red_data, green_data, and blue_data in every branch! Don't forget the outside "else" clause: to assign all three colors to 0 when you are not inside the square.

Part C: HDMI System Requirements

1. The system shall display a square of size at least 200x200 pixels on a monitor using the HDMI interface.
2. The color of the square shall change to four different unique colors.
3. The color of the square shall change one time per second.

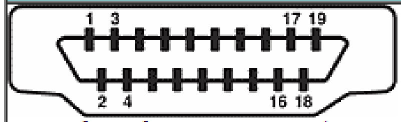
Once you finish the lab, consider adding Timing Constraints for both sys_clk and pclk in your xdc file! It's not required but it would be good practice.

Appendix A: High Definition Multimedia Interface

HDMI is a purely digital audio/video interface for sending uncompressed video data from a controller to a display. Founded by several companies including Hitachi, Panasonic, Phillips, and Sony, HDMI has become incredibly popular as a replacement for analog video interfaces like VGA, component video, or composite video.

HDMI makes use of a proprietary 19 pin connector:

HDMI



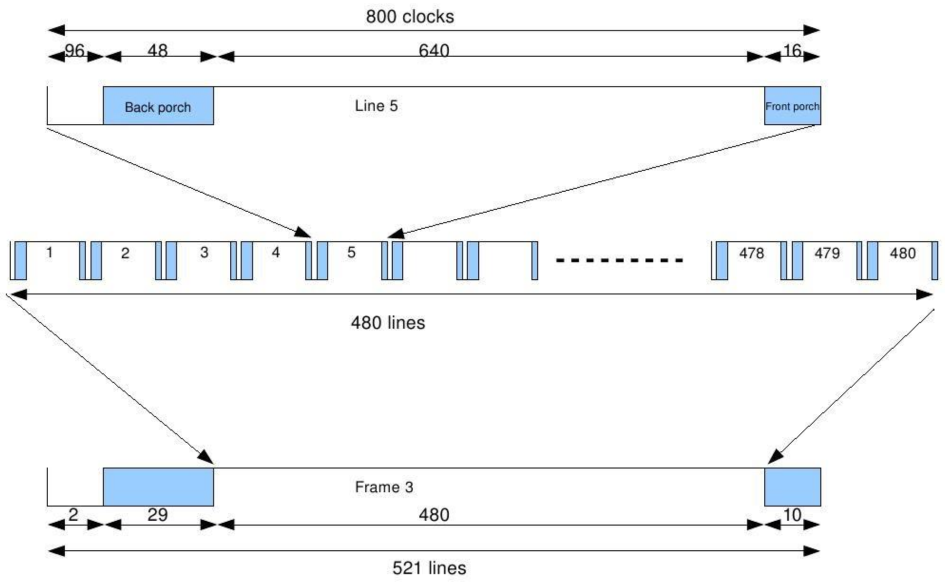
Mating face of HDMI Type-A male

PIN#	SIGNAL	PIN#	SIGNAL
1	TMDS Data2+	11	TMDS Clock Shield
2	TMDS Data2 Shield	12	TMDS Clock-
3	TMDS Data2-	13	CEC (not used)
4	TMDS Data1+	14	Reserved (N.C. on device)
5	TMDS Data1 Shield	15	SCL
6	TMDS Data1-	16	SDA
7	TMDS Data0+	17	DDC/CEC Ground
8	TMDS Data0 Shield	18	+5V Power
9	TMDS Data0-	19	Hot Plug Detect
10	TMDS Clock+		

Three channels of video data (called TMDS Data{0, 1, 2}) are used for red, green, and blue data. A pixel clock is sent as well, and the color data updates each pixel clock period. HDMI is more difficult to control than VGA because the parallel pixel RGB data needs to be serialized and sent one bit at a time at a much higher rate (in VGA, the pixel data simply has to be delivered to the DAC once per pixel clock).

Appendix B: Basic Video Signaling

Transmitting video data from a controller to a display is largely the same process whether an analog or digital interface is used. Pixels are “painted” from the top left corner to the bottom right corner of the display. Before each row of pixels, a blank region called the horizontal back porch is included; similarly a blank back porch follows the row. This is also true in the vertical direction. For example, the signaling for a 640x480 resolution video signal would be



The frequency of the pixel clock is easy to compute. For example, in this lab we send 720p video data, which has a resolution of 1280x720 pixels. Including all of the blanking regions, the number of pixels actually sent per frame is 1648x750. We update this at 60Hz (60 frames per second), so the pixel clock frequency is

$$(1648 \times 750)\frac{\text{pixels}}{\text{frame}} \times 60\frac{\text{frames}}{\text{sec}} = 74{,}000{,}000\frac{\text{pixels}}{\text{sec}} = 74\text{MHz}$$

Appendix C: Transition Minimized Differential Signaling

HDMI sends binary data over the red, green, blue, and clock channels using Transition Minimized Differential Signaling. Differential signaling is a signaling scheme in which information is sent on two separate lines as the difference of the two signals. The receiver subtracts the two signals. The two lines are twisted together, causing noise and interference to affect both signals approximately equally, so the difference of the two signals effectively cancels the noise.

Transition minimized signaling is an attempt to minimize the number of 0 to 1 or 1 to 0 changes. 8 bits of data are encoded as 10 bits with a minimum number of changes over the entire 10-bit sequence. This is done by computing either the XOR or XNOR of each input bit with the previous bit. By minimizing the number of transitions in the digital signal, the entire system is more reliable and less susceptible to errors from skew and delay over the channel.

Extra Credit

- For 10 points of extra credit on this lab, draw a circle instead of a square.
- For 20 points, change both the color and the shape every second to four different shapes.