# Lab 5 Combinatorial Logic on FPGA’s

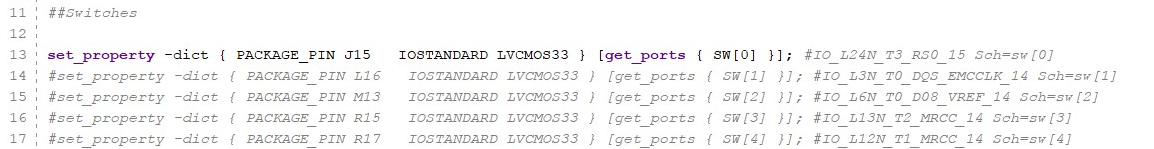
Goals:

* more practice implementing combinatorial logic with FPGA hardware.
* How do you read inputs on the FPGA board?
* How do you write outputs on the FPGA board?

Problem

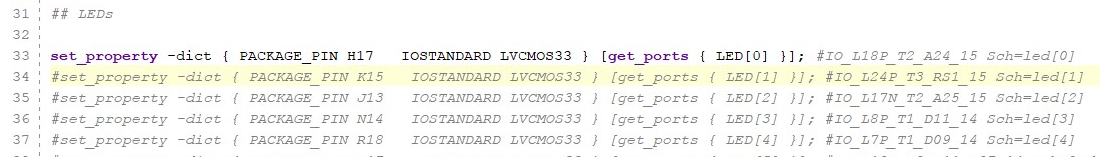
You need to implement a digital comparator with 2 x 2-bit inputs and three outputs. The inputs should be 4 digital switches on the bottom of the FPGA board. The outputs will be 3 LED’s, or RGB LEDs, or digital pmod pins.

## Workplan – Part A

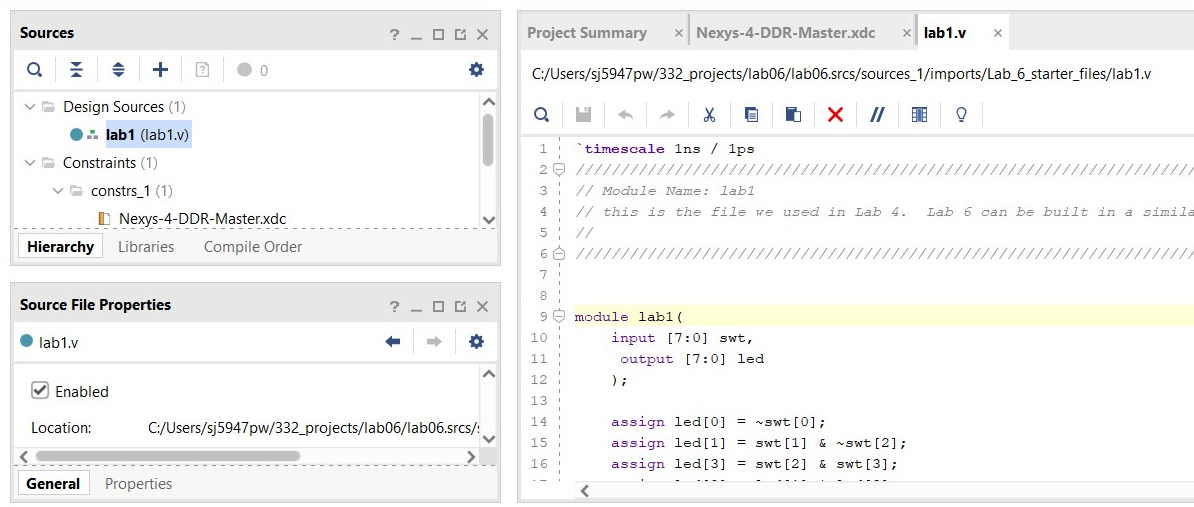
1. Open Vivado and create a new design. Or, copy the design files from Lab 4 into a renamed folder.
   1. The project should be an ”RTL project”
   2. You don’t need to specify sources (but you could use the one from lab 4 as a template)
   3. You should specify the Nexys4DDR constraints file (something like lab1\_nexys4\_ddr.xdc)
   4. You’ll need to select the FPGA architecture/chip/board. This can be automated (see the Appendix), or, as in Lab 4, you can specify the specific FPGA chip number.
2. Open the constraints file and look for the input switches. They’ll be something like the following: 

Uncomment the switches (remove the #) that you want to use and save the file.

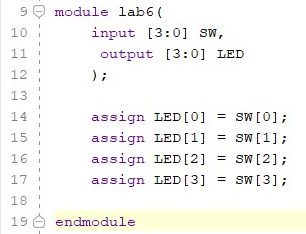
Then, find the output LED’s and uncomment 4 LED’s as well. You’re looking for something like:



Save the file.

1. Copy over the Verilog (programming) file from Lab 4 or create a new one. Starting from the Verilog file in Lab 4 might be slightly easier. After inserting a new (Design) source (lab1.v) you should see something like the following: 

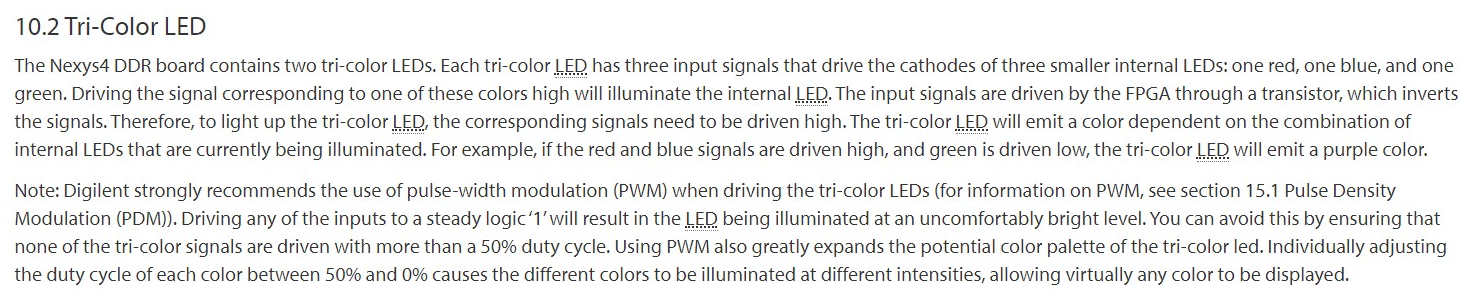
Note though that the syntax for switches in the constraints file “SW[0]” doesn’t match the code in the old lab. I think its better to NOT change the constraints file and change my verilog to match it.

1. Do the swtiches and LED’s work? Let’s try an easy code – each of the switches is directly mapped to an LED. This is done with a wire. Revise your verilog file to look like: 

Note that verilog is CASE SENSITIVE, so LED[0] is not the same as led[0]. Also note that the variables in the code (lines 14-17) are also used as I/O ports in the module. Undeclared variables are bad in verilog – make sure everything is declared. Finally, remember this is hardware layout, not software instruction. “LED[0]=SW[0]” in verilog means “run a breadboarding wire from SW[0] to LED[0]”.

After you’ve modified the verilog design file, save it and then Run Synthesis, Implementation, and then generate and upload a bitstream (program the FPGA device). Make sure that all 4 LED’s are controlled by the 4 switches before going on to the next step.

1. **RGB LED** There are lots of output possibilities with our boards. Another output is a set of RGB LED indicators. They are documented online (and in the onedrive) at <https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/reference-manual> in section 10.2 of the manual.

 Each of these LED’s are “active-high”

Modify your design from the previous step so that SW 0, 1, and 2 also turn on the three colors in LD17

The relevant parts of the constraint file are here:



Modify your Verilog design file so that the switches also control the RGB LED. You’ll need to re-run your synthesis and implementation files and create a new bitstream before uploading the design to the FPGA board.

1. Yikes! Those RGB LED’s are bright! Modify your design so that SW[1:3] map to RGB colors (as before) and have SW[0] serve as an “enable” bit. Ie, the RGB LED’s will only listen to their respective switch if enable (SW[0]) is also high.

Useful syntax: in Verilog,

NOT A is written ,

B AND C is coded , and

D OR E is coded .

These are all bitwise operators. Also remember the order of operations – not comes before and, and comes before or, and parentheses also work!

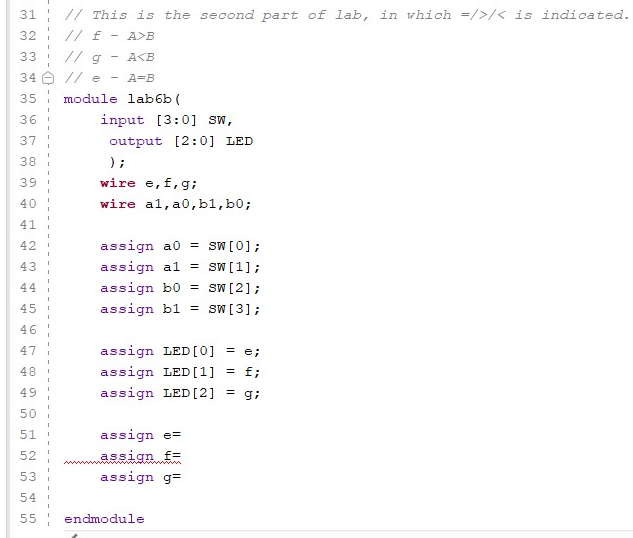
**Optional, not required:** (only if you’re way ahead) Connect an (input) switch to a PMOD i/o pin and measure how fast the FPGA responds to an incoming clock signal at ~ 100kHz with an oscilloscope (ask Moore for help with the wiring – you’ll need to share grounds between the trainer and FPGA for this).

## Part B

1. Now that you see how to use the inputs and outputs, let’s work on the comparator. Assuming that , (where is the MSB) and B is defined the same way, the variable **e** could mean equality, for example, if A=B, e = 1, and if e=0, A ~= B (~= means “not equal”).

Using a truth table or K-map, what is the logical expression for e? ( ie, when is A=B?)

1. Further, let **f** be the result of the test for A>B. Specifically, f=1 when A>B, and f is false otherwise. What is the expression for f?
2. Finally, **g** should indicate A<B. Use a truth table and/or K-map generate an expression for g. (note, this could be easy or hard to do)
3. Implement the logic for the comparator in a new verilog file (call it 5b), or comment out all of the old logic in your design file and create a new module at the bottom of the file. Send output () to 3 LED’s, or RGB LED’s. Note, just like on a breadboard, you can have intermediate wires in a design. In this problem, it might make it easier to have wires that correspond to e,f,g and a1,a0, and b1,b0. Here’s a possible starting place for this design:



You’ll obviously have to complete lines 51-53.

Implement the design, upload it to the board, and clap your hands with joy when it passes a manual test!

1. **Extra fun (not required):** map one input (eg ) from a function generator and send one output to an output (both via a PMOD extension). Then watch the input and output signals on an oscilloscope. Measure how fast the FPGA responds to the input signal from the function generator. Eg, can the FPGA keep up with a 100kHz input signal? What about 1MHz?

## Writeup

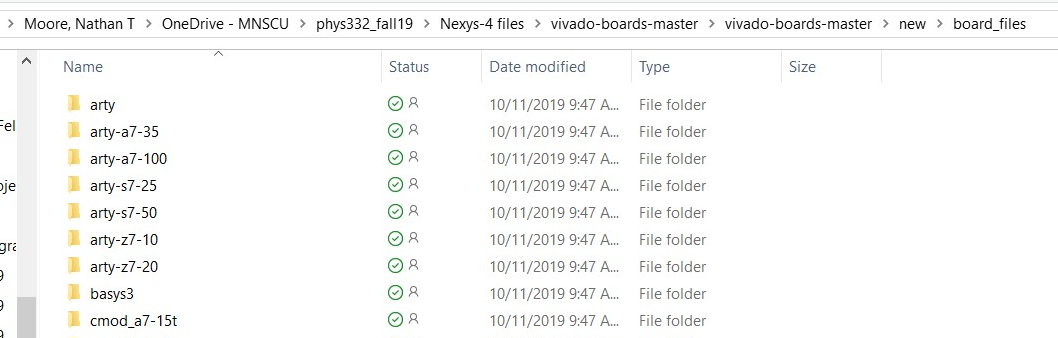
Please submit the Verilog file(s) you used to get both parts of the lab working. If you’d like to make a github repo that you share with me that’s fine.

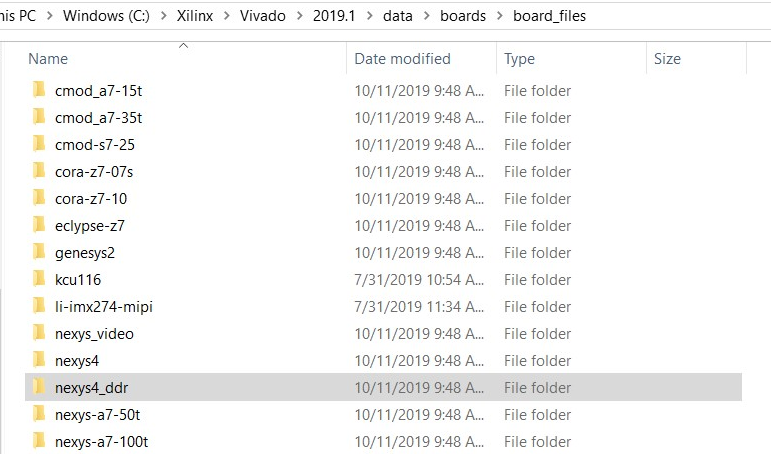
Also, please submit a brief summary of the new things you learned (perhaps including debugging Verilog code?) while finishing the lab.

Finally, please upload a cellphone video of the working comparator design to MS Streams, Youtube, etc and share the video with me.

## Appendix: Installing Digilent Board Files

Following step 3 in this tutorial, <https://reference.digilentinc.com/vivado/installing-vivado/start>,

1. Download the board files from <https://github.com/Digilent/vivado-boards/archive/master.zip>
2. Unzip the archive and copy all the folders inside vivado-boards-master\new\board\_files
3. Paste these folders in the directory c:\Xilinx\Vivado\*version*\data\boards\board\_files



1. Then, when you re-open Vivado, you should see the Nexys4 ddr as a board option. 