Lab 2:

What I think I have to do:

Basically I have to make it so that I can flip 4 bits, 0-3 and then check for parity.

Also I have to make a truth table to verify it.

Included is source code files along w/ the .txt file for the test.

1. tried to get nxclient working, but the graphics were garbled… could not make sense of it. Even the mouse cursor was goofed up.
2. Used ssh instead with
   1. ‘ssh-add nx\_key.key’ after swetting file permissions of 0600 to the file
   2. ssh –X tuc56100@electro9 –X
3. Run ISE project navigator with ‘remote\_xilnix.sh’
4. Opened lab2 odd bit parity project file on the unix machine
5. Never mind this fucking thing is way too god damn slow. I’m not sure if it is just this shitty computer or what, but I am going to try and get no machine working again by clearing the cache.
6. Ok fuck this, installed it on my windows xp virtual machine, now I am connected just fine.
7. I
8. Modify the test file for odd testing
9. Have all 16 types in there
10. I then check what causes the parity to be odd and put a \_1 behind that
11. Now that I know this I can then create a statement in the code to actually create this module properly.
12. Oops in the orig code I accidentally used ‘and’ instead of &. Not sure if this matters, but I changed it over to the new one.
13. OHHHHH! Also, you can not just have the (derple derp… | derple derp) you have to use (derple derp) | (derple derp)
14. Actually had to do assign parity =…
15. And I forgot to do the last line of the parity =
16. Compile to .bit file with
    1. Implementation
       1. Xc3s500e-4g320
          1. Lab2\_top\_io\_wrapper
             1. Implement design
             2. Generation programming file
17. Copy .bin file to local windows xp computer
18. Open digilent adept
    1. Click config
    2. Click prom/browse
    3. Find .bit file
    4. Click program
19. Press reset button on board

// General notes about how I think the switches map to my d…

in the file that defines the lab2\_top\_io\_wrapper module we see a line that creates an instance of the module for our 4 bit odd parity checker module, in that we see the following

op U1 (.d(sw[3:0], .parity(Led[0]););

I think that line creates a wire defining the input and output, it then maps d[3:0] to sw[3:0] which is defined in 2612\_ver3.ucf which defines sw as being our switches ld0,ld1, etc… I think that the switch on the very right is 0.