

CS/EE 5830/6830 VLSI Architecture

CAD Assignment #5

Due Tuesday March 29th, 11:59pm

This assignment has three components:

1. **Compute the power consumption of each of your 32-bit adders from CAD3.** Use the procedure described in Section 7.6 of the CAD book (linked to the class web site). Basically this involves simulating with the mixed-mode analog-digital simulator (with your adder being simulated with Spectre) but plotting current drawn from the power supply instead of voltages. You can then use the calculator in the Analog Environment to integrate under that current curve to compute total power consumed during the simulation.

Use the AddTest.v testbench file for your measurement linked to the class web site (in the labs directory). This file should work both as a Verilog testbench for switch-level testing, and for mixed-mode simulation with your adder being simulated as an analog component using Spectre. Note that the testbench has only 20 time units of delay between steps so if you want to use it as testbench for plain Verilog simulation, make sure you're using switch-level simulation (i.e. NOT unit-gate-delay behavioral simulation) so you don't get errors from not waiting long enough on each step. This testbench assumes that you've named your inputs X<31:0>, Y<31:0> and Cin, and your outputs S<31:0> and Cout. If you've used different names you will need to update either the testbench or your schematic.

Run your (mixed-mode) analog transient analysis using AddTest.v as the testbench for 400ns, and measure power over that same interval. Turn in both the Verilog output (the verilog.log file mentioned on page 194 of Chapter 4 of the CAD book), and your measurement of total power for that set of inputs.

2. **Design and build a 16-bit signed multiplier using radix-4 Booth encoding and using gates from the UofU_Digital_v1_2 library or your own transistor-based circuits using nmos and pmos gates from the UofU_Analog_Parts library, or a mixture of those two.** You should produce a 32bit result of the multiplication. I'd like everyone to include speed in some way in your design criteria. You can go for ultimate speed, or speed*power product, or speed*area product but use some techniques to improve performance (that is, don't just go for small size and poor performance). Write a short description of your design strategy. Use any techniques you like from what we've talked about in class. Tell us which techniques you decided to use and why.

If you know how to use Synopsys synthesis (Chapter 9) you may use Verilog, but only in a structural way. That is, you can't just say $A = B * C$; and let Synopsys build your multiplier. You can, if you like, use structural Verilog to assemble your circuit. Structural

Verilog means that you are only using instantiations of gates and your own modules. No behavioral code allowed.

Measure the speed performance of your multiplier. Tell us why the numbers you picked in your testbench are good numbers to use for measuring performance. Report the worst-case delay through your multiplier.

Also, count the number of transistors in your multiplier. The easiest technique is to use the LVS trick described in CAD3.

3. In this part you will synthesize a 16-bit signed multiplier using a Verilog behavioral description and the Synopsys Design Compiler. You will use the UofU_Digital_v1_2.db library as the target library that Synopsys uses to find gates for the synthesized circuit. Synopsys synthesis is described in detail in Chapter 9 of the CAD manual. I'll describe the process in class too. You should be able to use the mult-script.tcl script to do everything.

Play with the synthesis parameters (in the mult-script.tcl file) to synthesize the fastest multiplier that Synopsys can build, and also one that has the same delay as yours (assuming that yours isn't faster than Synopsys' fastest version!), and compare the transistor counts of your version and the synthesized versions.

Basically the synthesis procedure is:

- a. Make a new directory under IC_CAD in which to run synopsys. I call mine `$HOME/IC_CAD/syn-s11`
- b. Copy the `.synopsys_dc.setup` file (note the dot!) from `/uusoc/facility/cad_common/local/class/6830/S11/synopsys` to your `IC_CAD/syn-s11` directory
- c. Copy the `mult-script.tcl` file from that same directory to yours
- d. If you name your Verilog file (the one that describes the multiplier) `mult16.v`, and your multiplier macro "mult16" then you should be able to just run the synthesis with `syn-dc -f mult-script.tcl` which should result in `mult16_struct.rep` and `mult16_struct.pow` files that will report speed and power. If you use different names, you'll have to modify the `mult-script.tcl` file
- e. You can change the `myPeriod_ns` variable in the `mult-script.tcl` file to force Synopsys to target different speeds.

You will get some warnings when you run the `mult-script.tcl` script. You should get warnings about creating a virtual clock, about changing the design rule attributes from the driving cell to a port, and about nothing matching 'clk.' These are all OK. The clk warnings are because this should be a combinational circuit so we're "faking" a clock to give Synopsys a clock period speed target. The design rule attributes on the port are because we're specifying the driving cells for the inputs ports.

The process will stall for a while as the UofU_Digital_v1_2.db library is analyzed. This is normal and will only happen the first time you run Synopsys in your syn-s11 directory. This delay is caused by the tool parsing the database that describes the cells in the UofU_Digital_v1_2 library and stashing that info in your directory.

The speed result of your synthesis will be in the mult16_struct.rep file and will be in terms of the “slack” of your circuit. The “required time” is the time you asked Synopsys to hit. The “arrival time” is the time your circuit actually achieved. If the slack is negative, that means you didn’t hit timing (your circuit isn’t fast enough). If the slack is positive, then you went faster than you needed to according to the spec. Positive slack is good! But, regardless of that, the “data arrival time” is what Synopsys thinks the worst-case time is through your circuit.

The easiest way to specify a multiplier in Verilog is simply to use $A = B * C$; and let Synopsys (more specifically the DesignWare components) figure out how to build it. One issue in Verilog is how to force a signed version of a multiplier rather than an unsigned version. It turns out that Verilog2001 has a special keyword you can use to specify that a reg, input, or output is supposed to be signed. One way to write a signed multiplier macro, for example, is:

```
module mult8 (in1, in2, product);
input signed [7:0] in1;
input signed [7:0] in2;
output signed [15:0] product;

assign product = in1* in2;

endmodule
```

Without the “signed” keywords these are the results:

```
in1=00000000, in2=00000000, product=0000000000000000
in1=00000010, in2=11111000, product=0000000111110000
in1=11111111, in2=11111000, product=1111011100001000
```

With the “signed” keywords, these are the results:

```
in1=00000000, in2=00000000, product=0000000000000000
in1=00000010, in2=11111000, product=1111111111110000
in1=11111111, in2=11111000, product=0000000000001000
```