ECE 4/574 Homework 7 – Functional Coverage

All reports you submit must be plain text – no Word, no PDFs.

Background:

You will re-use your solution for Homework 4 GCD to collect functional coverage and test the design.

Work to do:

1) Make sure you have a GCD design and associated testbench that work.

Here "work" means that they compile and simulate without syntax or runtime errors. Have only this one line in the input_data file:

1 1

I.e. you take the GCD of 1 and 1.

2) Write covergroups.

In a separate .sv file called covergroups.sv, write three covergroups:
- the first covergroup has three coverpoints: the first monitors the present state of the controller FSM, the second monitors the next state of the controller FSM, and the last is a cross between the first two.
- the second covergroup has one coverpoint which monitors the output port of the GCD module. The coverpoint has 4 bins: the first bin covers the range 0:500, the second bin covers the range1000:2000, the third the range 7000:8000, and the last covers only the number 5000.
- the third covergroup has one coverpoint which monitors the present state of the GCD controller FSM, and declares one illegal transition bin. The illegal transition is 2'h3 => 2'h1

All covergroups sample their signals at the positive edge of the clock signal clk.

3) Collect covergroup coverage

In the testbench, create one instance of each of the covergroups. (Remember you need to `include the file that declares the covergroups).
Compile and simulate.
Generate a textual detailed coverage report named func.rpt.

4) Increase coverage

Now modify your input_data file so as to have 100% coverage on the first two covergroups.

5) Constrained random generation

In your testbench, generate 10 random pairs of 32'bit integers and feed them one by one to the gcd module. The numbers, call then a and b, must satisfy the following constraints:
a <= 500

b >= 0
a+b = 500
For every pair of numbers you generate, feed them to the GCD module and then $display the output result.
Compile, simulate while collecting functional coverage.
Generate a detailed textual coverage report, call it rand.rpt
Read below for how to structure your testbench so we can run different parts of it easily.

6) How to structure your testbench

Your testbench will have at least one initial block which will do the following, in sequence.

First, it reads data from input_data and feeds them to gcd()

Then it executes part 4 – i.e., it generates 100 random input pairs and feeds them to gcd()

You won't get grades for anything we can't test like this.

What to turn in

- All system Verilog files used
- All report files requested above: rand.rpt, func.rpt
- The input_data file from part 4

Grading

b.	Part 2: write covergroups	30%
c.	Part 3: collect	20%
d.	Part 4: increase coverage	10%
e.	Part 5: constrained random	20%
d.	Testbench structured as described	10%
e.	Code cleanliness, coding style	10%