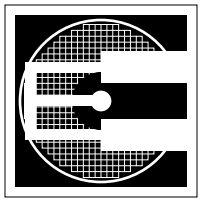

Project 1: Verilog Modeling (10%)

ENEE 359a: Digital VLSI Design, Spring 2007
Assigned: Thursday, February 1; Due: Tuesday, February 13

1. Purpose

The purpose of this assignment is to learn the rudiments of the Verilog hardware description language in the context of sequential circuits. Some of the most important concepts you will learn are those of *non-blocking assignments* and *concurrency.* Non-blocking assignments are specific to the Verilog language; concurrency is a powerful concept that shows up at all levels of digital circuit and digital system design.

2. Combinational Logic

First, you will design a simple switch. The switch has four inputs and two outputs.

Inputs:

- 24-bit data
- 2-bit routing data for output byte 0
- 2-bit routing data for output byte 1
- 2-bit routing data for output byte 2

Outputs:

- 24-bit data
- 1-bit valid

The switch will work as follows: whenever any of the routing inputs change, the 24-bit (3-byte) output data will contain bytes from the input, ordered as specified by the routing information. Routing information indicates which byte from the input bus should be routed to the corresponding output byte; e.g., if routing data for output byte 1 has the value ‘2’ in it, input byte 2 should be routed to output byte 1. Note that a value of ‘4’ is invalid; any other combination of values is acceptable and correct. If the routing information is invalid, then the output data can be anything, but the output “valid” bit should be zero. For valid routing information, the output “valid” bit should be “1”.

3. Sequential Logic

You will build a simple multiplier that performs a 16- x 16-bit multiplication using repetitive shifts and adds, assuming unsigned data input. The multiplier will produce a 32-bit output using the *early-out* technique of stopping as soon as there are no more bits left in the multiplicand. (note that this is *not* the traditional definition or usage of multiplicand/multiplier, which specifies a far simpler technique of adding the multiplicand to itself <multiplier> times …) The inputs and outputs and their behavior are as follows:

Inputs:

- 16-bit mcand
- 16-bit mplier
- 1-bit reset_n (asynchronous)
- 1-bit go (also asynchronous but paired with reset_n)
- 1-bit clock

Outputs:

- 32-bit product
- 1-bit done

A *reset_n* signal and matching *go* signal are necessary to initialize your state machine in a reset state and initialize the two registers in your design that hold *mcand* and *mplier*. Your reset using the signal *reset_n* should be asynchronous. The *go* signal is paired with *reset_n* in that will be low before and after the transition of *reset_n*: first, *go* will go low, then *reset_n* will transition from low to high, and then *go* will transition from low to high (which goes low first does not matter). Your design should accept the input values of *mcand* and *mplier* on their ports and reset any internal state when(ever) *reset_n* transitions from low to high. Once initialized, and once *go* is high, the design does not require any more inputs other than the clock *clk*. The output consists of the product on the port *output_number* and a *done* signal of 1 indicating that the output is valid. When the *done* signal is 0, then *output_number* can be anything. For the sake of simplicity, you can assume that the inputs will not be both zero (otherwise you would have to distinguish between an initial condition and a final condition that are identical).

4. Running & Submitting Your Project

First, tap verilog to get access to the simulators; i.e., at the Unix prompt type the following:

```
tap cds-v
```

Then you can invoke your code this way:

```
verilog testbench.v yourcode.v
```

or

```
ncverilog testbench.v yourcode.v
```

The ncverilog simulator has a longer start-up time but executes much faster once it gets going.

You must use the provided testbenches to test your design (unmodified). The testbenches test your design with different inputs.

Additional details:

- A couple of software-oriented approaches to stay away from since they are not synthesizable (you will later synthesize your design): There is no hardware analog to the Verilog *initial* block construct. The loop limit for a for loop or a while loop cannot be variable. In general, functions are synthesizable, but recursion is not synthesizable.
- The only type of register permitted in the design is rising-edge-triggered.
- In a clocked *always* block you should use the nonblocking assignment operator, <=, when assigning to a signal that is to be synthesized as a register.

When you submit your code to me via email, all I want is your verilog code … i.e., do not send me your testbench.v file (I will use my own). Do not change any of the variable names within the provided switch.v and multiplier.v files, for hopefully obvious reasons, and do not change the file names—I want two verilog files submitted via email, one named “switch.v” and the other named “multiplier.v”.