1. Purpose

The purpose of this assignment is to learn the rudiments of the Verilog hardware description language in the context of sequential circuits: you are going to build a simple state machine. 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 byte 0
- 2-bit routing data for byte 1
- 2-bit routing data for 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 the bytes from the input, ordered as specified by the routing information. If the routing information is invalid, meaning that two routing values are equal (specifying that two input bytes should be routed to the same output destination), then the output data can be anything, but the output “valid” bit should be zero. For valid routing information, the “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 is a far simpler technique of adding the multiplicand to itself <multiplier> times …) The inputs and outputs and their behavior are the same as in the original Project 1:

Inputs:
- 16-bit mcand
- 16-bit mplier
- 1-bit reset_n (asynchronous)
• 1-bit go
• 1-bit clock

Outputs:
• 32-bit product
• 1-bit done

Just as in the original Project 1, the reset_n signal is asynchronous and active low, the go signal tells the multiplier when to operate and when the input data becomes valid, and the output done bit should be high for only one clock cycle. When done is not high, the output product can be anything.

4. Running & Submitting Your Project

First, tap verilog to get access to the simulators; i.e., log on to jasminesun.eng.umd.edu, and at the Unix prompt type the following:

/software/cadence/cadence LDV

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.

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”.