LSU EE 4755 Homework 7 Due: 16 November 2018

For instructions visit `https://www.ece.lsu.edu/koppel/v/proc.html`. *For the complete Verilog for this assignment without visiting the lab follow* `https://www.ece.lsu.edu/koppel/v/2018/hw07.v.html`.

Problem 0: Following instructions at `https://www.ece.lsu.edu/koppel/v/proc.html`, set up your class account, copy the assignment, and run the Verilog simulator and synthesis program on the unmodified homework file, `hw07.v`.

Homework Overview

Modules `mult_seq_ds_prob_1` and `mult_seq_d_prob_2` have similar sets of ports as the fast pipelined multiplier from 2017 Homework 7, but the code within this semester's modules implements a sequential rather than a pipelined multiplier. In this assignment these modules will be modified so they use handshaking to start and announce the availability of a product, and in `mult_seq_d_prob_2`, so that the latency (number of clock cycles) needed to compute a product depends upon the number and placement of zeros in the multiplicand. Unlike 2017 Homework 7, the solution to this problem **will not be pipelined**.

Initially the testbench should report errors for both modules, identified as `Prob 1` and `Prob 2`, these errors are due to the modules ignoring the handshake signals (`in_valid` and `out_avail`). The testbench, however, should correctly synthesize both modules. The testbench will also synthesize an original copy of each module, for comparison.

A correct solution to Problem 1 will eliminate the testbench errors. A correct solution to Problem 2 will reduce the number of cycles needed to compute a correct result. A future assignment or possibly final exam questions will ask about the difference in performance between the Problem 1 and Problem 2 modules.

The testbench reports details of the first few errors encountered on each modules, and then a summary. If you would like to test your module on a specific multiplier/multiplicand pair search for `values to try out` and add those to the beginning of the list assigned to `tests`. The modules are instantiated with names `prob1_m1`, `prob1_m2`, etc. Look for these when using debugging tools such as SimVision.

The synthesis script for this assignment can be run with the command `genus -files syn.tcl`. It synthesizes modules `mult_seq_ds_prob_1` and `mult_seq_d_prob_2`, as well as unmodified copies of these modules, `mult_seq_ds_prob_1_orig` and `mult_seq_d_prob_2_orig`. Each is synthesized at two sizes and three different values of m. The synthesis script assumes at latency of $\lceil w/m \rceil$ for these modules, which is an overestimate for Problem 2.

Problem 1: Module `mult_seq_ds_prob_1` has two parameters, `w` and `m`, four input ports, `clk`, `in_valid`, `plier`, and `cand`, and two output ports, `prod`, and `out_avail`.

The unmodified module will set `prod` to the $2w$-bit product of w-bit inputs `plier` and `cand`, which hold unsigned integers. It computes the product using m-bit partial products, similar to the method used by `mult_seq_dm` but using the streamlined code in `mult_seq_stream`. In `mult_seq_ds_prob_1` the product will be available with a latency of between $\lceil w/m \rceil + 1$ and $2\lceil w/m \rceil - 1$ cycles. The latency will be $\lceil w/m \rceil + 1$ when the multiplier and multiplicand arrive when `iter` is reset to zero, but if they arrive one cycle later the latency will be $2\lceil w/m \rceil - 1$. If that higher latency bothers you then this is your problem. (Even those that don't care need to solve this problem.)

The reason for this variation in latency in `mult_seq_dm` and friends is that those modules have no way of knowing when a new multiplier/multiplicand pair has arrived (other than continually comparing them to prior values which would require extra hardware). As the alert student suspects, input in_valid in mult_seq_ds_prob_1 is used to indicate the arrival of a new pair. Modify mult_seq_ds_prob_1 so that it starts a new multiplication at the positive edge of clk when in_valid is 1, even if there's a multiplication in progress. When a new multiplication starts set out_avail to 0, and set it back to 1 when prod holds the correct product.

When this problem is correctly solved the testbench should not show errors on this module. The testbench instantiates the module for three sizes of m, and it has Prob 1 in the name. See the checkboxes in hw07.v for additional requirements and tips. Don't forget synthesizability as well as clear and efficient code.

Problem 2: The unmodified module mult_seq_d_prob_2 computes a product in at best $\lceil w/m \rceil + 1$ cycles. For some multiplicands the value of cand_2d[iter] (see the code) will be zero for certain values of iter. An extreme case is when the multiplicand is zero, but there are many other situations where cand_2d[iter] will be zero. Currently iter is incremented by 1 each clock cycle. Modify mult_seq_d_fast so that iter is incremented so that it points to the next non-zero value in cand_2d, or to $\lceil w/m \rceil$ if there are no more non-zero values. Doing so will reduce the number of clock cycles needed to compute a product. This should be reflected in the Avg Cyc shown for the Prob 2 module by the testbench.

Use the synthesis script syn.tcl to find the clock period of the module. The latency shown by the synthesis script assumes w cycles per multiply. To find the actual latency of your module multiply the clock period reported by the synthesis script with the average cycles reported by the testbench.

The goal is to reduce the average number of cycles, as reported by the testbench while also keeping clock period low (as reported by the synthesis script) so that the average latency, measured in seconds (or some fraction) is lower.