LSU EE 4755 Homework 7 Due: 29 November 2017

For instructions visit `http://www.ece.lsu.edu/koppel/v/proc.html`. *For the complete Verilog for this assignment without visiting the lab visit* `http://www.ece.lsu.edu/koppel/v/2017/hw07.v.html`.

Problem 1: Module `mult_pipe` is a simple pipelined multiplier which multiplies two `w`-bit operands, computing `m` partial products per stage in $\lceil w/m \rceil$ stages. The latency of this multiplier is $\lceil w/m \rceil$ cycles regardless of what is being multiplied, which in many circumstances is just fine.

In contrast `mult_fast` is designed for situations in which lower latency is beneficial. The goal is to compute the results for "easier" products in fewer cycles. For example, multiplying $abcd_{16} \times 9876_{16}$ in a 16-bit degree-4 (`m=4`) multiplier would take four cycles since all partial products are needed. But, $abcd_{16} \times 1_{16}$ requires one partial product and so the product should be available sooner.

Like the other multipliers `mult_fast` has `w`-bit inputs `plier` and `cand` and a `2w`-bit output `prod`, and a 1-bit `clk` input. But it also has a 1-bit input `in_valid` and a 1-bit output `out_avail`.

At each positive clock edge if input `in_valid` is `1` `mult_fast` should start computing the product of the input values, `plier`×`cand`. If input `in_valid` is `0` then the external hardware does not need `plier`×`cand`. Though the module can start computing `plier`×`cand` when `in_valid` is `0`, it should not set `out_avail` when the product is ready.

The outputs `out_avail` and `prod` should be set at each positive clock edge. If `out_avail` is `1` then `prod` is the product of values appearing earlier at the inputs at a time when `in_valid` was `1`. The products should appear in the same order as the inputs. For example, suppose in cycle 10 the values $8765_{16} \times 53ab_{16}$ appear at the inputs and at cycle 11 the values 1×1 appear. Even though 1×1 can be computed in one cycle it cannot appear at the outputs until after $8765_{16} \times 53ab_{16}$ appears. If it takes four cycles to compute $8765_{16} \times 53ab_{16}$ then it will appear at the outputs in cycle 14, and so the product 1×1 will not appear at the outputs until four cycles after it arrives, at cycle $11 + 4 = 15$.

A simple case is when `in_valid` is always equal to one. In that case after `w/m` cycles `out_avail` should always be set to one and the value at output `prod` is the product of inputs appearing `w/m` cycles earlier, which is how an ordinary pipelined multiplier, such as `mult_pipe` operates.

Next, consider the table below which shows inputs and possible outputs. In cycle 0 the values 1×11 arrive. Their product, 11 appears at the outputs in cycle 1. In cycle 1 values `98` and `99` appear at the inputs but since `in_valid` is `0` their product is not needed. At cycle 2 values `3` and `22` are at the inputs, the product $3 \times 22 = 66$ appears at the output in cycle 4. Note that at cycles 2 and 3 `out_avail` is `0`. The product 4×14 appears at the outputs in cycle 5.

cycle	0	1	2	3	4	5
in_valid	1	0	1	1		
plier	1	98	3	4		
cand	11	99	22	14		
out_avail	0	1	0	0	1	1
prod		11			66	56

Note that it took two cycles to compute 3×22 but one cycle to compute the other products.

(*a*) Modify `mult_fast` so that it sets `out_avail` when a product is ready. If this is completed correctly the testbench should show that there are zero errors.

(*b*) Modify `mult_fast` so that the product is ready when all of the remaining multiplicand bits are zero. That is, suppose stage i examines bits mi to $mi + m - 1$ of the multiplicand. If multiplicand bits $w - 1$ to $mi + m - 1$ are all zero then the product is finished at stage i. If this is completed

correctly then the testbench should show that the average number of cycles for the degree-2 fast multiplier is about 5.1 and for degree 4 it should be about 2.7.

- Modules must be synthesizable.
- Modules must be reasonably efficient.
- Do not assume specific parameter values.
- Use SimVision for debugging.