LSU EE 4755 Homework 6 Due: 29 November 2016

Problem 0: Review the instructions for account setup and homework workflow on the course procedures page, `http://www.ece.lsu.edu/koppel/v/proc.html`.

Look through the code in `hw06.v`. These modules compute the floating-point sum of squares of their input, similar to the midterm exam problem but without the square root.

Module `mag_functional` is a non-synthesizable version of the module. It is not synthesizable by Cadence Encounter because it operates on floating-point values. The module is included to help in understanding the other modules.

Module `mag_comb` is a synthesizable combinational version of the module. The floating-point operations are implemented using modules from the ChipWare library. See the ChipWare documentation, linked to the course references page, for details.

Module `mag_seq`, when finished, computes mag sequentially. It contains some code, including floating-point module instantiations, but is not complete. It has an input `start` to initiate the computation and an output `ready` to signal that the computation is complete.

Module `mag_pipe`, when finished, computes mag in pipelined fashion. At each positive edge it reads a vector from its input and provides the mag of a prior vector at its output.

Module `mag_comb` should be fastest, but of high cost. Module `mag_seq` should be the lowest cost module and `mag_pipe` should be the highest cost but also the highest throughput.

The testbench provides test inputs to the three synthesizable modules. Initially, `mag_comb` should pass all tests and the others should fail all tests. To facilitate debugging the first eight tests are the vectors $[0,0,0], [0,0,1], [0,1,0], [0,1,1], \ldots$. After that the vector components are randomly chosen over the range $[-10, 10]$.

Remember that the values are IEEE 754 single-precision floating point. A 0 in this FP representation is `32'h0` and a 1.0 is `32'h3f800000`, a 2.0 is `32'h40000000`, and a 3.0 is `32'h40400000`.

To solve this assignment it is very important to use the waveform viewer for debugging. To do so start the simulator graphically using the command `irun -gui hw06.v`. From the Design Browser pane locate `testbench` and under it look for `m2` (for `mag_seq`) or `m3` (for `mag_pipe`). Select objects in the Objects pane and send them to the waveform window by pressing the waveform toolbar button (it looks like a logic analyzer display) or by selecting the sequence Windows → Send To → Waveform. Run the simulator by pressing the play toolbar icon. If you've made changes to the Verilog or otherwise want to re-run the simulation without exiting select Simulation → Reinvoke Simulator.

For further documentation see the SimVision documentation on the course references page, `http://www.ece.lsu.edu/koppel/v/ref.html`.

There is a synthesis script that will synthesize each module at high (slow) and low (fast) clock period targets. To run it use the command `rc -files syn.tcl`. This will take a long time to run, so only run it to satisfy your curiosity. Check for synthesizability by manually running the synthesis using the instructions on the course procedures page, `http://www.ece.lsu.edu/koppel/v/proc.html`.

Note: There are no points for this problem.

Problem 1: Module `mag_seq`, when completed, will compute the magnitude sequentially. It should start when input `start` is logic 1 on a positive clock edge and it should signal completion by setting output `ready` to one several clock cycles later. The module should use one floating-point multiply and one FP add unit. The module already instantiates these, and contains some logic for performing the different steps, including setting `ready` (though not at the right time). Complete the module so that it works correctly. See the checklist in the Verilog source for hints and reminders.

Problem 2: Module `mag_pipe`, when completed, will compute the magnitude in pipelined fashion. That is, it will read a vector from its inputs at every clock cycle, and will present a magnitude at its output every cycle. The magnitude should be for the vector that was at input `v` `nstages` cycles in the past, where `nstages` is a constant in the module indicating the number of stages. The inputs to the module are available near the end of the clock cycle and the outputs are expected at the beginning of the clock cycle.

Choose the number of stages needed to maximize throughput. That is, minimize the delay in each stage. Of course, within that constraint minimize cost.

Pay close attention to where data is. Remember that at any one time the module will hold data for `nstages` different vectors. Use a pipeline diagram to make sure that data from the different vectors don't get mixed up. A common problem is a newly arriving vector overwriting data for an earlier vector. That's avoided by moving data long from stage to stage.

Be sure to use the waveform viewer for debugging. Remember that the first eight test vectors consists of 0 and 1 components, making debugging easy.