LSU EE 4755 Homework 6 Due: 22 November 2021

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/2021/hw06.v.html`.

Problem 0: If necessary, follow the instructions at `https://www.ece.lsu.edu/koppel/v/proc.html` to set up your class account, copy the assignment, and run the Verilog simulator and synthesis program on the unmodified homework file, `hw06.v`. Do this early enough so that minor problems (*e.g.*, password doesn't work) are minor problems.

Teamwork

Students can work on this assignment in teams. Each student should submit his or her own assignment but list team members. It is recommended that one team member be responsible for learning SimVision.

Every member of a team that has completed a project, must be capable of re-solving the problem. It is recommended that all team members re-solve the problem on their own for their own pedagogical benefit.

Problem 1: Complete module `multi_step_pipe` so that it is a pipelined version of the `multi_step_functional` or `multi_step_seq` modules. All of modules are in `hw06.v`. (This is based on 2020 Solve-Home Final Exam Problem 2.)

The module must accept a new set of `v0` and `v1` values each clock cycle and produce a new result each clock cycle. In the module set `nstages` to the number of stages in your module, so that the value of output `result` is based on the inputs that appeared `nstages` clock cycles ago.

Instantiate as many Chipware floating-point multiplication and addition modules as needed. (Do not use procedural code for the arithmetic.) The critical path should pass through at most one floating-point module.

Also, set the `ready` output at the correct time. Output `ready` should be set to the value that `start` has `nstages` ago.

The testbench will show a trace for about the first three computations (inputs in which `start` was 1), and will show a trace for the ten cycles preceding each error, up to seven errors. A tally of errors will be shown at the end. Here is a sample of the testbench for a working module:

```
MS Pipe  Cyc  20  In:   0.0,  0.0 ->   0.0  Rdy 0 , Res:   0.0
MS Pipe  Cyc  21  start=0                  Rdy 0 , Res:   0.0
MS Pipe  Cyc  22  start=0                  Rdy 0 , Res:   0.0
MS Pipe  Cyc  23  start=0                  Rdy 1 , Res:   0.0 Good
MS Pipe  Cyc  24  start=0                  Rdy 0 , Res:   0.0
MS Pipe  Cyc  25  In:   1.0,  0.0 ->   1.0  Rdy 0 , Res:   0.0
MS Pipe  Cyc  26  start=0                  Rdy 0 , Res:   0.0
MS Pipe  Cyc  27  start=0                  Rdy 0 , Res:   0.0
MS Pipe  Cyc  28  In:   0.0,  1.0 ->   1.0  Rdy 1 , Res:   1.0 Good
MS Pipe  Cyc  29  start=0                  Rdy 0 , Res:   0.0
MS Pipe  Cyc  30  start=0                  Rdy 0 , Res:   0.0
MS Pipe  Cyc  31  start=0                  Rdy 1 , Res:   1.0 Good
For MS Pipe ran 400 tests: Errors: 0 wrong val, 0 bad timing
```

On a cycle in which input `start` is `1` the trace line will show the word `In:` followed by the values of `v0` and `v1`, and to the right of `->` the correct result (which should appear `nstages` cycles later). The text to the right of `Rdy` shows the value of the `ready` output. If the value is incorrect it is followed by an `x`, for example, `Rdy 1x,`.

The text to the right of `Res:` shows the value on the module `result` output. That is followed by text commenting on the result. A comment will be shown if `Rdy` is `1` or if an output is expected. `Good` indicates a correct value at the correct time. `XX: Need Rdy` indicates that the correct value appears at the correct time, but the `ready` output isn't `1`. `XX: Wrong` indicates the wrong value at the time when an output was expected. `XX: Early` indicates the correct value arriving too early. `XX: Unexpected` indicates the wrong value at a time when no value at all is expected.

Below are excerpts from the testbench output on the unmodified module.

```
MS Pipe  Cyc  20  In:   0.0,  0.0 ->   0.0  Rdy 0X, Res:   0.0 XX: Need Rdy
MS Pipe  Cyc  21  start=0                   Rdy 1X, Res:   0.0 XX: Early
MS Pipe  Cyc  22  start=0                   Rdy 0 , Res:   0.0
MS Pipe  Cyc  23  start=0                   Rdy 0 , Res:   0.0
MS Pipe  Cyc  24  start=0                   Rdy 0 , Res:   0.0
MS Pipe  Cyc  25  In:   1.0,  0.0 ->   1.0  Rdy 0X, Res:   0.0 XX: Wrong
MS Pipe  Cyc  26  start=0                   Rdy 1X, Res:   1.0 XX: Unexpected
MS Pipe  Cyc  27  start=0                   Rdy 0 , Res:   0.0
MS Pipe  Cyc  28  In:   0.0,  1.0 ->   1.0  Rdy 0X, Res:   0.0 XX: Wrong
MS Pipe  Cyc  29  start=0                   Rdy 1X, Res:   0.0 XX: Early
MS Pipe  Cyc  30  start=0                   Rdy 0 , Res:   0.0
MS Pipe  Cyc  31  start=0                   Rdy 0 , Res:   0.0
MS Pipe  Cyc  32  start=0                   Rdy 0 , Res:   0.0
MS Pipe  Cyc  33  In:   1.0,  1.0 ->   3.0  Rdy 0X, Res:   0.0 XX: Wrong
MS Pipe  Cyc  34  start=0                   Rdy 1X, Res:   1.0 XX: Unexpected
MS Pipe  Cyc  35  In:  -8.6,  5.0 ->  55.9  Rdy 0X, Res:   0.0 XX: Wrong
MS Pipe  test 4: Inputs at cyc 35, result expected at cyc 35. Wrong val: h'00000022
0.0000 != 55.8659 (correct)
MS Pipe  Cyc  36  In:   0.4,  3.9 ->  16.7  Rdy 1 , Res:  -8.6 XX: Wrong
MS Pipe  test 5: Inputs at cyc 36, result expected at cyc 36. Wrong val: h'c109657e
-8.5873 != 16.7235 (correct)
```

The following is the testbench output on a module in which `nstages` is set too low by 1, and in which `v00` is used where `v01` should be:

```
MS Pipe  Cyc  20  In:   0.0,  0.0 ->   0.0  Rdy 0 , Res:   0.0
MS Pipe  Cyc  21  start=0                   Rdy 0 , Res:   0.0
MS Pipe  Cyc  22  start=0                   Rdy 0X, Res:   0.0 XX: Need Rdy
MS Pipe  Cyc  23  start=0                   Rdy 1X, Res:   0.0 XX: Early
MS Pipe  Cyc  24  start=0                   Rdy 0 , Res:   0.0
MS Pipe  Cyc  25  In:   1.0,  0.0 ->   1.0  Rdy 0 , Res:   0.0
MS Pipe  Cyc  26  start=0                   Rdy 0 , Res:   0.0
MS Pipe  Cyc  27  start=0                   Rdy 0X, Res:   0.0 XX: Wrong
MS Pipe  Cyc  28  In:   0.0,  1.0 ->   1.0  Rdy 1X, Res:   2.0 XX: Unexpected
MS Pipe  Cyc  29  start=0                   Rdy 0 , Res:   0.0
MS Pipe  Cyc  30  start=0                   Rdy 0X, Res:   0.0 XX: Wrong
MS Pipe  test 2: Inputs at cyc 28, result expected at cyc 30. Wrong val: h'00000000
0.0000 !=  1.0000 (correct)
MS Pipe  Cyc  31  start=0                   Rdy 1X, Res:   1.0 XX: Unexpected
```

```
MS Pipe  Cyc  32  start=0                    Rdy 0 , Res:   0.0
MS Pipe  Cyc  33  In:   1.0,  1.0 ->   3.0  Rdy 0 , Res:   0.0
MS Pipe  Cyc  34  start=0                    Rdy 0 , Res:   0.0
MS Pipe  Cyc  35  In:  -8.6,  5.0 ->  55.9  Rdy 0X, Res:   0.0 XX: Wrong
MS Pipe  Cyc  36  In:   0.4,  3.9 ->  16.7  Rdy 1X, Res:   3.0 XX: Unexpected
MS Pipe  Cyc  37  In:  -9.5, -4.5 -> 152.0  Rdy 0X, Res:   0.0 XX: Wrong
```

Make sure that your modules are synthesizable.

The smart way to solve the problem is to base the design on `ms_functional`. Remember that the control logic in `multi_step_seq`, such as logic related to `step`, is not needed in a pipelined implementation. The solution should be relatively short and uncomplicated. For example, no conditionals are needed.

A good way to start is to compute everything in one stage, and when that's correct break the logic into stages so that the critical path passes through at most one floating-point module.