LSU EE 4755 Homework 5 Due: 18 Nov 2024 (evening)

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/2024/hw05.v.html`.

Student Expectations

To solve this assignment students are expected to avail themselves of references provided in class and on the Web site, such as for Verilog programming and synthesis examples, and to seek out any additional help and resources that might be needed. (Of course this doesn't mean asking someone else to solve it for you.) It is the students' responsibility to resolve frustrations and roadblocks quickly. (If you get stuck *just ask for help!*)

This assignment cannot be solved by blindly pasting together parts of past assignments. Solving the assignment is a multi-step learning process that takes effort, but one that also provides the satisfaction of progress and of developing skills and understanding.

Collaboration Rules

Each student is expected to complete his or her own assignment. It is okay to work with other students and to ask questions in order to get ideas on how to solve the problems or how to overcome some obstacle (be it a question of Verilog syntax, interpreting error messages, how a part of the problem might be solved, etc.) It is also acceptable to seek out digital design resources for help on Verilog, digital design, etc. It is okay to make use of AI LLM tools such as ChatGPT and Copilot to generate sample Verilog code. (Do not assume LLM output is correct. Treat LLM output the same way one might treat legal advice given by a lawyer character in a movie: it may sound impressive, but it can range from sage advice to utter nonsense.)

After availing oneself to these resources **each student is expected to be able to complete the assignment alone**. Test questions will be based on homework questions and **the assumed time needed to complete the question will be for a student who had solved the homework assignment on which it was based**.

Problem 0: Following instructions at `https://www.ece.lsu.edu/koppel/v/proc.html`, set up your class account (if somehow necessary at this point), copy the assignment, and run the Verilog simulator on the unmodified homework file, `hw05.v`. Do this early enough so that minor problems (*e.g.*, password doesn't work) are minor problems.

Homework Overview

The module in this assignment, `dot_seq_2`, computes a dot product of two vectors, `a` and `b`, each of length n, where n is even (and greater than zero). At each cycle two elements of `a` and `b` arrive, so it takes $n/2$ cycles for the complete vectors to arrive. The connections to `dot_seq_2` are:

```
module dot_seq_2 #( int w = 5, wi = 4 )
   ( output logic [w-1:0] dp,
     output logic [wi-1:0] first_id, last_id,
     input uwire [w-1:0] a[2], b[2],
     input uwire [wi-1:0] in_id,
     input uwire reset, first, last,
     input uwire clk );
```

Each element of input vector `a` and `b` is `w` bits. Inputs `a` and `b` carry the first two elements of each vector when input `first=1`. At each subsequent cycle `a` and `b` will carry two more elements of the vector until reaching a cycle where `last=1`, in which case `a` and `b` carry the last two elements of each vector. (See the testbench section for examples.) If `first=1` and `last=1` in the same cycle

then the vectors are just of length 2. There is no upper bound on the vector length. Some time after `last=1` output `dp` is to be set to the dot product of the vectors.

In the unmodified assignment `dp` is set to the correct value in the same cycle that `last=1`. In a correct solution `dp` should be set three cycles later to limit the critical path (see the problem description).

Input `in_id` is a `wi`-bit ID number which is used to identify the beginning and end of the vector. Outputs `first_id` and `last_id` are each `wi`-bit ID numbers. In a correctly solved assignment `first_id` is the ID of the beginning of the vector whose dot product appears on `dp` and `last_id` is the ID appearing at the end of that vector. In the unmodified assignment `last_id` is set to the correct value (though along with `dp` it is set too soon).

When input `reset=1` any in-progress dot-product computation is abandoned that outputs `first_id` and `last_id` are set to zero.

Testbench

To compile your code and run the testbench press F9 in an Emacs buffer in a properly set up account. The testbench will apply inputs to `dot_seq_2` and report on the results. Unlike other assignments, only one module instantiation will be tested.

The testbench will show a trace for at least the first 20 cycles, with additional trace output shown before errors. The trace lines from a correctly solved assignment appear below.

```
 2 _F_ ID a8  A 01,01  B 01,10  exp: fid 00  lid 00  dp 00  MOD: FID 00  LID 00  DP xx
 3 ___ ID aa  A 01,01  B 01,10  exp: fid 00  lid 00  dp 00  MOD: FID 00  LID 00  DP xx
 4 ___ ID ad  A 01,01  B 01,10  exp: fid 00  lid 00  dp 00  MOD: FID 00  LID 00  DP xx
 5 ___ ID af  A 01,01  B 01,10  exp: fid 00  lid 00  dp 00  MOD: FID 00  LID 00  DP xx
 6 __L ID b1  A 01,01  B 01,10  exp: fid a8  lid b1  dp 55  MOD: FID 00  LID 00  DP xx
 7 ___ ID b3  A 01,01  B 01,10  exp: fid a8  lid b1  dp 55  MOD: FID 00  LID 00  DP xx
 8 _F_ ID b5  A 01,01  B 01,10  exp: fid a8  lid b1  dp 55  MOD: FID 00  LID 00  DP xx
 9 ___ ID b8  A 01,01  B 01,10  exp: fid a8  lid b1  dp 55  MOD: FID a8  LID b1  DP 55
10 ___ ID b9  A 01,01  B 01,10  exp: fid a8  lid b1  dp 55  MOD: FID a8  LID b1  DP 55
11 __L ID ba  A 01,01  B 01,10  exp: fid b5  lid ba  dp 44  MOD: FID a8  LID b1  DP 55
12 _FL ID bc  A 01,01  B 01,10  exp: fid bc  lid bc  dp 11  MOD: FID a8  LID b1  DP 55
13 ___ ID bd  A 01,01  B 01,10  exp: fid bc  lid bc  dp 11  MOD: FID a8  LID b1  DP 55
14 ___ ID c0  A 01,01  B 01,10  exp: fid bc  lid bc  dp 11  MOD: FID b5  LID ba  DP 44
15 ___ ID c3  A 01,01  B 01,10  exp: fid bc  lid bc  dp 11  MOD: FID bc  LID bc  DP 11
16 _F_ ID c4  A 01,01  B 01,10  exp: fid bc  lid bc  dp 11  MOD: FID bc  LID bc  DP 11
17 ___ ID c7  A 01,01  B 01,10  exp: fid bc  lid bc  dp 11  MOD: FID bc  LID bc  DP 11
18 ___ ID ca  A 01,01  B 01,10  exp: fid bc  lid bc  dp 11  MOD: FID bc  LID bc  DP 11
19 ___ ID cc  A 01,01  B 01,10  exp: fid bc  lid bc  dp 11  MOD: FID bc  LID bc  DP 11
```

Trace lines start with a cycle number, in the sample above they go from 2 to 19 as of this writing. That is followed by three characters that show the state of the `reset`, `first`, and `last` inputs (in that order). In cycle 2 `first=1` and the other two are zero. In cycle 12 `first=1` and `last=1`. In the sample above `reset` is always `0`.

Upper-case labels, such as `ID` and `FID` are used for module inputs and outputs. Lower-case labels, such as `lid` and `dp` are used to show what the testbench expects to find at the outputs (though the expectation can be for several cycles later). Label `exp:` is an abbreviation for expected outputs, and label `MOD:` indicates module outputs. (A reminder of what the lower- and upper-case labels signify.)

Each `a` and `b` input has two values. Both values appear next to `A` and `B`, shown in hexadecimal. In the first tests the `a` and `b` vectors are set to make it easy to debug problems: each component of vector `a` is 1. The even components of `b` are 1 and the odd components of `b` are set to $16_{10} = 10_{16}$.

Consider the trace for cycle 9. The dp output is 55 and the IDs are a8 (first) and b1 (last). The vectors started arriving in cycle 2 (note the matching a8 on the module input, labeled ID), and the last pair of elements arrived at cycle 6. Though the vector finished arriving at cycle 6, its dot product did not appear at the module output until cycle 9. Also consider cycle 8. A new vector starts arriving, as one can tell by the first=1 signal. The newly arriving vector does not interfere with the computation of the vector that finished arriving in cycle 6. Dot products for both will be computed.

The testbench will print a summary of results at the end, for the correct solution:

```
Done with 10000 tests. 0 dp errs ( 9718 correct )
Done with 10000 tests. 0 FID errs, 0 LID errs.
Done with 10000 tests. Correct 9718, avg latency 3.0 cyc Okay
```

The first line above indicates that there were zero errors with the dp output. A total of 10000 vectors were offered to the module and the module finished 9718 of them. The $10000 - 9718 = 282$ other vectors were interrupted by the reset signal. In the second line FID errs indicates how many first_id outputs were incorrect, the same for LID errs. The last line indicates the average number of cycles to compute a dot product. In this assignment that's expected to be an integer, and equal to 3.

When there are errors the trace indicates which outputs are wrong by changing labels FID, LID, and DP to ERR if the respective output value is incorrect. The expected correct values of first_ID and last_ID are based on the dp output. For example, if first_ID and last_ID are changed for a new vector but dp shows the dot product of the previous vector, then the values of first_ID and last_ID will be considered wrong and so FID and LID will show ERR.

If the dp value is correct but arrives in less than 3 cycles an error message will be printed. Also, the DP label will be changed to EY (early) for a correct dot product value that appears earlier than 3 cycles after last arrived.

Note that the exp (lower case) labels show values that are expected now or in the future. For example, in cycle 6 below the dp label shows a 55 based on the arrival of the last signal. However, that dp value should not be seen at the outputs until 3 cycles later.

Also additional detail is provided in the first line following an error. Consider the output from the unmodified assignment:

```
   2 _F_ ID a8  A 01,01  B 01,10  exp: fid 00  lid 00  dp 00  MOD: FID 00  LID 00  EY 11
   3 ___ ID aa  A 01,01  B 01,10  exp: fid 00  lid 00  dp 00  MOD: FID 00  LID 00  DP 22
   4 ___ ID ad  A 01,01  B 01,10  exp: fid 00  lid 00  dp 00  MOD: FID 00  LID 00  DP 33
   5 ___ ID af  A 01,01  B 01,10  exp: fid 00  lid 00  dp 00  MOD: FID 00  LID 00  DP 44
   6 __L ID b1  A 01,01  B 01,10  exp: fid a8  lid b1  dp 55  MOD: ERR 00  LID b1  EY 55
Error first ID: 0 != a8 (correct)
Error dp timing: latency 0 cyc < 3 cyc (minimum)
   7 ___ ID b3  A 01,01  B 01,10  exp: fid a8  lid b1  dp 55  MOD: FID 00  LID b1  ER 66
Error dp: 66 != 55 (correct)
```

At cycle 6 output dp is set to the correct value, 55 (see the dp item) as is the LID, but since dot product arrives early the label is EY rather than DP. Also, the first_id output is incorrect, 0, so it is labeled ERR in the trace and the correct value is shown, a8. At cycle 7 the DP output is incorrect, it should have stayed at 55.

Helpful Examples

A pipelined execution unit was the subject of 2016 Homework 6. Other than the need to do

floating-point computation, the 2016 Homework 6 pipeline was simpler than the one in this assignment. Another pipelined calculation (something to be covered in class on on Friday 15 November 2024) is the subject of 2021 Homework 6. Past assignments and their solutions are linked to the assignments and exams page.

Problem 1: In the unsolved assignment `dot_seq_2` computes the correct dot products, but does not accompany those with the correct `first_id` value, and the synthesized module would have a longer critical path than desired. Complete the module so that `first_id` is set to the correct ID and meeting the following timing requirements:

- ☐ Don't perform any computation on values arriving at the inputs. Instead, write them to registers and start computation in the next cycle.
- ☐ The critical path can include at most one arithmetic operation on the vector components. For example, don't compute `a[0]*b[0]+a[1]*b[1]` because that operation has a critical path of 2 operations: a multiply and add. Instead compute the products in one cycle and the sum in another.
- ☐ The correct `dp` should be available three cycles after the `last=1`.
- ☐ Once the `dp`, `first_id`, and `last_id` outputs are set they should remain unchanged until either the `reset` is asserted or until the next dot product is ready.
- ☐ The module must be synthesizable. Verify with the command `genus -files syn.tcl`.