LSU EE 4755 Homework 2 Due: 18 September 2020

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/2020/hw02.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, `hw02.v`. Do this early enough so that minor problems (*e.g.*, password doesn't work) are minor problems.

Problem 1: Module `nn4x4`, below, has two inputs, a 4-element vector `ai` and a 4×4 matrix `wht`, and one output, a 4-element vector `ao`. Output `ao` is set to the product of `wht` and `ai`. Parameter `ww` (width of weight) gives the number of bits in the elements `wht` and parameter `wa` (width of activation) gives the number of bits in the elements of `ai` and `ao`.

The illustration below the module shows hardware that might be inferred for `nn4x4`. The illustration also includes three dotted green boxes. These are suggestions on how to hierarchically decompose this large, some would say unwieldy, module.

The two smaller boxes, labeled `nn1x2b`, show hardware computing part of one output using two inputs. The larger box, labeled `nn1x4b` shows hardware computing one output using four inputs. As those who took the time to look at the illustration might have guessed by now the module suggested by `nn1x4b` can be constructed using two instances of `nn1x2b`. Further, a `nn4x4b` can be constructed using four instances of `nn1x4b`. Sounds interesting? Good!

The homework file `hw02.v` contains module `nn4x4`, it is there for your reference. The file also contains mostly empty modules `nn4x4b`, `nn1x4b`, and `nn1x2b`. Complete these so that they compute the same output as `nn4x4` and are constructed as suggested in the illustration and follow the guidelines below.

Module `nn4x4b` must instantiate exactly four `nn1x4b` modules and `nn1x4b` must instantiate exactly two `nn1x2b` modules. Module `nn1x4b` will also need an adder. Module `nn4x4b` has parameters. Don't change them. The other modules should have similar parameters with the same default values as `nn4x4b`. Do not ignore the parameters when declaring inputs and outputs. A standing rule in this class is that all code must be clearly written.

The modules must be synthesizable. This should not be a challenge for this assignment. Verify synthesizablity by running the synthesis script using the command `genus -files syn.tcl`.

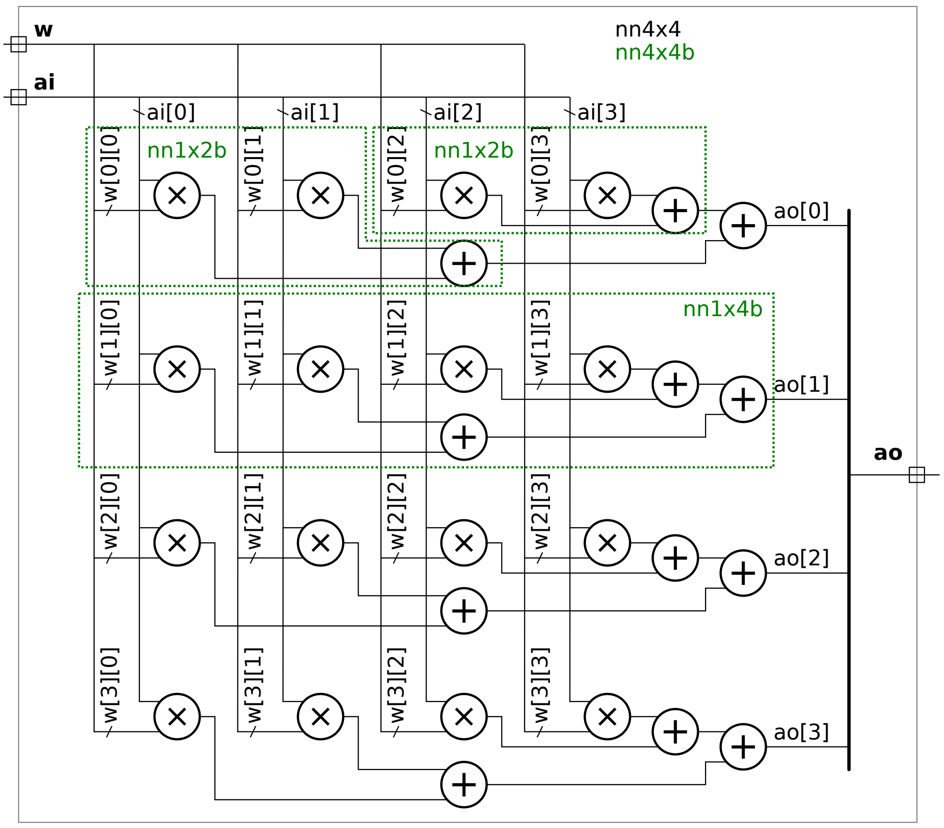
Those who fear they might forget to address some part of the problem described here can rest easy. There is a checklist in the part of the Verilog file where the solution goes.

To help you solve the problem in stages the testbench will perform three rounds of tests. In the first round, labeled `n12`, only output `ao[0]` will be examined and only inputs `ai[0]` and `ai[1]` will have non-zero values. In the second round, labeled `n14`, only output `ao[0]` will be examined but all inputs will have non-zero values. The full test, all outputs checked and all inputs are non-zero, is labeled `n44`.

Some might find it helpful to look at two past homework assignments in which a flat module was to be decomposed hierarchically. The simpler one (perhaps) is 2019 Homework 1, in which a multiplier is decomposed. But the multiplier had two scalar inputs, `a` and `b`. In this (2020) assignment one input is a 1-D array (`ai`) and the other is a 2-D array (`wht`). In the Fall 2017 Homework 1 Problem 2 an 8-input multiplexor is to be decomposed. The mux input `a` is a 1-D array that had to be split between two instances.

Module and illustration on the next page.

```
module nn4x4
  #( int wa = 10, ww = 5 )
   ( output uwire [wa-1:0] ao[4],
     input uwire [wa-1:0] ai[4],
     input uwire [ww-1:0] wht[4][4] );

   assign ao[0] = ai[0] * wht[0][0] + ai[1] * wht[0][1]
          + ai[2] * wht[0][2] + ai[3] * wht[0][3];

   assign ao[1] = ai[0] * wht[1][0] + ai[1] * wht[1][1]
          + ai[2] * wht[1][2] + ai[3] * wht[1][3];

   assign ao[2] = ai[0] * wht[2][0] + ai[1] * wht[2][1]
          + ai[2] * wht[2][2] + ai[3] * wht[2][3];

   assign ao[3] = ai[0] * wht[3][0] + ai[1] * wht[3][1]
          + ai[2] * wht[3][2] + ai[3] * wht[3][3];

endmodule
```


w
ai
nn4x4
nn4x4b
ai[0]
ai[1]
ai[2]
ai[3]
nn1x2b
nn1x2b
nn1x4b
w[0][0]
w[0][1]
w[0][2]
w[0][3]
w[1][0]
w[1][1]
w[1][2]
w[1][3]
w[2][0]
w[2][1]
w[2][2]
w[2][3]
w[3][0]
w[3][1]
w[3][2]
w[3][3]
ao[0]
ao[1]
ao[2]
ao[3]
ao