LSU EE 4755 Homework 5 Due: 12 October 2018

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/2018/hw05.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, `hw05.v`.

Homework Overview

The sorting networks used in past assignments were not very efficient, they were the rough hardware equivalent of bubble sorts. In this assignment much better sorters will be implemented, sorting networks based on Batcher's odd/even merge design.

Testbench Code

The testbench for this assignment, which can be run when visiting the file in Emacs in a properly set-up account by pressing F9, tests module `batcher_sort` and `batcher_merge` at several different sizes.

Here is a transcript showing the output of the testbench (after the compiler's own messages):

```
ncsim> run
Starting testbench.
Mod batcher_merge, n=2, sort  1 idx  0, wrong elt  18 !=   7 (correct)
Mod batcher_merge, n=2, sort  1 idx  1, wrong elt   7 !=  18 (correct)
Mod batcher_merge, n=2, sort  4 idx  0, wrong elt 216 != 120 (correct)
Mod batcher_merge, n=2, sort  4 idx  1, wrong elt 120 != 216 (correct)
Mod batcher_merge, n=2, sort  7 idx  0, wrong elt 150 !=  12 (correct)
Tests for batcher_merge (idx 1)  n=2 done, errors in 3 of 10 sorts.
Tests for batcher_merge (idx 2)  n=4 done, errors in 6 of 10 sorts.
Tests for batcher_merge (idx 3)  n=8 done, errors in 10 of 10 sorts.
Tests for batcher_merge (idx 4)  n=16 done, errors in 10 of 10 sorts.
Tests for batcher_merge (idx 5)  n=32 done, errors in 10 of 10 sorts.
Tests for batcher_sort (idx 7)  n=2 done, errors in 2 of 10 sorts.
Tests for batcher_sort (idx 8)  n=4 done, errors in 10 of 10 sorts.
Tests for batcher_sort (idx 9)  n=8 done, errors in 9 of 10 sorts.
Tests for batcher_sort (idx 10)  n=16 done, errors in 10 of 10 sorts.
Tests for batcher_sort (idx 11)  n=32 done, errors in 10 of 10 sorts.
Done with all tests, errors on 10 sorters.
```

The transcript shows the first five errors in detail, this is on lines starting with `Mod`. A tally of the total number of errors for a particular module is shown on a line starting `Tests for`.

Here is the output when the assignment is correctly solved:

```
ncsim> run
Starting testbench.
Tests for Batcher Merge (idx 1)  n=2 done, errors in 0 of 10 sorts.
Tests for Batcher Merge (idx 2)  n=4 done, errors in 0 of 10 sorts.
Tests for Batcher Merge (idx 3)  n=8 done, errors in 0 of 10 sorts.
Tests for Batcher Merge (idx 4)  n=16 done, errors in 0 of 10 sorts.
Tests for Batcher Merge (idx 5)  n=32 done, errors in 0 of 10 sorts.
Tests for Batcher Sort (idx 7)  n=2 done, errors in 0 of 10 sorts.
```

```
Tests for Batcher Sort (idx 8)  n=4 done, errors in 0 of 10 sorts.
Tests for Batcher Sort (idx 9)  n=8 done, errors in 0 of 10 sorts.
Tests for Batcher Sort (idx 10)  n=16 done, errors in 0 of 10 sorts.
Tests for Batcher Sort (idx 11)  n=32 done, errors in 0 of 10 sorts.
Done with all tests, errors on 0 sorters.
```

Debugging

To debug your code run SimVision: `irun -gui hw05.v`. Locate your module and copy symbols to the waveform viewer. See the SimVision instructions on the `https://www.ece.lsu.edu/koppel/v/proc.html` page.

Synthesis

The synthesis script, `syn.tcl`, will synthesize `sort2` with two delay targets, an easy 10 ns and a unachievable 0.1 ns. If the module doesn't synthesize −.001 s is shown for the delay. The script is run using the shell command `genus -files syn.tcl`, which invokes Cadence Genus. In past semesters Cadence RTL Compiler (`rc`) was used, which would be invoked using `rc -files syn.tcl`, **but that won't work on the 2018 homework assignments**.

The synthesis script shows area (cost), delay, and the delay target in a neat table. Additional output of the synthesis program is written to file `spew.log`.

Problem 1: Complete module `batcher_sort` so that it implements a sorter as described below. The module has one input, an n-element array `a`, and one output, an n-element array `x`. Above some minimum value of `n` it should instantiate two copies of itself, each copy should sort half the the input array, `a`. A `behav_merge` module should be instantiated to merge the output of the two recursive implementations.

The `behav_merge` module, which is already written, has two inputs, `a` and `b`, each an n-element array, and one output, `x`, a $2n$-element array, where n is the value of the first parameter. Output `x` contains the elements of `a` and `b` in sorted order.

Once Problem 2 is solved correctly replace `behav_merge` with `batcher_merge`.

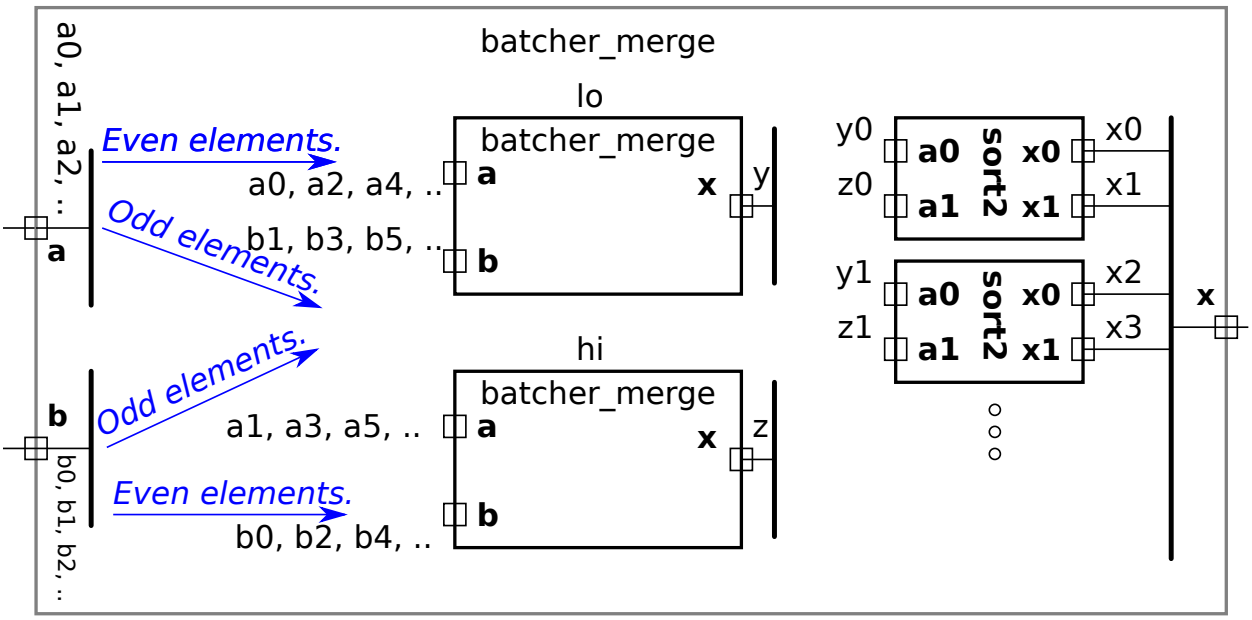
The module must use structural code, be synthesizable, be reasonably efficient, clearly written, and of course pass the testbench. See other conditions on the solution and tips in the Verilog file.

The solution to this problem is straightforward and will be in the form of other tree-structured designs shown in class.

Warning: Do not search for a solution to this problem. Exam questions will be written under the assumption that each student has solved all homework problems.

Problem 2: Complete module `batcher_merge` so that it recursively implements a Batcher odd/even merge module in which the number of elements of each input list is a power of 2. Use `sort2` instantiations to combine the output of the recursively instantiated modules. Use either structural or behavioral code to separate each input sequence into odd and even parts.

The `batcher_merge` module should recursively instantiate two copies of itself, call them lo and hi. Input `a` of the lo module should connect to the even-numbered `a` elements of the enclosing module, input `b` of lo connects to odd-numbered `b` elements of the enclosing module. For the hi module switch odd and even. See the illustration below. The illustration also shows how the outputs should connect.



Warning: Do not search for a solution to this problem. Exam questions will be written under the assumption that each student has solved all homework problems.

The module must be synthesizable, reasonably efficient, clearly written, and of course pass the testbench.

Do not compare the cost and performance reported by genus for your module, `batcher_merge`, to those for `behav_merge`. That's because genus does not correctly infer hardware for `behav_merge`.