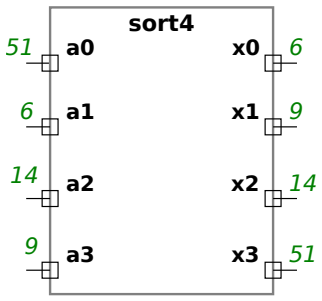
LSU EE 4755 Homework 1 Due: 5 September 2018

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/hw01.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, `hw01.v`. Do this early enough so that minor problems (*e.g.*, password doesn't work) are minor problems.

Homework Overview

An n-input *sorting network* is a combinational circuit with n inputs and n outputs. The n values at the inputs appear at the outputs in sorted order. The illustration below shows a four-input sorting network with example values shown in green.



File `hw01.v` contains correctly functioning 2-input and 3-input sorting networks, `sort2_is`, and `sort3`. Modules `sort2` and `sort4` are empty and are to be completed for this assignment as described in the problems. File `hw01.v` contains several other modules for use in solutions, and a testbench.

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 four modules: `sort2_is`, `sort2`, `sort3`, and `sort4`. Modules `sort2_is` and `sort3` should pass, the others await your solution. A sample of the end of the testbench output appears below:

```
Mod sort2, sort  2 index  0, wrong elt   z !=   0 (correct)
Tests for sort2 done, errors in         100 of         100 sorts.
Tests for sort2_is done, errors in          0 of         100 sorts.
Tests for sort3 done, errors in           0 of         100 sorts.
Mod sort4, sort  0 index  0, wrong elt   z !=  24 (correct)
Mod sort4, sort  0 index  1, wrong elt   z !=  26 (correct)
Mod sort4, sort  0 index  2, wrong elt   z !=  64 (correct)
Mod sort4, sort  0 index  3, wrong elt   z !=  94 (correct)
Mod sort4, sort  1 index  0, wrong elt   z !=   0 (correct)
Tests for sort4 done, errors in         100 of         100 sorts.
ncsim: *W,RNQUIE: Simulation is complete.
ncsim> exit

Compilation finished at Tue Aug 28 16:53:25
```

A count of the number of tests and errors is shown for four modules. The testbench shows the first five errors it finds on each module, to see more modify the testbench (search for `g_elt_err_count`). In the output above the testbench is showing that the module outputs are `z` (an unconnected wire) which of course don't match the expected outputs.

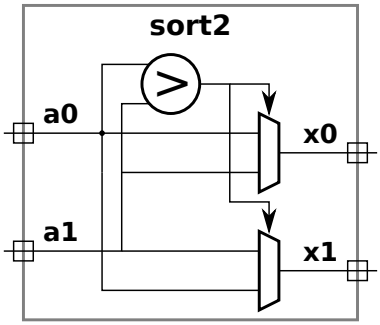
Use Simvision to debug your modules. Feel free to modify the testbench so that it presents inputs that facilitate debugging.

Synthesis

The synthesis script, `syn.tcl`, will synthesize the four modules each with two delay targets, an easy 10 ns and a un-achievable 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`. Sample synthesis script output appears below:

Problem 1: Complete module `sort2` so that it implements a 2-input sorter using a comparison unit and two 2-input multiplexors, as illustrated below. The module must pass the testbench and be synthesizable.



Use only structural code in the module (do not use `assign`, `initial`, or `always` blocks). Instantiate `mux2` for the multiplexors and `compare_le` for the comparison unit. See the check boxes in `hw01.v` near the problem for other requirements and tips.

Problem 2: Complete module `sort4` so that it implements a 4-input sorting network. Do so by instantiating `sort3` and `sort2` (or `sort2_is`) modules. As with `sort2`, use only structural code and make sure that the module passes the testbench and synthesizes.

For this assignment, implement `sort4` using one `sort3` and several `sort2` modules. Use the `sort2` modules to find the largest of the four inputs to `sort4` and connect that largest value to output `x3`. Use `sort3` to handle the remaining three values.

Implement `sort4` to minimize the critical path (measured in `sort2` or `sort2_is` modules). That is, minimize the maximum number of `sort2` (or `sort2_is`) modules traversed by any signal. The critical path for `sort3` is 3: from input `a0`, through `s0_01`, `i11`, `s1_12`, `i21`, `s2_01`, to output `x0`.

The `sort3` module uses three `sort2_is` modules. Feel free to examine `sort3` to see how modules are instantiated and interconnected.