LSU EE 4755 **Homework 4** **Due: 12 October 2016**

Problem 0: First, follow the instructions for account setup and homework workflow on the course procedures page, `http://www.ece.lsu.edu/koppel/v/proc.html`.

Look through the code in `hw04.v`. Module `lookup_behav` in file `hw04.v` has a w-bit input `char` and an n-element array of w-bit quantities named `chars`. (Parameter `nelts` is n and parameter `charsz` is w.) The module also has a 1-bit output `found` which is logic `1` iff any element of `chars` is equal to `char`. Finally, the module has a $\lceil \lg n \rceil$-bit output `index` which is set to the element number of `chars` that matches `char`, or `0` if `found` is `0`. Assume that no two elements of `chars` are identical.

For example, suppose input `char` is set to `102` and that `chars` is `{63,124,102,92}`. Then output `found` will be `1` and `index` will be `2`. If `char` were `7` `index` would be `0` and `found` would be `0`, if `char` were `63` `index` would be `0` and `found` would be `1`, etc. The alert student will have recognized that $n = 4$ and that $w \geq 7$ in these examples.

Module lookup is coded in synthesizable behavioral form that describes combinational logic. The `hw04.v` file contains two other modules which are to do the same thing, `lookup_linear` and `lookup_tree`, but those modules are not yet finished.

The testbench tests all of these modules. It tests them for sizes (n) of 4, 5, 10, 15, 16, 30, 40, and 64. To change which sizes are tested (or the order in which they are tested) edit the `testbench` module.

To have the testbench test only some of these modules (say, skip the `lookup_tree` tests until after `lookup_linear` is working) look for the `for` loop with `mut=0` and modify it appropriately. (It should be easy to figure out the numbers.)

A synthesis script is provided that will synthesize all three modules at different sizes and both with and very lax timing constraint and a very strict timing constraint. The script can be run using the command `rc -files syn.tcl`. Initially it will stop with an error. To see it run to completion before starting the assignment have it only synthesize `lookup_behav` (see below). Pre-set synthesis options (in file `.synth_init`) were chosen to reject any design that is not combinational.

If there is an error when using the synthesis script then follow the manual synthesis steps on the procedures page and look for error messages.

To change which modules are synthesized edit the `set modules` line (near the bottom) in file `syn.tcl`. The values for `nelts` and other items can also be changed by editing the file.

Note: There are no points for this problem.

Problem 1: Complete `lookup_linear` so that it does the same thing as `lookup_behavioral` but by using as many copies of `lookup_elt` as it needs. That is, `lookup_linear` should use generate statements to instantiate `lookup_elt` and it should include whatever other code is needed to use these instances to compute the correct outputs.

- Behavioral or structural code can be used.
- The module must be synthesizable.
- Assume that all elements of `chars` are different.

Problem 2: Complete module `lookup_tree` so that it performs the lookup using recursive instantiations of itself. Take care so that `index` is computed efficiently. *Hint: think about how to compute index efficiently when n (*`nelts`*) is a power of 2, then get the same efficiency for any n.*

If completed correctly, the cost and especially the performance at larger sizes should be better than `lookup_behavioral` and (unless you did an unexpectedly good job) better than `lookup_linear`.

- Behavioral or structural code can be used.
- The module must be synthesizable.
- Assume that all elements of `chars` are different.

Problem 3: Run the synthesis script and characterize the strengths and weaknesses of each module. (For example, module X has lowest cost for low-speed designs.)

In a follow-on homework assignment additional questions will be asked about these modules.