EE 4702 **Homework 5** **Due: 2 May 2001**

Solve this problem by modifying a copy of `http://www.ee.lsu.edu/v/2001/hw05.html` *(or .v) which can also be found in* `/home/classes/ee4702/files/v/hw05.v`*. See* `http://www.ee.lsu.edu/v/proc.html` *for instructions on running the simulator. Alternate instructions can be found in Lesson 7 of the ModelSim Tutorial, linked to the references web page,* `http://www.ee.lsu.edu/v/ref.html`*. This page also has links to manuals for the synthesis program, Leonardo. The links are clickable when this assignment is viewed with Acrobat Reader. The ModelSim tutorial and other documentation can also be accessed from the Help menu on the ModelSim GUI (started by the command vsim -gui).*

Module `bsearch`, in the homework template, stores numbers and can find whether a number had been seen before. The module has four inputs and an output. Input `clk` is a clock, input `reset` is a reset signal, `op` is a command, and `din` in the number to store or find. The module checks commands on a positive edge of the clock, unlike the calculator a command should be present for just one positive edge. (If it is present for two consecutive positive edges it may be performed twice.) Output `result` should be set by the negative edge following the command (though it can be set right after the positive edge). If the result is ready then `result` is set to the appropriate code, explained below, otherwise it is set to `re_busy` until the result is available. The module recognizes three commands, plus a nop.

When `op = op_insert` the module will attempt to store the number present at input `din`, if successful `result` will be set to `re_i_inserted`. If the module were full `result` is set to `re_i_full`. An inserted number must be strictly greater than the last one inserted, if not `result` is set to `re_i_misordered`. When `op = op_find` the module will set `result` to `re_i_present` if the number at `din` was inserted since the last reset, otherwise it is set to `re_i_absent`. When `op = op_reset` the module is emptied.

The homework template contains four copies of a behavioral description of this module, all named `bsearch` and each bracketed by an `‘ifdef`/`‘endif` pair.

The module just below `‘ifdef NOT_SYN` is complete and does not have to be modified. (If would have been Problem 1 if there were more time left in the semester. `:-)`). The other `bsearch` modules are to be converted into synthesizable form as explained in the problems below.

Problem 1: Convert the module below `‘ifdef FORM2` to a synthesizable module in Form 2 *that does one iteration of the forever loop per cycle*. (The original code does the entire loop in one cycle.) The synthesized module must pass the testbench. In the appropriate place in the comments indicate the clock frequency, area (number of gates), and worst-case time needed to find a number (time from positive edge when `op = op_find` to when `result` is set to `re_f_present`).

Problem 2: Convert the module below `‘ifdef FORM3` to a synthesizable module in Form 3 *that does no more than one iteration of the forever loop per cycle*. The synthesized module must pass the testbench. Show how the critical path (as identified by the synthesis program) can be shortened by adding an event control `@(posedge clk)`. Include the line if that would improve performance, otherwise, include it and comment it out. (Remember that performance is more than just clock frequency.) Next to the line indicate the endpoints of the critical path that is, or would be, shortened.

In the appropriate place in the comments indicate the clock frequency, number of gates, and worst-case time needed to find a number.

Problem 3: Convert the module below `‘ifdef FORM3_FAST` to a synthesizable module in Form 3. The synthesized module must pass the testbench. Modify the description so that two iterations of the original code is done by one iteration (and clock cycle) in the modified code. This module should take fewer clock cycles than the one in the previous problem (nearly half when the capacity is large). In the appropriate place in the comments indicate the clock frequency, number of gates, and worst-case time needed to find a number.

The modules must be synthesizable using the provided synthesis script (see below) and the synthesized hardware must pass the testbench.

Follow these steps:

(1) Modify the modules as needed. Be sure to include a `‘define FOO` when you are working on a module next to `‘ifdef FOO`.

(2) Synthesize the module. This can be done in two ways:

- In Emacs: press `S-F9` (shift f9) while a buffer with the Verilog description is active. Lines containing error, warning, and information messages will be highlighted. If mouse-2 (the middle button) is pressed while the pointer is over a highlighted message Emacs will jump to the corresponding line in the Verilog description.
- From a shell: type `syn.pl hw05sol.v`.

The clock frequency, number of gates, and critical path information are written by the synthesis program and script.

Make sure the module synthesizes (look for a “Synthesis Complete” message), correct any problems if it does not.

(3) Run the testbench on the synthesized module. To do this, load or restart the testbench into Modelsim **without** recompiling it. (The synthesis script should have compiled the synthesized module for you.) If this is done correctly Modelsim should print many lines that look like “Loading work.OR4T2,” the names of the technology modules. Run the testbench and correct any errors.