LSU EE 4755 Homework 4 Due: 11 November 2019

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/2019/hw04.v.html`.

Problem 0: Following instructions at `https://www.ece.lsu.edu/koppel/v/proc.html`, set up your class account (if for whatever reason you haven't done so or neeed to do it again), copy the assignment, and run the Verilog simulator and synthesis program on the unmodified homework file, `hw04.v`. Do this early enough so that minor problems (*e.g.*, password doesn't work) are minor problems.

Homework Overview

Module `best_match_behavioral` has two inputs, a longer vector, `val`, and a short vector, `k`. It sets `pos` to the start of a subvector of `val` that best matches `k` and sets `err` to the number of bit positions that don't match. For example, suppose `val = 8'b11110000` and `k=4'b1100`. Then `pos` would be set to 2 and `err` to 0 because there is an exact match at position 2 in `val`. If `k=4'b1101` then there isn't an exact match for `k` in `val`, but at position 2 there is a match with one error. If `k=2'b00` then there are matches at positions 0, 1, and 2, all with zero errors.

Module `best_match_behavioral` is combinational (and was written as a behavioral module). In this assignment a sequential version will be written and analyzed.

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 the modules. Initially, the testbench will exit because module `best_match` has not responded in sufficient time. When that happens one of the last lines of the testbench output shows that the final cycle count is the same as the cycle limit (128 below), and "CYCLE LIMIT EXCEEDED" is shown.

```
ncsim> run
Exit from clock loop at cycle 128, limit 128.  ** CYCLE LIMIT EXCEEDED **
ncsim: *W,RNQUIE: Simulation is complete.
ncsim> exit

Compilation finished at Mon Nov  4 17:56:24
```

To get rid of this message `best_match` must handshake correctly, see Problem 1. If `best_match` responds in time, the testbench will check to see if `pos` is in the right range. The output below shows errors when `pos` is out of range: `Error in best_match, test #     3, pos out of range: 0xff`

```
Error in best_match, test #     4, pos out of range: 0xff
Done with best_match_behavioral tests,           0 errors found.
Done with best_match tests,        1000 errors found.
Exit from clock loop at cycle 59001, limit 59069.
ncsim: *W,RNQUIE: Simulation is complete.
ncsim> exit
```

The output `err` is supposed to be the number of non-matching bits at `pos`. If not, the testbench shows output like:

```
Error in best_match, test #     2, err wrong 1 != 3 (correct) pos   2  84 ^ 01
Error in best_match, test #     3, err wrong 1 != 2 (correct) pos  13  1f ^ 3d
Error in best_match, test #     4, err wrong 1 != 2 (correct) pos   4  78 ^ f9
Done with best_match_behavioral tests,           0 errors found.
```

```
Done with best_match tests,         972 errors found.
Exit from clock loop at cycle 59001, limit 59069.
ncsim: *W,RNQUIE: Simulation is complete.
ncsim> exit
```

For test # 4, the testbench reports that `err` was 1 but should have been 2. The line also shows that `pos` was set to 4, and that `val` at that position was `78` (in hexadecimal) and that `k=f9`.

The testbench also checks whether the `err` returned is the minimum error for that value of `val` and `k`.

The testbench prints the details of the first few errors it finds. A grand total is printed at the end, see the transcript above.

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 `best_match_behavioral` (for reference) and `best_match` (your solution). Each module will be synthesized at three widths, and with two delay targets, an easy 90 ns and a un-achievable 0.1 ns. If a module doesn't synthesize −.001 s is shown for its delay. The script is run using the shell command `genus -files syn.tcl`, which invokes Cadence Genus. If you would like to synthesize additional modules or sizes edit `syn.tcl` near the bottom.

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-file.log`.

Problem 1: Complete module `best_match` so that it computes the best match sequentially as described below. In addition to `val` and `k`, the module has 1-bit inputs `start` and `clk` and 1-bit output `ready`.

Handshaking works as follows: When `start=1` at a positive edge the module should set `ready` to zero. It should then start scanning for the best match, checking one shifted position per cycle. The maximum number of cycles needed should be `wv-wk` plus one or two more needed for handshaking. (The testbench will wait `2*wv` cycles before giving up.) The module should set `err` and `pos` to their correct values and `ready` to 1.

The inputs, `val` and `k` will be held steady at least until `ready` is set to 1.

The module must use the `pop` (population) module (in `hw04.v`) to compute possible values for `err`. That is, don't use something like the `b` loop in `best_match_behavioral` to accumulate the sum `e`. Instead compute the XOR of the appropriate bit range and provide that to the `pop` module as an input.

For maximum credit avoid the use of large (such as `wv`-input) multiplexors in your design, or the use of a non-constant shifter.

The module must be synthesizable and correct.

The behavioral best match module is shown below for reference.

```
module best_match_behavioral
  #( int wv = 32, int wk = 10, int wvb = $clog2(wv), int wkb = $clog2(wk+1) )
   ( output logic [wvb:1] pos,  // Position of best match.
     output logic [wkb:1] err,  // Number of non-matching bits.
     input uwire [wv-1:0] val, input uwire [wk-1:0] k );

   always_comb begin

      automatic int best_err = wk + 1;
      automatic int best_pos = -1;
```

```
      for ( int p=0; p<=wv-wk; p++ ) begin
         automatic int e = 0;
         for ( int b=0; b<wk; b++ ) e += k[b] !== val[p+b];
         if ( e < best_err ) begin
            best_err = e;
            best_pos = p;
         end
      end
      err = best_err;
      pos = best_pos;

   end

endmodule
```

Problem 2: Run the synthesis program and indicate how your module compares to the behavioral module.

(*a*) Compare the amount of time needed for your module compared to the behavioral one. The answer to this question requires some manipulation of the values in the `Delay Actual` column. Indicate which results are expected, and which are not expected, and explain why.

(*b*) Compare the area of your design to the behavioral one. Indicate which results are expected, and which are not expected, and explain why.