LSU EE 4755 Homework 6 Due: 2 December 2015

Problem 0: The homework Verilog file, `hw06.v`, contains something similar to the integer compression modules presented in class. (Follow the homework workflow instructions on the course procedures page to get a copy of the assignment package.) These modules compress an ASCII character stream by substituting a binary-encoded integer for a string of ASCII digits. These modules were based on 2014 Homework 4. Feel free to look at that assignment an solution for help.

Module `icomp_none` is a version of the module that does no compression at all. It does though implement the handshaking protocol so that characters can be passed from input to output. This module can be studied to help understand how the others work.

Module `icomp_2cyc` is one of the compression modules covered in class. It computes the encoded value in stage 0, and checks for overflow in stage 1. Don't modify this module, save if for reference. Module `icomp_sol` is initially identical to `icomp_2cyc`, but it should be modified as part of this assignment.

The testbench is set to simulate `icomp_sol` on a sample test string. At the end it will report the amount of compression and whether there was any errors. The testbench also prints out a trace showing some module inputs and outputs and the status of internal signals. Examine the testbench code to see how this is done and feel free to modify it to add signals of your own. A more detailed trace of execution can be obtained using the SimVision gui. To start that use the command `irun hw06.v -gui`. See `http://www.ece.lsu.edu/koppel/v/v/s/SimVisionIntro.pdf` for documentation. (On campus access only without password.)

The synthesis script will synthesize the modules `icomp_2cyc` and `icomp_sol`. Use the synthesis script to make sure that your designs are synthesizable and to determine their cost and performance.

(There is nothing to turn in for this assignment.)

Problem 1: In module `icomp_sol` there is a declaration of a variable named `val_encode_size_1`, but no uses of that variable. Add code to that module so that `val_encode_size_1` is set to the number of bytes that are needed for the number currently in the register `val_encode_1`. For example, if `val_encode_1` has a 0, then `val_encode_size_1` should be 0. If `val_encode_1` has a 123 then `val_encode_size_1` should be 1 (one byte), if `val_encode_1` has a 300 then `val_encode_size_1` should be 2 (for 2 bytes), etc.

To help with your solution add code to the testbench to show the value of this variable.

Problem 2: Modify module `icomp_sol` so that a group of ASCII digits is compressed into the smallest number of bytes needed, up to `max_chars`. For example, if `max_chars` is 4 then just use one byte to compress 200, two bytes for 4000, and for 1234567890123 use a four-byte integer (for 1234567890) followed by a one byte integer (for 123).

Precede the compressed integer by the character 128 plus the number of bytes in the compressed number. For example, if the compressed value takes two bytes then where the first character of the uncompressed value would go emit a 130, then the next two characters should be the compressed number. (See how `char_out` is assigned in the unmodified code.)

To solve this problem you'll need to understand how the existing code works, how to interpret the trace output provided by the simulator, and how to use the SimVision waveform viewer. Random guesses based on a vague understanding will get you nowhere.

- The module should be written for arbitrary values of `max_chars`.
- Make sure that the testbench is not reporting errors.

- Make sure that your module is compressing the string.