LSU EE 4755 **Homework 4** **Due: 24 November 2014**

Problem 0: Copy the code package from `/home/faculty/koppel/pub/ee4755/hw/2014f/hw04`. Verify that everything is working by running the simulation on the unmodified file. It should report that there is correct output but no compression:

```
Correct output, strings match. But no compression!
In size         117 bytes, out size         117 bytes.
```

Problem 1: Module `asc_to_bin` is to filter a stream of ASCII characters so that ASCII decimal numbers are replaced by binary numbers preceded by an escape character. The idea is to reduce the size of data streams that contain lots of large numbers. For example, consider the sentence, "There are 31536000 seconds in a year." The module `asc_to_bin` should replace that sequence of eight ASCII characters 31536000 with an escape character and an integer encoding of the number.

The module has an 8-bit input and output for the character, `char_in` and `char_out`. There is a 1-bit input `can_insert` which is true when the module can read a character from `char_in`. If input `insert_req` is asserted when `can_insert` is true then the character on `char_in` will be read.

There is a 1-bit output `can_remove` which is true when the character on `char_out` is valid. (It would not be valid if the module does not contain any characters and for other reasons.) If input `remove_req` is set to `1` and `can_remove` is true then the character at `char_out` will change to the next character or, if that's the last available character, `can_remove` will go to zero.

There is also a 1-bit input `reset`. If `reset` is high at the positive edge of the clock then the module should reset itself.

Initially in the homework package, module `asc_to_bin` passes through characters unchanged. Modify it so that it converts ASCII decimal numbers to binary as described above.

At the end of the simulation the testbench will indicate whether the output string is correct, and the original and compressed sizes. For example, the output using the unmodified code package will be:

```
Correct output, strings match. But no compression!
In size         117 bytes, out size         117 bytes.
```

The testbench also provides a trace showing some information each time a character is removed. For the unmodified code,

```
ncsim> run
c  79 = O   tail  1 head  0
c 110 = n   tail  3 head  1
c 101 = e   tail  4 head  2
c  32 =     tail  7 head  3
c  49 = 1   tail  8 head  4
```

The character removed is shown as a decimal number and as a character, for example 110 and "n" for the second line. Also shown are the values of two objects in the `asc_to_int` module, `tail` and `head`. Feel free to add your own variables to the list. Search for "Trace execution" to find the code that prints this trace.

The parameter `max\_chars` indicates the maximum size of the integer that should be created. Currently the testbench expects all integers to be of this size.

Keep the following in mind:

- Do not convert a number to binary if it would take more space than the original.
- The module must be synthesizable.
- The synthesized hardware must be reasonably efficient.

For extra credit, modify both the `asc_to_bin` module and the testbench so that `asc_to_bin` can compress a string of ASCII digits to the smallest integer (in multiple of bytes) that can hold the integer. (The current behavior is to use one size integer, determined by parameter `max_chars`.)

Problem 2: Synthesize your module.

(*a*) Indicate the cost and performance with and without timing optimization. (With timing optimization means using `define_clock`.)

(*b*) Even if `define_clock` is used, the synthesis program won't optimize all paths, only those with both ends affected by the clock. Show how to use the Encounter `external_delay` command to get the proper timing optimization.