LSU EE 4755 Homework 1 Due: 24 September 2021

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/2021/hw01.v.html`.

Problem 0: Following instructions at `https://www.ece.lsu.edu/koppel/v/proc.html`, set up your class account, copy the assignment, and run the Verilog simulator and synthesis program on the unmodified homework file, `hw01.v`. Do this early enough so that minor problems (*e.g.*, password doesn't work) are minor problems.

Problem 1: The partially completed `insert_at` module below and in the homework assignment file has three inputs, a `wa`-bit input `ia`, a `wb`-bit input `ib`, and a $\lceil \lg(\texttt{wa}+1) \rceil$-bit input `pos`, and there is one output, a `wa+wb`-bit output `o`. Complete the module following the coding requirements given further below so that `o` consists of the bits of `ia` with `ib` inserted at `pos`. That is, `o[pos-1:0]` should be set to `ia[pos-1:0]`, `o[wb+pos-1:pos]` should be set to `ib`, and `o[wa+wb-1:wb+pos]` should be set to `ia[wa-1:pos]`.

For example, let `wa=6` and `wb=2`, `ia=111111`, `ib=00`, and `pos = 2`. Then `o=11110011`. For `pos=5`, `o=10011111`. For those still not 100% sure of what `o` should be set to should look at how `o_shadow` is computed in the `testbench` module. Also, the testbench will show what the output should be when it isn't.

```
module insert_at
  #( int wa = 20, wb = 10, wo = wa+wb, walg = $clog2(wa+1) )
   ( output logic [wo-1:0] o,
     input uwire [wa-1:0] ia, input uwire [wb-1:0] ib,
     input uwire [walg-1:0] pos );

   // The line assigning mask_low must be replaced with a mask module.
   uwire [wo-1:0] mask_low = ( 1 << pos ) - 1;  // REPLACE ME!

   uwire [wo-1:0] ib_at_pos;
   shift_left #(wb,wo,walg) sl1( ib_at_pos, ib, pos );

   assign o = ia & mask_low | ib_at_pos;
endmodule
```

The `insert_at` module must be synthesizable and must not use procedural code and must not use shift operators. (That includes the line assigning `mask_low`, it must be replaced.) Instead, rely on instantiations of the provided shift and mask modules.

The testbench will test your module and report the first few errors. For example, here is the testbench output for the unmodified module:

```
Error for ia=11111111  ib=000  pos= 0  00000000000 != 11111111000 (correct)
Error for ia=11111111  ib=000  pos= 1  00000000001 != 11111110001 (correct)
Error for ia=11111111  ib=000  pos= 2  00000000011 != 11111100011 (correct)
Error for ia=11111111  ib=000  pos= 3  00000000111 != 11111000111 (correct)
Error for ia=11111111  ib=000  pos= 4  00000001111 != 11110001111 (correct)
Done with 27 tests, 15 errors found.
```

The text `00000001111 != 11110001111 (correct)` shows the output of `insert_at` to the left of the `!=` and the correct answer to the right. So in this case `00000001111` is the module output

and 11110001111 is what the module output should have been. Only the first few errors are shown, but the total number of errors is reported at the end, 15 in this case.

Synthesizability can be checked by running the synthesis script using the command `genus -files syn.tcl`. If the module is synthesizable (though not necessarily correct) a table of area and delay will be shown, for example:

```
Module Name                              Area   Delay   Delay
                                              Actual  Target
insert_at                               51832   0.987   1.000 ns
insert_at_1                             97968   0.616   0.100 ns
Normal exit.
```

One common problem encountered by beginners is setting the correct port sizes. For example, the `shift_left` module the port sizes are `wi`, `wo`, and `wolg`:

```
module insert_at #( int wa = 20, wb = 10, wo = wa+wb, walg = $clog2(wa+1) )
   ( output logic [wo-1:0] o,
     input uwire [wa-1:0] ia, input uwire [wb-1:0] ib,
     input uwire [walg-1:0] pos );
   uwire [wo-1:0] ib_at_pos;
   shift_left #(wb,wo,walg) sl1( ib_at_pos, ib, pos );
```

So the first connection to a `shift_left` instantiation must be `wi` bits, the second must be `wo` bits, and the third `wolg` bits. In the unmodified `insert_at` these parameters to `insert_at` were set explicitly to match the connection sizes. Sometimes it may be necessary to use an intermediate object or to cast in order to get the correct connection size. For example, if we wanted to shift by `pos+1` the following would not work:

```
   shift_left #(wb,wo,walg) sl1( ib_at_pos, ib, pos + 1 );
```

because the `1` in the `pos+1` expression implicitly expands it to 32 bits. (This results in a warning, but it's not good to clutter compiler output with ignorable warnings.) The problem can be solved using a cast:

```
   shift_left #(wb,wo,walg) sl1( ib_at_pos, ib, walg'(pos + 1) );
```