LSU EE 4755 Homework 3 Due: 18 October 2021

To help solve the problems below, look at problems listed in the simple model slides, 2020 Homework 4, 2019 Midterm Exam Problem 2b and c, and especially 2018 Final Exam problems 1 and 2.

Problem 1: As requested in the subproblems below use the simple model to determine the cost and delay of the `insert_at` module from the solution to Homework 1 (see last page) instantiated with `wa` $= w_a$ and `wb` $= w_b$, and using $C_{\rm lsb}(w_a)$ for the cost of the `mask_lsb` module and $D_{\rm lsb}(w_a)$ for the delay of the `mask_lsb` module. The `wo` and `walg` parameters are not set so you can use their default values, $w_o = w_a + w_b$, $l_a = \lceil \lg(w_a+1) \rceil$, and $l_b = \lceil \lg w_b \rceil$, in your answers.

For partial credit, and to help you solve the problems provide a sketch of the inferred hardware. It may help to first solve the problem for specific values of w_a and w_b, and then to generalize for arbitrary w_a and w_b.

(*a*) Find the cost and delay of the hardware inferred for the line of Verilog from `insert_at` shown below. Just for the hardware described on the line. There's no trick, this part is easy. Just remember to express your answers in terms of w_a, w_b, and w_o.

```
   assign o = ia_high | ib_at_pos | ia_low;
```

(*b*) Find the cost and delay of the `shift_left` module instances `slc` and `slb` taking into account any constant inputs and assuming that the synthesis program infers a logarithmic shifter. Don't forget that your answer must be in terms of w_a, w_b, w_o, l_a, and l_b, and that these denote the parameters of `insert_at`, not the parameters of the shifters. For more information on the logarithmic shifter see the additional material provided for the Set 1 lectures on the course lectures page.

Before cutting-and-pasting simple-model cost and delay expressions for a logarithmic shifter, take a close look at the parameters set for `slc` and `slb` and be sure to optimize for them. Notice that unlike typical shifters, the shift-out and shift-in ports are not the same size and that the shift amount is not necessarily ceiling-log-two of the input width.

Hint: The cost and delay for one of these shifters will be really easy to compute.

(*c*) Find the cost and delay of `insert_at`. Use the answers above and work out cost and delay for the remaining hardware in the module. Don't forget to use $C_{\rm lsb}(w_a)$ for the cost of the `mask_lsb` module and $D_{\rm lsb}(w_a)$ for the delay of the `mask_lsb` module.

Problem 2: Some of you may have seen this coming: Find expressions for $C_{\rm lsb}(w)$, the cost of the `mask_lsb` module and $D_{\rm lsb}(w)$, the delay of the `mask_lsb` module, in both cases `wo` $= w$, where `wo` is the parameter used in the `mask_lsb` definition. Assume a well-optimized design, not something that uses w $\lceil \lg w \rceil$-bit magnitude comparison units.

Hint: Think about the problem for about 30 minutes, then look at 2018 Final Exam Problems 1 and 2.

An uncommented Homework 1 solution appears below.
For the full version visit `https://www.ece.lsu.edu/koppel/v/2021/hw01-sol.v.html`.

```
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 [wa-1:0] mask_low;
   mask_lsb #(wa) ml(mask_low, pos);
   uwire [wa-1:0] ia_low = ia & mask_low;
   uwire [wa-1:0] ia_high_low = ia & ~mask_low;

   localparam int wblg = $clog2(wb);
   uwire [wo-1:0] ia_high;
   shift_left #(wa,wo,wblg) slc( ia_high, ia_high_low, wblg'(wb) );

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

   assign o = ia_high | ib_at_pos | ia_low;

endmodule

module shift_left
  #( int wi = 4, wo = wi, wolg = $clog2(wo) )
   ( output uwire [wo-1:0] o,
     input uwire [wi-1:0] i,
     input uwire [wolg-1:0] amt );
   assign o = i << amt;
endmodule

module mask_lsb
  #( int wo = 6, wp = $clog2(wo+1) )
   ( output logic [wo-1:0] o, input uwire [wp-1:0] n1 );
   always_comb for ( int i=0; i<wo; i++ ) o[i] = i < n1;
endmodule
```