LSU EE 4755 **Homework 3** Solution **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 $C_{\text{lsb}}(w_a)$ for the cost of the `mask_lsb` module and $D_{\text{lsb}}(w_a)$ 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;
```

Suppose for a moment that each of the quantities being ORd, `ia_high`, `ib_at_pos`, and `ia_low`, are w_o bits. Then for each of the w_o bit positions in `o` there will be a 3-input OR gate (or possibly two 2-input OR gates) and the total cost would be $2w_o\,\text{u}_\text{c}$. But while `ia_high` and `ib_at_pos` are w_o bits, `ia_low` is only w_a bits. So the cost of the hardware computing the low w_a bits of `o` will be $2w_a\,\text{u}_\text{c}$. Each of the remaining $w_o - w_a = w_b$ bits will just be an OR of a bit of `ia_high` with a bit of `ib_at_pos`, for a cost of $w_b\,\text{u}_\text{c}$. So the total cost will be $\boxed{[2w_a + w_b]\,\text{u}_\text{c}}$ or equivalently $\boxed{[w_o + w_a]\,\text{u}_\text{c}}$.

The low w_a bits are computed using either two 2-input OR gates or a 3-input OR gate, either way $\boxed{\text{the delay is } 2\,\text{u}_\text{t}}$. Note that the delay should be based on the critical path, and in this case it is one of the low w_a bits. I suppose it's nice that those other bits are computed in just $1\,\text{u}_\text{t}$ but the important number is when all bits are done.

Grading Note: Many gave the delay as $\lceil \lg 3 \rceil\,\text{u}_\text{t}$. Normally I don't expect numbers to be computed for arithmetic expressions, but that's for complex ones. In this case, please just give the answer as 2, lest I assume you don't know what $\lceil \lg 3 \rceil\,\text{u}_\text{t}$ means.

Common Mistake: A common mistake was to OR together all $2w_o + w_a$ bits in one big OR gate, or perhaps two large OR gates. That's wrong because that's not what a bitwise OR does.

(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.

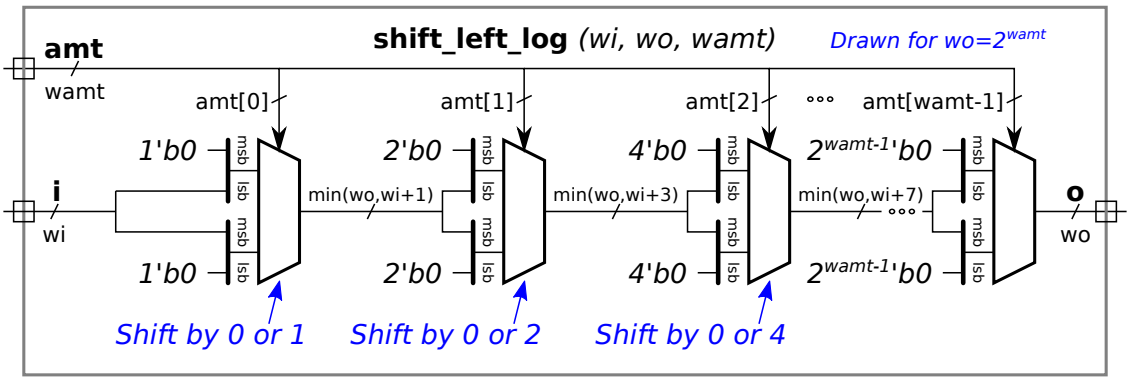
Notice that the shift amount connection (`amt`) to `slc` is an elaboration-time constant, `wb`. Therefore, the cost of `slc` is zero. A bit in the output `ia_high` is either connected to a bit of input `ia_high_low` or to the constant zero.

Grading Note: Most people did not see that the shifter required no hardware at all (other than something to generate a constant zero which would be optimized away). A few that did notice that the shift amount was zero did not properly optimize the multiplexors to which the shift amount is connected. If one of the data inputs of a mux is constant the cost drops from $3\,\mathrm{u_c}$ to $1\,\mathrm{u_c}$ per bits. But if the select input is constant the cost goes to zero. If that's not obvious please review what a mux does.

Next, consider `slb`, in which none of inputs are constant. The width of the input is w_b, the width of the output is w_o, and the input can be shifted by at most $2^{\lceil \lg(w_a+1)\rceil}$ bits. Let $l_a = \lceil \lg(w_a+1)\rceil$, that's the number of bits used to represent the shift amount. The value of the shift amount is at most $2^{l_a}-1$.

A logarithmic shifter with an l_a-bit shift amount consists of l_a multiplexors, one multiplexor for each bit in the shift amount. Multiplexor 0 shifts by either 0 or $2^0 = 1$ bit, mux 1 shifts by either 1 or $2^1 = 2$ bits, mux i shifts by either 0 or 2^i bits, and mux $l_a - 1$ shifts by 0 or 2^{l_a-1} bits. In a conventional logarithmic shifter with $l_a = 4$, the input and output would each be $2^4 = 16$ bits, and as a whole the shifter could shift by an amount ranging from 0 bits to 15 bits (but not by 16 bits). (Why not 16 bits? That's a convention, but why not allow a shift amount that would shift away all of the bits. Good question, I'm sure it was debated.)

Lets consider the shifter needed for `slb`. Let the first multiplexor making up this shifter shift by 0 or 1 bits. In a conventional shifter the mux has two w-bit inputs and a w-bit output. But in `slb` the output will be larger than the input, w_o bits. So we need to make the mux large enough to handle the largest value produced at that stage. For the first stage, since it can shift by one bit, we need to make the mux $w_b + 1$ bits (remembering that input is w_b bits). The second mux can shift by 0 or 2 bits, and to it needs to be $w_b + 1 + 2 = w_b + 3$ bits. Because the output is w_o bits the maximum mux size is w_o bits, which will be the last mux. That last mux can shift by 0 or 2^{l_a-1} bits. (Because w_a and w_b are not constrained, it is not always true that $2^{l_a-1} = w_a/2$.) The diagram below shows such a shifter in which `wi` would be used for w_b and `wamt` would be used for l_a.



A general w-bit 2-input mux has cost $3w\,\mathrm{u_c}$. But in a shifter some mux input bits are zero, and at those positions the cost is $1\,\mathrm{u_t}$ each. First lets assume that all bits have cost $w\,\mathrm{u_t}$. Also, lets restrict ourselves to the case where $w_o = w_b + 2^{l_a-1}$.

The cost under that assumption and restriction is

$$\begin{aligned} C_{\mathrm{sl-noopt}}(w_b, w_o, l_a) &= \sum_{i=0}^{l_a-1} 3\left(w_b + \sum_{j=0}^{i} 2^j\right)\,\mathrm{u_c} \\ &= \sum_{i=0}^{l_a-1} 3(w_b + 2^{i+1} - 1)\,\mathrm{u_c} \\ &= \left[3(w_b-1)l_a + \frac{3}{2}(2^{l_a}-1)\right]\,\mathrm{u_c} \end{aligned}$$

For a tighter cost estimate, consider the number of zero bits in stage i. Stage i shifts by 2^i bits and so 2^i zeros must be appended to the most-significant side of the unshifted input and 2^i zeros are appended to the least-significant side of the shifted input. So there are 2×2^i mux bits with a zero at either input, and so the cost is

$[3(w_b + 2^{i+1} - 1) - 2 \times 2 \times 2^i]\,\mathrm{u_c}$ or $[3(w_b + 2^{i+1} - 1) - 2 \times 2^{i+1}]\,\mathrm{u_c}$ or $[3(w_b - 1) + (3-2)2^{i+1}]\,\mathrm{u_c}$ or $[3(w_b - 1) + 2^{i+1}]\,\mathrm{u_c}$.

The total cost is

$$\begin{aligned} C_{\mathrm{sl-opt}}(w_b, w_o, l_a) &= \sum_{i=0}^{l_a-1} [3(w_b - 1) + 2^{i+1}]\,\mathrm{u_c} \\ &= [3(w_b - 1)l_a + \frac{1}{2}(2^{l_a} - 1)]\,\mathrm{u_c} \end{aligned}$$

Grading Note: No one computed the cost completely correctly. A small deduction, 0.5, was given for a cost of $w_o l_a\,\mathrm{u_c}$ since that overstates the cost of all but the last mux. A much larger deduction was given if the cost was based on muxen that were too small.

The delay is far less tedious to compute because regardless of the size of each multiplexor, the critical path through a mux passes through two 2-input gates. Under the simple model their delay is $2\,\mathrm{u_t}$, and so the total delay is $2l_a\,\mathrm{u_t}$. That's it.

(*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 $C_{\mathrm{lsb}}(w_a)$ for the cost of the `mask_lsb` module and $D_{\mathrm{lsb}}(w_a)$ for the delay of the `mask_lsb` module.

For this discussion refer to the `insert_at` module below which includes labels such as `Line 1` in the comments. In the sub-problems above the cost and delay of hardware described by Lines 7, 5, and 6 has been found. The cost and delay of the `ml` instance, Line 1, are given in this problem as $C_{\mathrm{lsb}}(w_a)$ and $D_{\mathrm{lsb}}(w_a)$. The Verilog on Line 4 is executed at elaboration time and so does not describe hardware. All that remains to work out is the hardware described on Lines 2 and 3.

Each of these lines is a bitwise AND of two w_a-bit quantities, for a cost of $w_a\,\mathrm{u_c}$ each. Their delay is $1\,\mathrm{u_t}$. Combining all of these yields the total cost,

$$C_{\mathrm{insertat}}(w_a, w_b) = \big[\, \overbrace{C_{\mathrm{lsb}}(w_a)}^{\texttt{ml}-\mathrm{L1}} + \overbrace{2w_a}^{\mathrm{L2\text{-}3}} + \overbrace{0}^{\mathrm{L5}} + \overbrace{3(w_b-1)l_a + \frac{1}{2}(2^{l_a}-1)}^{\texttt{slb}-\mathrm{L6}} + \overbrace{2w_a + w_b}^{\texttt{o}-\mathrm{L7}} \,\big]\,\mathrm{u_c}$$

Collecting terms and using C_{lsb} from the problem below:

$$\begin{aligned} C_{\mathrm{insertat}}(w_a, w_b) &= \left[C_{\mathrm{lsb}}(w_a) + 2w_a + 0 + 3(w_b-1)l_a + \frac{1}{2}(2^{l_a}-1) + 2w_a + w_b\right]\,\mathrm{u_c} \\ &= [w_a + 2^{l_a} - 4 + 2w_a + 0 + 3(w_b-1)l_a + \frac{1}{2}(2^{l_a}-1) + 2w_a + w_b]\,\mathrm{u_c} \\ &= [w_a + w_a - 4 + 2w_a + 0 + 3(w_b-1)l_a + \frac{1}{2}(w_a-1) + 2w_a + w_b]\,\mathrm{u_c} \\ &= [6.5w_a - 4.5 + 3(w_b-1)l_a + w_b]\,\mathrm{u_c} \\ &= [3(w_b-1)l_a + w_b + 6.5w_a - 4.5]\,\mathrm{u_c} \end{aligned}$$

The dominant term is $3w_b l_a$, which isn't so bad.

```
// SOLUTION -- Line numbers are referenced in the solution discussion.
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);              // Line 1.
   uwire [wa-1:0] ia_low = ia & mask_low;         // Line 2.
   uwire [wa-1:0] ia_high_low = ia & ~mask_low;   // Line 3.

   localparam int wblg = $clog2(wb);  // Line 4. No Hardware. (Computed during elaboration.)
   uwire [wo-1:0] ia_high;
   shift_left #(wa,wo,wblg) slc( ia_high, ia_high_low, wblg'(wb) ); // Line 5

   uwire [wo-1:0] ib_at_pos;
   shift_left #(wb,wo,walg) slb( ib_at_pos, ib, pos );              // Line 6

   assign o = ia_high | ib_at_pos | ia_low;                         // Line 7

endmodule
```

To find the total delay **we need to find the critical path**. *Note: Emphasis added after grading.* The critical path is easy to find because the parts taking a substantial amount of time, `ml` (the `mask_lsb` instance) and `slb`, connect only to `insert_at` module inputs. The default assumption for timing analysis is that module inputs arrive at $t = 0$, and so the output of `ml` is available at $D_{\text{lsb}}(w)$ and the output of `slb` is available at $2l_a$ u_t. Peeking ahead to the solution of the next problem, we know that `ml` has a delay of l_a u_t.

The output of both `ml` and `slb` each connect only to the `o` expression, Line 7, and so the critical path is from `slb` to Line 7. That would add a delay of 1 (if connected intelligently), and so the delay is $D_{\text{insertat}}(w_a, w_b) = [2l_a + 1]$ u_t, where $l_a = \lceil \lg(w_a + 1) \rceil$.

Problem 2: Some of you may have seen this coming: Find expressions for $C_{\text{lsb}}(w)$, the cost of the `mask_lsb` module and $D_{\text{lsb}}(w)$, 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.

The `gtd_rec` module from the 2018 final exam is similar to `mask_lsb` but has three differences. In `mask_lsb` the input value, `n1`, specifies that there should be `n1` ones followed by zeros. In `gtd` the input value, `iter`, specifies that there should be `iter+1` zeros followed by ones. The second difference (or a consequence of the first) is that while the output of `mask_lsb` can be all zeros or all ones, the output of `gtd_rec` must have at least one zero. Finally, `gtd_rec` can only be instantiated at power-of-two sizes.

Those minor differences are easy to fix. For example, inverting the output (change each zero to a one) will fix the first difference. The non-power-of-two issue can be fixed by making sure that the size of the recursive instantiation is always a power of two. The initial instantiation does not have to be a power of two. Also a special case can be added to the initial instantiation to handle the all ones case.

I'm tempted to show the recursive version of `mask_lsb`, but I might make it a midterm exam problem. (Not the whole thing, just a small part.) If I do I'll provide a warning in class on Monday, 25 October 2021.

For cost, the easiest thing to do is assume that w is a power of 2 and then just use the expressions from the exam. Using this assumption: $C_{\mathrm{lsb}}(w) = [2w - 4]\,\mathrm{u_c}$. For arbitrary positive w the cost of the initial instantiation is $w\,\mathrm{u_c}$ and the cost of the recursive instantiation (one level down) is $2^{\lceil \lg w \rceil - 1}\,\mathrm{u_c}$. The terminal case for recursion is for $w = 2$, and the cost of that hardware is zero under the simple model. So the summation will end at $w = 4$ (which is $i = 2$ in the summation). The total cost is

$$\begin{aligned} C_{\mathrm{lsb}}(w) &= [w + \sum_{i=l_w-1}^{2} 2^i]\,\mathrm{u_c} \\ &= [w + 2^{l_w} - 4]\,\mathrm{u_c} \end{aligned}$$

where $l_w = \lceil \lg w \rceil$.

Each level has a delay of 1, and so the total delay is $[\lceil \lg w \rceil - 1]\,\mathrm{u_t}$ for $w \geq 4$.

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
```