LSU EE 4755 **Homework 3** Solution **Due: 23 October 2019**

Problem 1: Appearing below is a module excerpted from the solution to Homework 1. Compute the cost and delay of this module using the simple model under the following assumptions:

- The inputs arrive at $t = 0$. Don't assume that any bit is early or late, they all arrive at exactly $t = 0$.
- A ripple adder will be used to implement addition.
- Apply obvious optimizations. In particular, don't use a BFA if a BHA would suffice. And only use a BHA if that is needed.
- Don't overlook the fact that one of the shifter inputs is a constant.

Show the cost and delay in terms of `wa` and `wb`, but use symbol a for `wa` and b for `wb`. For example, "The cost is $(a+b)9\,\mathrm{u_c}$ and the delay is $(a+b)2\,\mathrm{u_t}$." (Those answers assume that BFAs are used for the entire module, which is wrong.)

The simple model slides (AOTW) don't show the cost and delay of a BHA, so work that out yourselves.

```
module mult_piece
  #( int wa = 16, int wb = 16, int wp = wa + wb,
     int wn = wa / 2, int wx = wb + wn )
   ( output uwire [wp:1] prod,
     input uwire [wx:1] prod_lo, prod_hi );

      assign prod = prod_lo + ( prod_hi << wn );

endmodule
```

Short answer: Cost: $[9b + 3\frac{a}{2}]\,\mathrm{u_c}$, Delay: $[2 + 2b + \frac{a}{2}]\,\mathrm{u_t}$.

Long answer: Because `prod_hi` is shifted and because `prod_lo` and `prod_hi` are the same width the adder can be broken into three regions: an $a/2$-bit low region consisting of the low $a/2$ bits of `prod_lo`, a b-bit middle region consisting of the high b bits of `prod_lo` and the low b bits of `prod_hi`, and an $a/2$-bit region consisting of the high $a/2$ bits of `prod_hi`.

There is no hardware at all for the low region. The middle region consists of b binary full adders, and the high region consists of $a/2$ binary half adders. (The high region has to handle the carry out from the middle region.)

Under the simple model a BFA cost $9\,\mathrm{u_c}$ and in a w-bit ripple configuration has a delay of $[2 + 2w]\,\mathrm{u_t}$. A BHA can be derived from a BFA by setting the `a` or `b` input to logic zero and then simplifying. Such a BHA would have a cost of $3\,\mathrm{u_c}$ per bit and a delay of $1\,\mathrm{u_t}$ per bit in a ripple configuration.

The total cost is then $[9b + 3\frac{a}{2}]\,\mathrm{u_c}$ and the delay is $[2 + 2b + \frac{a}{2}]\,\mathrm{u_t}$.

There's another problem on the next page!

Problem 2: A w-bit multiplier needs to add together w partial products using $w-1$ adders. A naïve timing analysis of a non-tree ripple adder implementation would compute a delay of $w(2 \times 2w+2) = (4w^2+w)\,\mathrm{u_t}$ for the $2w$-bit product using the simple model and ignoring ripple-unit cascading. As we should know $4w^2$ is not a good term to have in an expression for time. The goal of this problem is to see how the tree multiplier compares to this naïve timing analysis.

Appearing below is the Bonus Solution to Homework 1 in which a single `mult_tree` module is used rather than separate modules `mult16_tree`, `mult8_tree`, etc. Also shown is a module, `my_module`, that instantiates the `mult_tree`. Also shown a page or two ahead is the diagram from Homework 1. You may want to use this to help work out the solution to this problem.

Analyze the cost and performance of `my_module` as described below. When computing the cost and performance don't forget to account for the full elaboration, not just the top level. For example, `my_module` with `w=4` consists of one `mult_tree` at `w=4` and two `mult_tree` modules at `w=2`, and four `mult_tree` modules at `w=1`.

```
module mult_tree
  #( int wa = 16, int wb = 16, int wp = wa + wb )
   ( output logic [wp:1] prod,
     input uwire [wa:1] a,
     input uwire [wb:1] b );

   if ( wa == 1 ) begin

      assign prod = a ? b : 0;
      // Equivalent to: prod = a * b;

   end else begin

      // Split a in half and recursively instantiate a module for each half.
      localparam int wn = wa / 2;
      localparam int wx = wb + wn;

      uwire [wx:1] prod_lo, prod_hi;

      mult_tree #(wn,wb) mlo( prod_lo, a[wn:1],    b );
      mult_tree #(wn,wb) mhi( prod_hi, a[wa:wn+1], b );

      // Combine the partial products.
      always_comb prod = prod_lo + ( prod_hi << wn );

   end
endmodule

module my_module
  #( int w = 8, int wp = 2 * w )
   ( output uwire [wp-1:0] p,
     input uwire [w-1:0] x, y );
   mult_tree #(w,w) mt1(p,x,y);
endmodule
```

(*a*) Compute the cost of `my_module` using the same assumptions as in Problem 1. The cost must

be in terms of w. It's okay, indeed encouraged, to use sample values like $w = 16$ when working out the problem, but once you have it figured out give the answer in terms of w. (If you have not solved Problem 1 then use the incorrect sample answers provided in Problem 1.)

The following identity may be helpful: $\sum_{i=0}^{m-1} 2^i = 2^m - 1$. In such a summation i might indicate the level of recursion and 2^i might indicate the number of modules at that recursion level. For the top level of the recursion $i = 0$.

Let j denote the recursion level such that $a = 2^j$, and note that j starts at $\lg w$ with the initial instantiation of `mult-tree` (the one made by `my-module`) and ends at $j = 0$, the terminal case. At level j there are a total of $w/2^j$ instances. For the terminal case, $j = 0$ and $a = 1$, `mult-tree` produces just a mux, which itself will be optimized to b AND gates. There will be $w/2^0$ instances for $j = 0$, so their total cost will be $w^2\,\mathrm{u_c}$, after setting $b = w$.

The $j > 0$ levels consist of binary full and half adders. Each instance has about w BFAs and $a/2 = 2^{j-1}$ BHAs. Let c_f denote the per-bit cost of a BFA and c_h denote the per-bit cost of a BHA. By the simple model $c_f = 9\,\mathrm{u_c}$ and $c_h = 3\,\mathrm{u_t}$. (In the BHA the carry out can be used to compute the sum, reducing the number of additional gates for the XOR to 2.) Then the total cost of the adders is

$$\begin{aligned}
&\sum_{j=1}^{\lg w} \frac{w}{2^j}\left(wc_f + 2^{j-1}c_h\right) \\
=&w\sum_{j=1}^{\lg w}\left(\frac{wc_f}{2^j} + \frac{c_h}{2}\right) \\
=&w^2\sum_{j=1}^{\lg w}\frac{c_f}{2^j} + \frac{w}{2}\sum_{j=1}^{\lg w} c_h \\
=&w^2\left(1 - \frac{1}{w}\right)c_f + \frac{w}{2}(\lg w)c_h
\end{aligned}$$

(Note that $\sum_{j=1}^{\lg w} 2^{-j} = (1 - 1/w)$.) The total overall cost is

$$\begin{aligned}
&w^2\,\mathrm{u_c} + w^2\left(1 - \frac{1}{w}\right)c_f + \frac{w}{2}(\lg w)c_h \\
=&\left[10w^2 - 9w + \frac{3w}{2}(\lg w)\right]\,\mathrm{u_c}
\end{aligned}$$

An important point to note is that the cost is proportional to w^2. That should not be surprising because we know that to multiply two w-bit quantities we need $w - 1$ adders, each costing about wc_f.

(b) Compute the delay of the multiplier using a simplifying assumption similar to the one used in Problem 1: when computing the delay of `prod = prod_lo + ( prod_hi << wn )` assume that all bits for `prod_lo` and `prod_hi` arrive at the same time and that all bits of `prod` are sent to the outputs at the same time. (Don't like simplifying assumptions? The next subproblem is for you!)

Show your answer for `w=8` and as an expression in terms of w. Don't forget to consider the entire elaboration, not just the top-level module.

The launch point starts at $j = 0$ (the terminal case), which depends only on the inputs to `my-module`, `a` and `b`. The delay is just $1\,\mathrm{u_t}$.

Level $j > 0$ has an adder consisting of w BFAs and 2^{j-1} BHAs. The total delay through that is $[2(w-1) + 2^{j-1}]\,\mathrm{u_t}$, where the delay through a 2^{j-1}-bit BHA is $2^{j-1}\,\mathrm{u_t}$. The total delay including the AND gates is

$$\begin{aligned}
&\left[1 + \sum_{j=1}^{\lg w} \left(2(w-1) + 2^{j-1}\right)\right] \mathrm{u_t} \\
&= \left[1 + 2(w-1)(\lg w) + w - 1\right] \mathrm{u_t} \\
&= \left[w + 2(w-1)(\lg w)\right] \mathrm{u_t} \\
&\approx 2w \lg w \, \mathrm{u_t}
\end{aligned}$$

The dominant term is $2w \lg w$ which is not as bad as a linear connection of adders which would have a delay of $\approx 2w^2 \, \mathrm{u_t}$ under similar assumptions.

(*c*) Compute the delay of the multiplier without the simplifying assumption. That is, account for the fact that the less-significant bits of `mult_tree` will be ready before the more-significant bits.

Show your answer for `w=8` and as an expression in terms of w. Don't forget to consider the entire elaboration, not just the top-level module.

At level j the least significant BFA (which could actually be a BHA, but we'll keep it simple) is connected to bit 2^{j-1} of `prod-lo` and bit zero of `prod-hi`. Since level j is waiting for 2^{j-1} to be ready, the next level, $j+1$, must be waiting for bit 2^j. Therefore for level j we need to compute the delay though $2^{j-1}+1$ BFAs: stating at the least significant BFA (bit position 2^{j-1}) and ending at the BFA computing bit 2^j. The delay for w bits for a ripple adder under the simple model is $2(w+1) \, \mathrm{u_t}$, so the delay at level j before $j+1$ can start is $2((2^{j-1}+1)+1) = 2^j + 4$.

For level $j = 0$ the delay is $1 \, \mathrm{u_t}$ (an AND gate). For level $j = \lg w$ we need to add on the remaining bits in the ripple adder: $w/2 - 1$ BFA delays and $w/2$ BHA delays.

The total delay is:

$$\begin{aligned}
&\left[1 + \left(\sum_{j=1}^{\lg w} 2^j + 4\right) + \left(\frac{w}{2} - 1\right) 2 + \frac{w}{2}\right] \mathrm{u_t} \\
&= \left[1 + 2(2w-1) + 4(\lg w) + \left(\frac{w}{2} - 1\right) 2 + \frac{w}{2}\right] \mathrm{u_t} \\
&= \left[5.5w + 4(\lg w) - 1\right] \mathrm{u_t}
\end{aligned}$$

Useful diagram on next page.

Use the diagram below to help work out solutions.