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 $(a + b)9\,u_c$ and the delay is $(a + b)2\,u_t$." (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
```

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.

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 $i = 0$.

(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.

(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.

Useful diagram on next page.

Use the diagram below to help work out solutions.