Problem 1: Appearing below is the output of the simulator and synthesis script, showing data for the Homework 7 solution modules. Modules are simulated and synthesized for $w = 32$.

```
Module Name                              Area  Period  Period
                                               Target  Actual
mult_seq_ds_prob_1_w32_m1              157813    1000   14926
mult_seq_ds_prob_1_w32_m2              185493    1000   15431
mult_seq_ds_prob_1_w32_m4              242568    1000   16296

mult_seq_d_prob_2_w32_m1               288580    1000   31944
mult_seq_d_prob_2_w32_m2               301203    1000   32204
mult_seq_d_prob_2_w32_m4               329226    1000   32192

For Prob 1 Deg 1       ran  400 tests,    0/   0/   0 errors found. Avg cyc 33.0
For Prob 1 Deg 2       ran  400 tests,    0/   0/   0 errors found. Avg cyc 17.0
For Prob 1 Deg 4       ran  400 tests,    0/   0/   0 errors found. Avg cyc 9.0

For Prob 2 Deg 1       ran  400 tests,    0/   0/   0 errors found. Avg cyc 9.5
For Prob 2 Deg 2       ran  400 tests,    0/   0/   0 errors found. Avg cyc 7.3
For Prob 2 Deg 4       ran  400 tests,    0/   0/   0 errors found. Avg cyc 5.0
Modules instantiated with w = 32.
```

The Problem 1 modules are based on the streamlined multiplier and so are faster. But the Problem 2 modules skip zeros. Based on the data above, indicate the ways, if any, that the Problem 2 modules are better than the Problem 1 modules. Explain using the numbers above.

By skipping zeros the Problem 2 modules should compute a result with lower latency (in less time) than the Problem 1 modules, which require $\lceil w/m \rceil + 1$ cycles regardless of the numbers being multiplied. The latency for a multiplication is the product of the clock period and the average number of cycles required. For the Problem 1 modules that works out to

$$33 \times 14.926\,\mathrm{ns} = 492.6\,\mathrm{ns}, \quad 17 \times 15.431\,\mathrm{ns} = 262.3\,\mathrm{ns}, \text{ and } \quad 9 \times 16.296\,\mathrm{ns} = 146.7\,\mathrm{ns}$$

for the degree (m) 1, 2, and 4 modules respectively. Though the clock periods for the Problem 2 modules are larger, fewer cycles are needed to produce an answer according to the data collected by the testbench. (See the number to the right of `Avg cyc`.) The Problem 2 module latencies are

$$9.5 \times 31.944\,\mathrm{ns} = 303.5\,\mathrm{ns}, \quad 7.3 \times 32.204\,\mathrm{ns} = 235.1\,\mathrm{ns}, \text{ and } \quad 5.0 \times 32.192\,\mathrm{ns} = 161.0\,\mathrm{ns}.$$

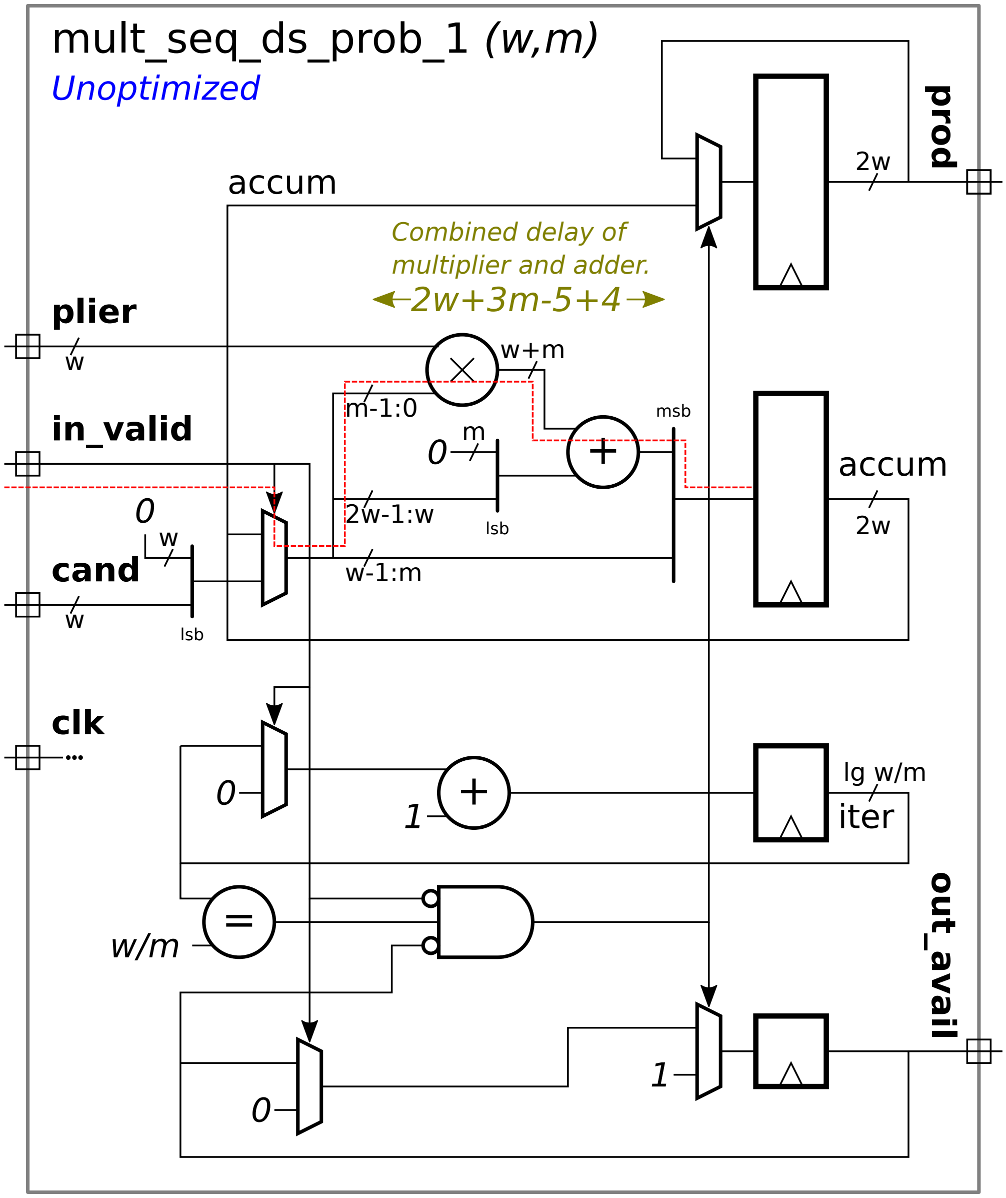
In all but the $m = 4$ case the Problem 2 module has a lower latency than the respective Problem 1 module.

There are more problems on the next pages.

Problem 2: Appearing below is a solution to Homework 7, Problem 1, the streamlined degree-m multiplier with handshaking. The complete solution is at https://www.ece.lsu.edu/koppel/v/2018/hw07-sol.v.html. For this problem assume that w and m are both powers of 2.

```
module mult_seq_ds_prob_1 #( int w = 16, int m = 2 )
   ( output logic [2*w-1:0] prod,  output logic  out_avail,
     input uwire clk, in_valid,    input uwire [w-1:0] plier, cand );

   localparam int iterations = ( w + m - 1 ) / m;
   localparam int iter_lg = $clog2(iterations);

   uwire [iterations-1:0][m-1:0] cand_2d = cand;

   bit [iter_lg:0] iter;
   logic [2*w-1:0] accum;

   always_ff @( posedge clk ) begin

      if ( in_valid ) begin

         accum = cand;
         iter = 0;
         out_avail = 0;

      end else if ( !out_avail && iter == iterations ) begin

         out_avail = 1;
         prod = accum;

      end

      accum = { 0 + plier * accum[m-1:0] + accum[2*w-1:w], accum[w-1:m] };
      iter++;
   end

endmodule
```

(*a*) Show the hardware that will be inferred for this module. The Inkscape SVG format diagram of the hardware for the streamlined sequential module from the class demo notes can be used as a starting point. It is at `https://www.ece.lsu.edu/koppel/v/2018/ill-mul-seq-str.svg`.



Solution appears above with the critical path shown in red. The hardware is un-optimized. Optimization opportunities include the logic for computing `out_avail`.

(*b*) Compute the cost and delays for this module using the simple model. Show these in terms of w and m. Clearly show the critical path on your diagram.

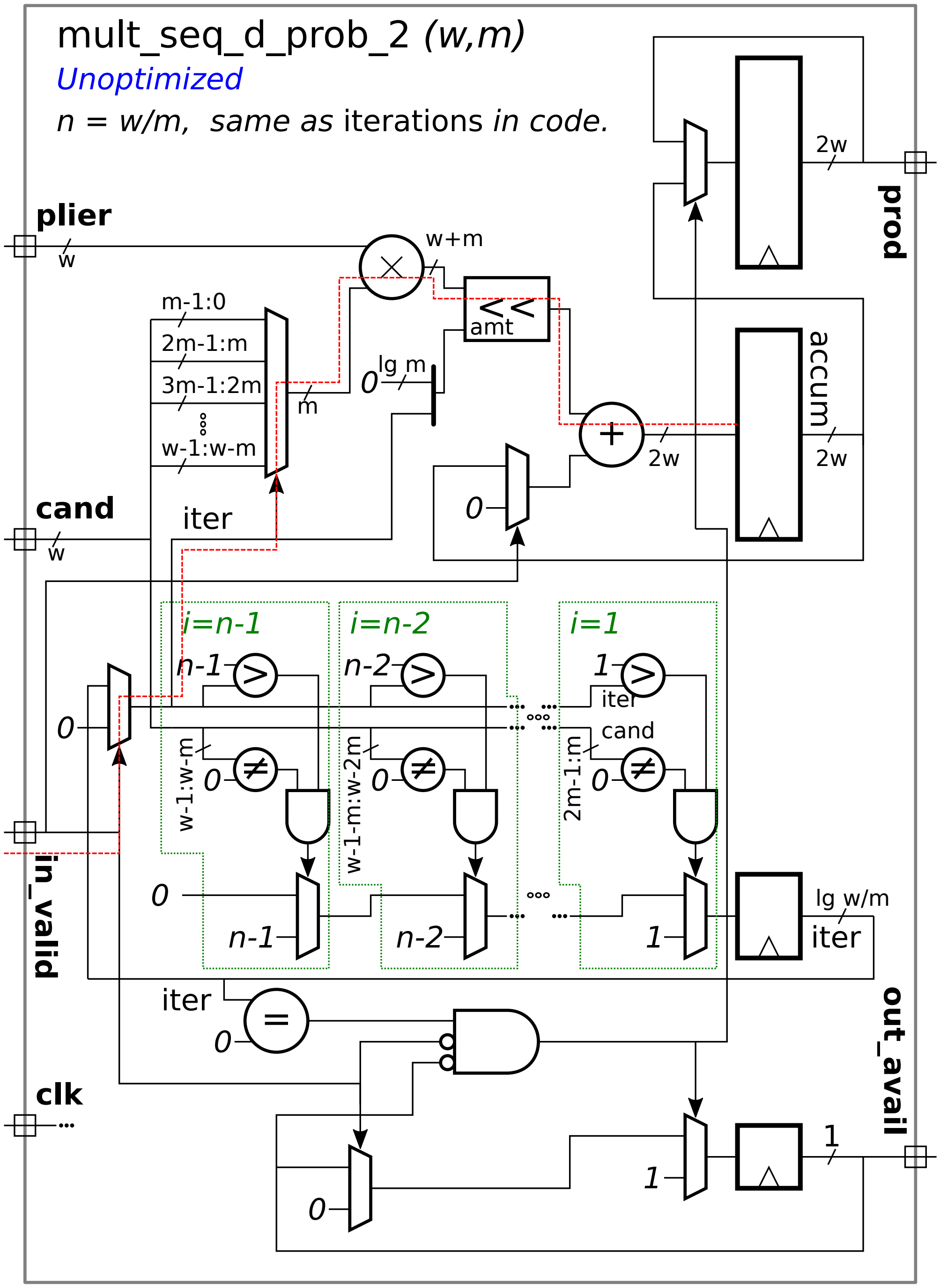
See the solution to Problem 3 for a complete delay and timing analysis. In this (Problem 2) module the cost of the adder is less because it is $w + m$ bits, rather than $2w$ bits for the Problem 3 adder. Also, this module does not use a shifter or a mux to extract the multiplicand bits.

There is a problem on the next page.

Problem 3: Appearing below is a solution to Homework 7, Problem 2, the streamlined degree-m multiplier with handshaking. The complete solution is at https://www.ece.lsu.edu/koppel/v/2018/hw07-sol.v.html. For this problem assume that w and m are both powers of 2.

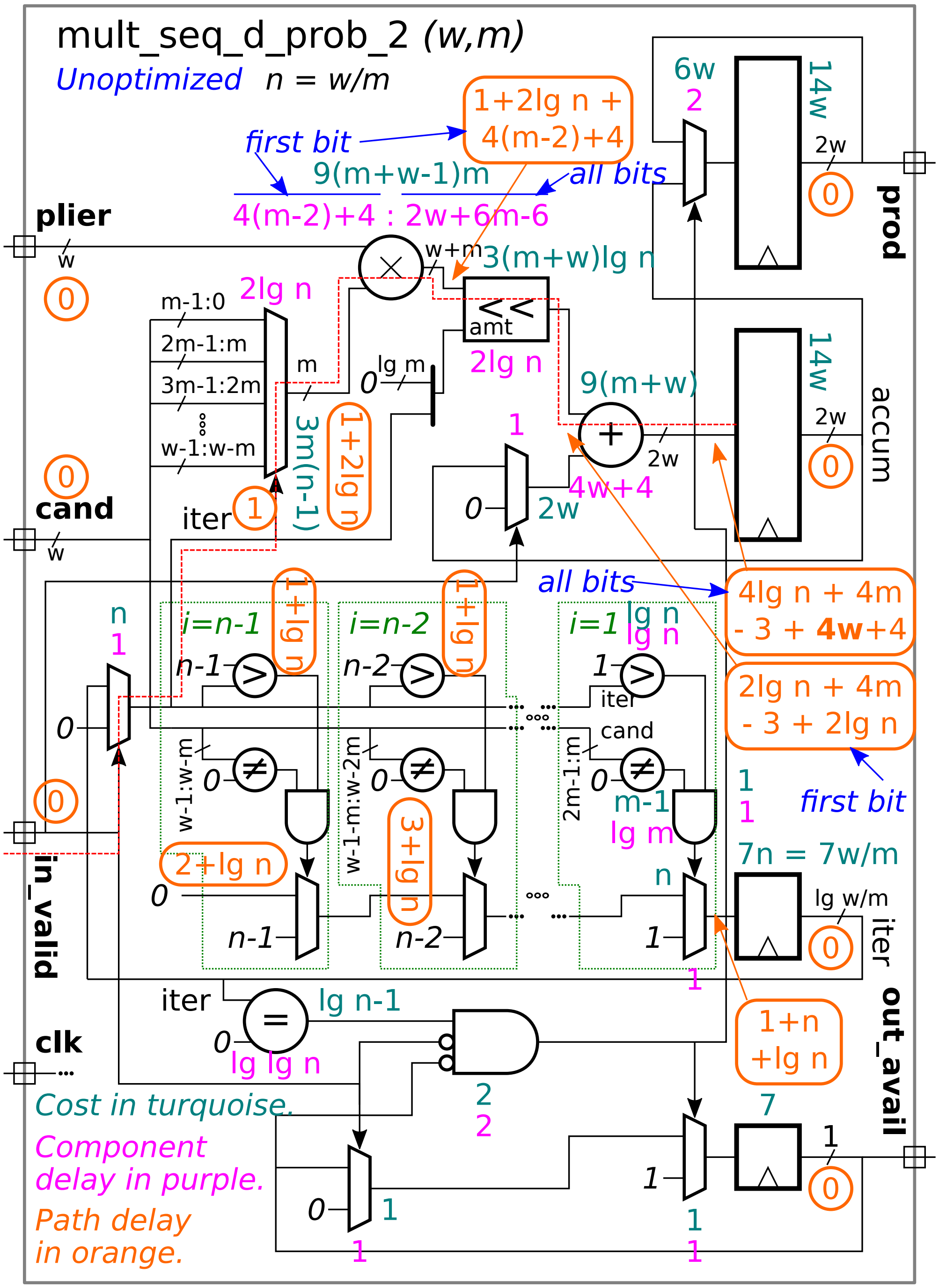
```
module mult_seq_d_prob_2 #( int w = 16, int m = 2 )
   ( output logic [2*w-1:0] prod,   output logic out_avail,
     input uwire clk, in_valid,     input uwire [w-1:0] plier, cand );

   localparam int iterations = ( w + m - 1 ) / m;
   localparam int iter_lg = $clog2(iterations);

   uwire [iterations-1:0][m-1:0] cand_2d = cand;

   bit [iter_lg-1:0] iter;
   logic [2*w-1:0] accum;

   always_ff @( posedge clk ) begin

      logic [iter_lg-1:0] next_iter;

      if ( in_valid ) begin
         iter = 0;
         accum = 0;
         out_avail = 0;
      end else if ( !out_avail && iter == 0 ) begin
         prod = accum;
         out_avail = 1;
      end

      accum += plier * cand_2d[iter] << ( iter * m );

      next_iter = 0;
      for ( int i=iterations-1;  i>0;  i-- )
        if ( i>iter && cand_2d[i] ) next_iter = i;
      iter = next_iter;
   end

endmodule
```

(*a*) Show the hardware that will be inferred for this module.



Hardware shown above with the critical path shown in red.

(b) Compute the cost and delays for this module using the simple model. Show these in terms of w and m. Clearly show the critical path on your diagram.



The costs and delay of each component are shown in the diagram above. The path delay for selected paths is shown in the circled orange numbers. Note that one input to all of the comparison units (for example, the zero in $\neq 0$), is a constant, reducing their costs and delays. Many of the multiplexors also have one constant data input.

The interesting thing to compare is the time needed to compute the updated `accum` value versus the time needed to find the next non-zero digit. The $i >$ iter comparison, because i is a constant, takes time $\lg w/m\,\mathrm{u_t} = \lg n\,\mathrm{u_t}$ and the $\neq 0$ takes less, especially if $w/m > m$. The mux delay is $1\,\mathrm{u_t}$ because one data input is a constant. The time to generate the new `iter` signal is $(1 + n + \lg n)\,\mathrm{u_t}$.

The updated `accum` value consumes most of the time. Inputs arrive at the multiplier at time $1 + 2\lg n$. For an unoptimized m-bit by $w + m$-bit multiplier, the least significant bit takes $(4(m-2)+4)\,\mathrm{u_t}$ to compute. Since the shifter can shift by n possible amounts its delay is $2\lg n$. The least significant bit arrives at the adder at time $1 + 2\lg n + 4(m-2) + 4 + 2\lg n = (4\lg n + 4m - 3)\,\mathrm{u_t}$ (see the diagram). The adder requires $(4w+4)\,\mathrm{u_t}$ to finish and so the adder output is ready at time $(4\lg n + 4m - 3 + 4w + 4)\,\mathrm{u_t}$.

The clock period would include six more cycles for the latch setup time.