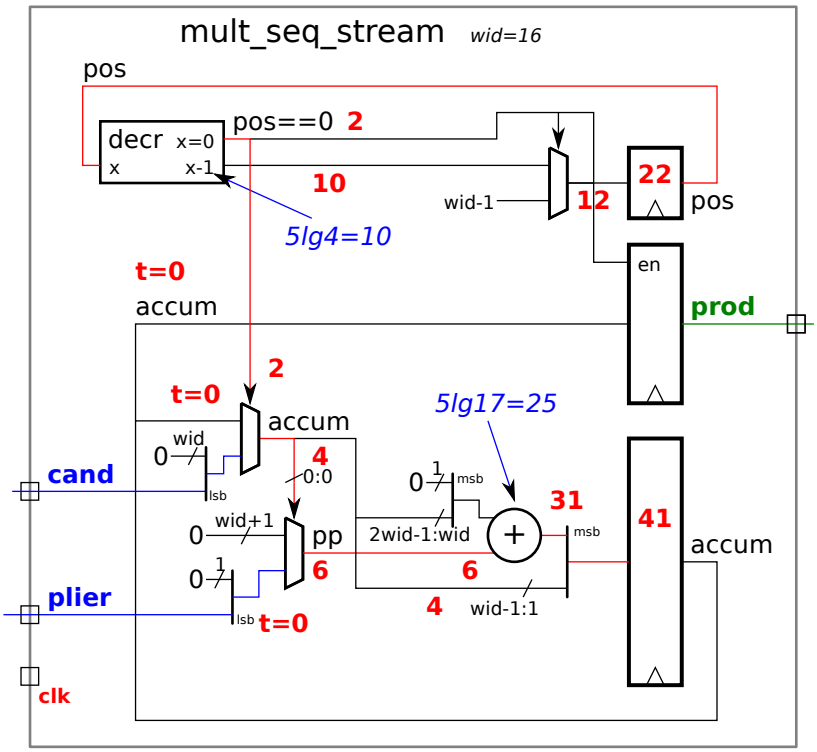
**LSU EE 4755** **Homework 5** Solution **Due: 23 Oct 2015 17:00**

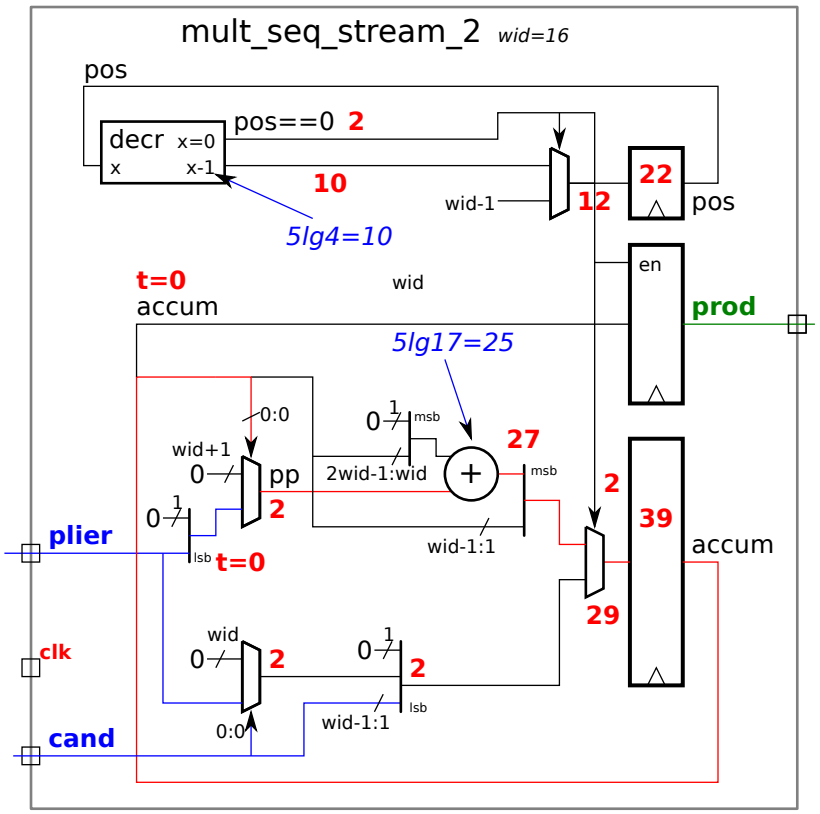
**Problem 1:** The homework Verilog file, `hw05.v`, contains something similar to the streamlined multiplier presented in class, `mult_seq_stream`, and even more streamlined versions of the multiplier, `mult_seq_stream_2`, and `mult_seq_stream_3`. These modules are reproduced at the end of this assignment. For an HTML version visit `http://www.ece.lsu.edu/koppel/v/2015/hw05.v.html`. *See the 2014 midterm exam for similar problems.*

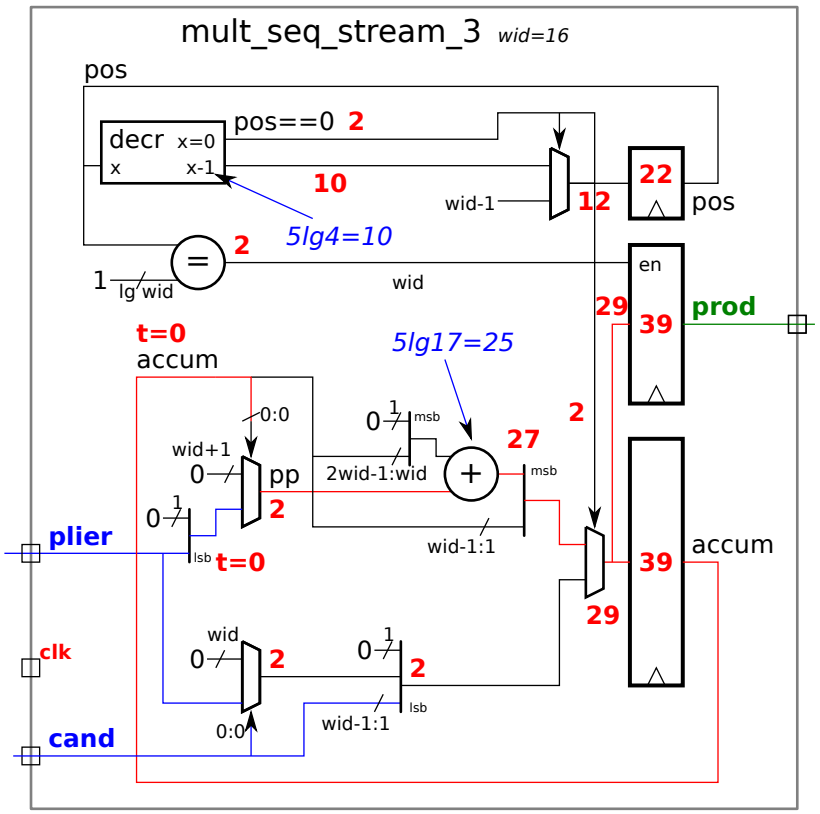
(*a*) Show the hardware that will be synthesized for each module for the default parameters. Show the module after optimization.

The synthesized hardware for each module appears below, and they also appear next to the respective Verilog descriptions at the end of this assignment. The red numbers show signal arrival times based on the assumptions given in the sub-problem below. The red wires show the critical path based on this analysis.

A `decr` unit has been used compute both `pos-1` and `pos==0`, under the assumption that it might be possible to share some hardware. An enable signal is used on the `prod` register.




mult_seq_stream_2 wid=16
pos
decr x=0
x x-1
pos==0 2
10
5lg4=10
wid-1
12
22
pos
en
prod
t=0
accum
wid
5lg17=25
0:0
wid+1
0
pp
2
1
0
lsb
t=0
plier
clk
cand
0
1
msb
2wid-1:wid
27
msb
wid-1:1
2
39
accum
29
wid
0
2
0:0
0
1
2
lsb
wid-1:1


mult_seq_stream_3 wid=16
pos
decr x=0
x x-1
pos==0 2
10
5lg4=10
wid-1
12
22
pos
=
2
1
lg wid
wid
en
29
39
prod
t=0
accum
5lg17=25
2
0:0
wid+1
0
pp
2
1
0
lsb
t=0
plier
clk
cand
0
1
msb
2wid-1:wid
27
msb
wid-1:1
39
accum
29
wid
0
2
0:0
0
1
2
lsb
wid-1:1

(*b*) Estimate the clock frequency of each module based on the following latencies:

Latch delay: 10 units. Multiplexor latency: 2 units. Latency of an $n$-bit adder: $5\lceil \lg n \rceil$ units. Latency of an $n$-input gate: $\lceil \lg n \rceil$ units. Let a unit be equal to 10 ps. *Note: The duration of a unit was not given in the original assignment.*

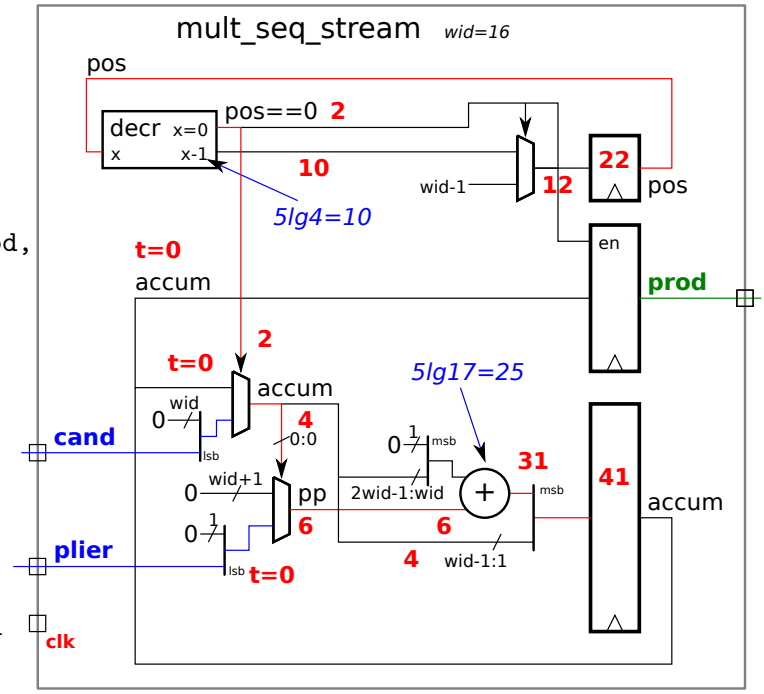
The timing analysis is shown in red on the three modules and the wires carrying the critical path are shown in red. This timing analysis strictly follows the guidelines above, using a $5\lceil \lg 17 \rceil = 25$ unit delay for the big adder. Realistically, that would be a 16-bit adder with a carry out. Solutions that used 20 rather than 25 units for the adder are correct.

For `mult_seq_stream` the critical path ends at `accum` with a period of 41 units or 410 ps. That would give a clock frequency of $\frac{1}{41}$ cycles per unit or 2.44 GHz.

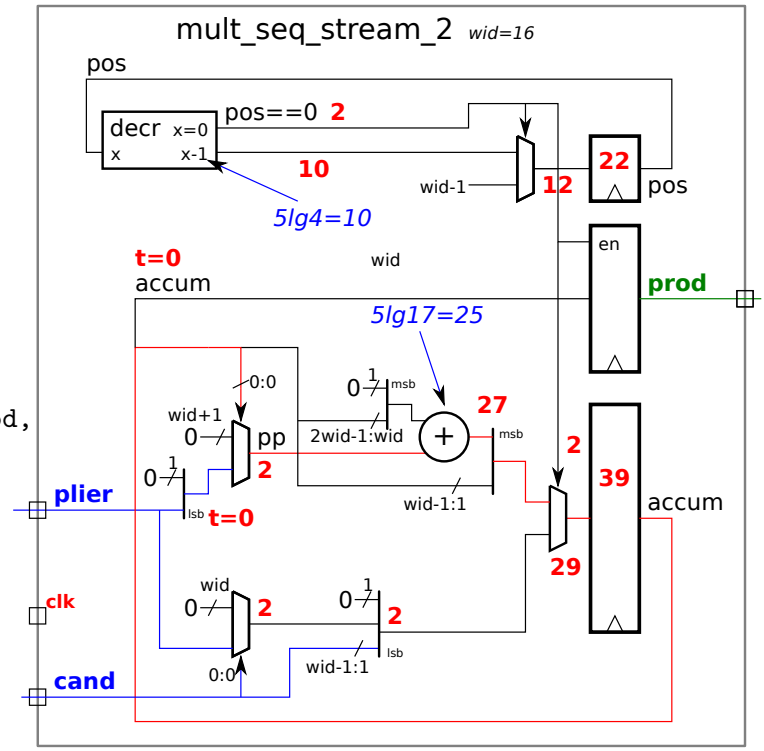
For `mult_seq_stream_2` and `mult_seq_stream_3` the critical path is 2 units shorter, at 39 units. This is because the big adder uses the `accum` signal right out of the register outputs, in contrast to `mult_seq_stream` in which the particular `accum` to use must be routed through a mux based on a `pos==0` select, adding delay. The clock frequency for these two modules would be 2.56 GHz.

(*c*) Why would module `mult_seq_stream_3` provide a result in less time than the other two, even assuming that the clock frequency for all the modules was the same?

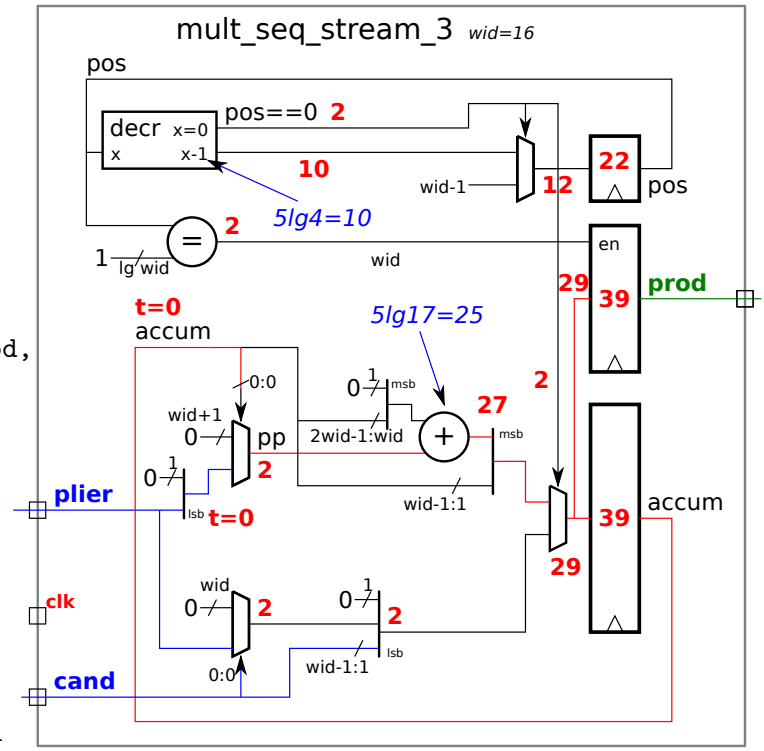
The product is available one cycle earlier because it is written to `prod` from the output of the big adder rather than from `accum`.




```
module mult_seq_stream
 #( int wid = 16 )
  ( output logic [2*wid-1:0] prod,
    input logic [wid-1:0] plier,
    input logic [wid-1:0] cand,
    input clk);

   localparam int wlog =
     $clog2(wid);

   logic [wlog-1:0] pos;
   logic [2*wid-1:0] accum;

   always @( posedge clk ) begin

      logic [wid:0] pp;

      if ( pos == 0 ) begin

         prod = accum;
         accum = cand;
         pos = wid - 1;

      end else begin

         pos--;

      end

      // Note: the multiplicand is in the lower bits of the accumulator.
      //
      pp = accum[0] ? { 1'b0, plier } : 0;

      // Add on the partial product and shift the accumulator.
      //
      accum = { { 1'b0, accum[2*wid-1:wid] } + pp, accum[wid-1:1] };

   end

endmodule
```




```
module mult_seq_stream_2
 #( int wid = 16 )
  ( output logic [2*wid-1:0] prod,
    input logic [wid-1:0] plier,
    input logic [wid-1:0] cand,
    input clk);

   localparam int wlog =
     $clog2(wid);
   logic [wlog-1:0] pos;
   logic [2*wid-1:0] accum;

   always @( posedge clk ) begin

      if ( pos == 0 ) begin

         prod = accum;
         accum = { 1'b0, cand[0] ? plier : wid'(0), cand[wid-1:1] };
         pos = wid - 1;

      end else begin

         logic [wid:0] pp;

         // Note: the multiplicand is in the lower bits of the accumulator.
         //
         pp = accum[0] ? plier : 0;

         // Add on the partial product and shift the accumulator.
         //
         accum = { { 1'b0, accum[2*wid-1:wid] } + pp, accum[wid-1:1] };

         pos--;

      end

   end

endmodule
```




```
module mult_seq_stream_3
 #( int wid = 16 )
  ( output logic [2*wid-1:0] prod,
    input logic [wid-1:0] plier,
    input logic [wid-1:0] cand,
    input clk);

   localparam int wlog =
     $clog2(wid);

   logic [wlog-1:0] pos;
   logic [2*wid-1:0] accum;

   always @( posedge clk ) begin

      if ( pos == 0 ) begin

         accum = { 1'b0, cand[0] ? plier : wid'(0), cand[wid-1:1] };
         pos = wid - 1;

      end else begin

         logic [wid:0] pp;

         // Note: the multiplicand is in the lower bits of the accumulator.
         //
         pp = accum[0] ? plier : 0;

         // Add on the partial product and shift the accumulator.
         //
         accum = { { 1'b0, accum[2*wid-1:wid] } + pp, accum[wid-1:1] };

         if ( pos == 1 ) prod = accum;

         pos--;

      end

   end

endmodule
```