Name ______________________ *Formatted For 2-Sided Printing*

Staple This Side

Digital Design Using HDLs
LSU EE 4755
Midterm Examination

Wednesday, 23 October 2024, 11:30-12:20 CDT

Staple This Side

Problem 1	______	(22 pts)
Problem 2	______	(18 pts)
Problem 3	______	(20 pts)
Problem 4	______	(10 pts)
Problem 5	______	(30 pts)
Exam Total	______	(100 pts)

Alias ______________________

Good Luck!

Problem 1: [22 pts] Below is the Homework 3 Problem 1 solution with some object names shortened.

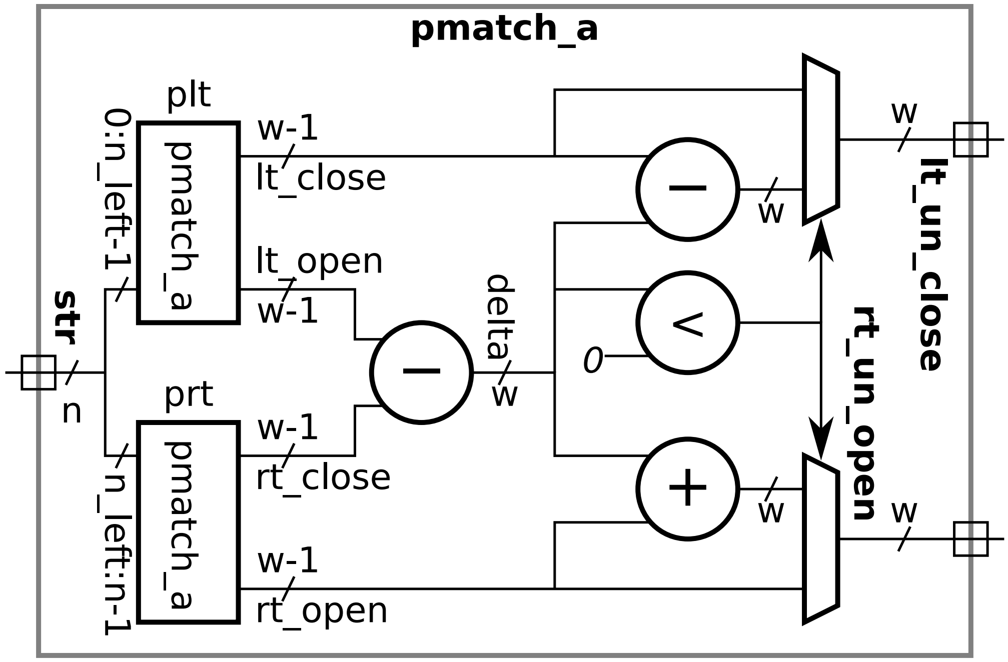
```
typedef enum logic [3:0] {Char_Blank=0, Char_Dot=1, Char_Open=2, Char_Close=3} Char;
module pmatch_a #( int n = 5, wn = $clog2(n+1) )
   ( output logic [wn-1:0] lt_un_close, rt_un_open,   input uwire [3:0] str[0:n-1] );
   if ( n == 1 ) begin
      assign lt_un_close = str[0] == Char_Close ? 1 : 0;
      assign rt_un_open =  str[0] == Char_Open  ? 1 : 0;
   end else begin
      localparam int n_left = n/2;
      localparam int n_right = n - n_left;
      localparam int wl = $clog2(n_left+1),  wr = $clog2(n_right+1);
      uwire [wl-1:0] lt_close, lt_open;
      uwire [wr-1:0] rt_close, rt_open;
      pmatch_a #(n_left,  wl) plt( lt_close, lt_open, str[0:n_left-1] );
      pmatch_a #(n_right, wr) prt( rt_close, rt_open, str[n_left:n-1] );

      uwire logic signed [wn-1:0] delta = lt_open - rt_close;
      assign lt_un_close = delta <  0 ? lt_close - delta : lt_close;
      assign rt_un_open =  delta >= 0 ? rt_open  + delta : rt_open;
   end
endmodule
```

(*a*) Show the hardware that will be inferred for the base case. Show hardware after optimization taking into account constants.

☐ Show inferred hardware for base (`n==1`) case of the module above. ☐ Show input and output ports.

☐ Optimize taking into account ☐ constant values of all kinds. ☐ Don't miss the `Char` definition above the module. ☐ Don't show a comparison unit such as $\boxed{\texttt{==}}$, instead show the gates from which it was made and ☐ optimize them, ☐ taking into account the number of bits on each output port.



Appearing above is hardware that will be inferred for the non-base case.

(*b*) Compute the cost of the hardware at this level (ignore what's inside `plt` and `prt`) based on the simple model using the bit widths from the diagram, such as `w-1`.

☐ Show the cost of each component *except for hardware inside of* `plt` *and* `prt`.

☐ Be sure to show the cost of the optimized comparison unit!

(*c*) Compute the delay through the module **starting from** launch points `lt_close`, `lt_open`, `rt_close`, and `rt_open`. The capture points are `lt_un_close` and `rt_un_open`. Use the bit widths from the diagram, such as `w-1`.

☐ Show the arrival time at each wire from launch to capture.

☐ Take into account cascaded ripple units and ☐ and the optimized comparison unit.

Problem 2: [18 pts] Appearing below is an alternative solution to Homework 3 Problem 1. The only difference is the last few lines.

```
module pmatch_b #( int n = 5, wn = $clog2(n+1) )
   ( output logic [wn-1:0] lt_un_close, rt_un_open,
     input uwire [3:0] str[0:n-1] );
   if ( n == 1 ) begin
      assign lt_un_close = str[0] == Char_Close ? 1 : 0;
      assign rt_un_open = str[0] == Char_Open ? 1 : 0;
   end else begin
      localparam int n_left = n/2;
      localparam int n_right = n - n_left;
      localparam int wl = $clog2(n_left+1),  wr = $clog2(n_right+1);
      uwire [wl-1:0] lt_close, lt_open;
      uwire [wr-1:0] rt_close, rt_open;
      pmatch_b #(n_left,  wl) plt( lt_close, lt_open, str[0:n_left-1] );
      pmatch_b #(n_right, wr) prt( rt_close, rt_open, str[n_left:n-1] );
      uwire logic signed [wn-1:0] delta = lt_open - rt_close;
      // Lines above are identical to pmatch_a.

      uwire [wn-1:0] delta_n = delta <  0 ? delta : 0;
      uwire [wn-1:0] delta_p = delta >= 0 ? delta : 0;
      assign lt_un_close = lt_close - delta_n;
      assign rt_un_open  = rt_open  + delta_p;
   end
endmodule
```

(*a*) Show the hardware that will be inferred for `pmatch_b`. For your convenience the hardware for `pmatch_a` is shown in the upper right. *Note: In the original exam the condition for* `delta_n` *was* `delta <= 0` *and the condition for* `delta_p` *was* `delta > 0`*. Though the hardware computed the correct result, the comparison would have been more expensive since it would have had to check for a zero condition, not just negative.*

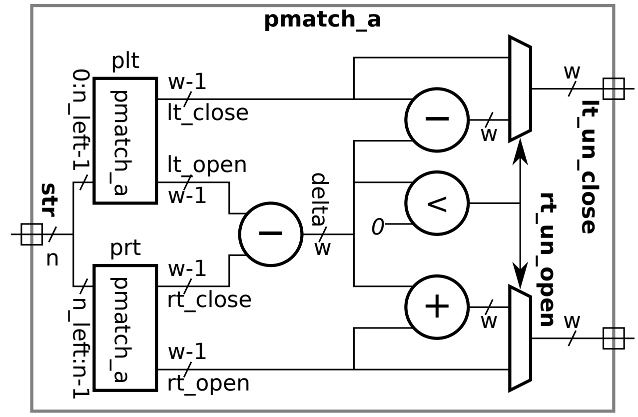
☐ Show inferred hardware on the facing page.

(*b*) Compute the simple-model cost of the hardware.

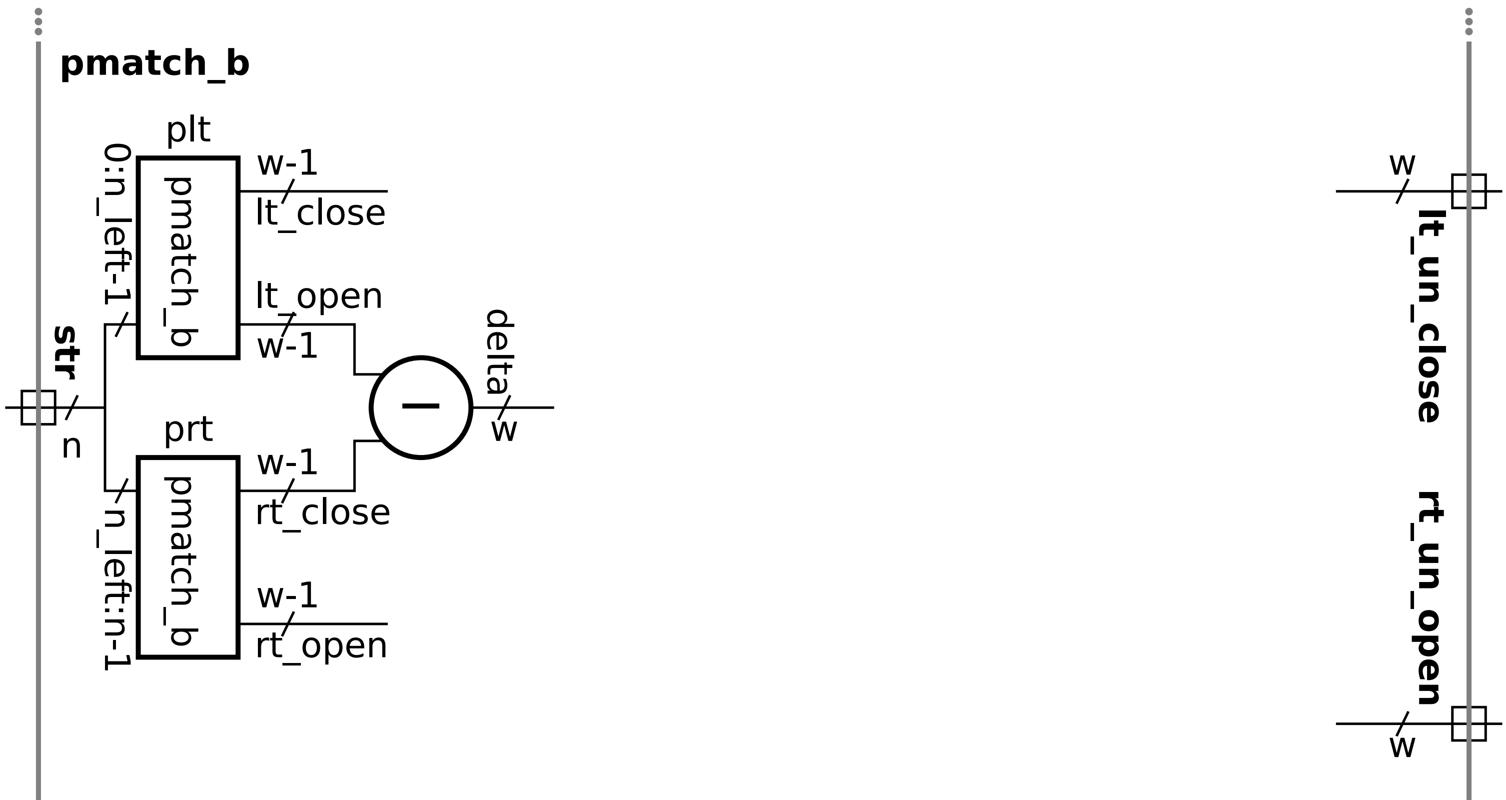
☐ Write $\boxed{\textit{same}}$ next to components that cost the same as corresponding components in `pmatch_a` and ☐ compute the cost of other components ☐ after optimization.

(*c*) Compare the critical path lengths.

☐ Will the critical path in `pmatch_a` be much different than the one in `pmatch_b`? ☐ Explain.



Staple This Side



Staple This Side

Problem 3: [20 pts] Appearing below are some of the `dot` modules from the solution to Homework 1. On the facing page is incomplete module `dotn`. Complete `dotn` so that it describes hardware that computes the dot product of `n`-element vectors recursively, where `n` is the parameter. That is, `dotn` must instantiate `dotn` and should instantiate `mult` and `add` where needed.

```
module mult #( int w = 5 ) ( output uwire [w-1:0] p, input uwire [w-1:0] a, b );
   assign p = a * b;
endmodule

module add #( int w = 5 ) ( output uwire [w-1:0] s, input uwire [w-1:0] a, b );
   assign s = a + b;
endmodule

module dot2 #( int w = 5 )
   ( output uwire [w-1:0] dp,    input uwire [w-1:0] a[1:0], b[1:0] );
   // Computes dp = a[0] * b[0] + a[1] * b[1];
   uwire [w-1:0] p0, p1;
   mult #(w) m0(p0, a[0], b[0] );
   mult #(w) m1(p1, a[1], b[1] );
   add #(w) ad(dp, p0, p1 );
endmodule

module dot3 #( int w = 5 )
   ( output uwire [w-1:0] dp,   input uwire [w-1:0] a[2:0], b[2:0] );
   // Computes   dp = a[0] * b[0] + a[1] * b[1] + a[2] * b[2];
   uwire [w-1:0] p0, p2;
   dot2 #(w) d0( p0, a[1:0], b[1:0] );
   mult #(w) m2( p2, a[2], b[2] );
   add #(w) a2(dp, p0, p2 );
endmodule
```

Staple This Side

Staple This Side

☐ Complete `dotn` so that it describes tree-structured hardware computing an `n`-element dot product. The tree depth should be $\lceil \lg n \rceil$.

☐ Instantiate `mult` for multiplication and `add` for addition, and of course `dotn` for a dot product of a smaller vector.

☐ To keep things easy all wires are `w` bits.

```
module dotn
  #( int w = 5, n = 4 )
   ( output uwire [w-1:0] dp,
     input uwire [w-1:0] a[n-1:0], b[n-1:0] );









endmodule
```


Staple This Side

Problem 4: [10 pts] Appearing below is the logarithmic shifter presented in class, followed by a version that's supposed to be better (but isn't). The hoped-for improvement is due to instantiating the exact number of multiplexors (`muxw2`) needed, rather than enough for the maximum shift amount.

```
module shift_right_logarithmic #( int w = 16, lgw = $clog2(w) )
   ( output uwire [w-1:0] shifted,
     input uwire [w-1:0] un,   input uwire [lgw-1:0] amt );
   // This module is correct.
   uwire [w-1:0] s[lgw:-1];
   assign s[-1] = un;
   for ( genvar i=0; i<lgw; i++ )
     muxw2 #(w) st( s[i],  amt[i],  s[i-1],  s[i-1] >> ( 1 << i )  );
   assign shifted = s[lgw-1];
endmodule

module shift_right_logarithmic_better_maybe #( int w = 16, lgw = $clog2(w) )
   ( output uwire [w-1:0] shifted,
     input uwire [w-1:0] un,  input uwire [lgw-1:0] amt );
   uwire [w-1:0] s[lgw:-1];
   assign s[-1] = un;

   // Use exactly the number of stages needed!!!
   uwire [lgw-1:0] lg_amt;              // LINE ADDED
   my_clog2 #(lgw) mc( lg_amt, amt );   // LINE ADDED. Set lg_amt = $clog2(amt) = ⌈lg amt⌉;

   for ( genvar i=0; i<lg_amt; i++ )    // LINE DIFFERS
     muxw2 #(w) st( s[i],  amt[i],  s[i-1],  s[i-1] >> ( 1 << i )  );
   assign shifted = s[lg_amt-1];        // LINE DIFFERS
endmodule
```

☐ Why won't the Verilog above compile?

☐ Is it possible to fix the Verilog error in such a way that cost is lower with smaller shift amounts? ☐ Explain.

☐ Is it possible to fix the Verilog error in such a way that the delay reported by a synthesis program is lower? ☐ Explain.

☐ Is it possible to fix the Verilog error in such a way that the delay actually is lower? ☐ Explain.

Staple This Side

Problem 5: [30 pts] Answer the following Verilog questions.

(*a*) The module below uses multidimensional arrays.

```
module mda( input uwire [2:1] c [5:1],    input uwire [7:1][2:1] a [5:1][3:1] );

   //     Add dimension(s) to the declaration of e so that the assignment is correct.
   //
   uwire             e             = c;


   //     Add dimension(s) to the declaration of b so that the assignment is correct.
   //
   uwire             b             = a[1][1][1];


   logic g [7:0];
   logic [7:0] h;

   initial begin
//     Which is correct,
          only the assignment to g,     only the assignment to h, or     both are correct.
       Explain.

      g[1] = h[1];
      h[1] = g[1];
   end

endmodule
```

What is the size of c, in bits? What is the size of a, in bits?

Staple This Side

(*b*) In the module below indicate whether each code fragment is correct.

```
module kinds #( logic [31:0] pg = 123 )
   ( output uwire [31:0] o0,  input uwire [31:0] ik );

   // ☐ Is the line below correct?       ◯ Yes   ◯ No   ☐ If not, explain.

   localparam logic [31:0] z02 = pg + 4755;




   // ☐ Is the line below correct?       ◯ Yes   ◯ No   ☐ If not, explain.

   localparam logic [31:0] z03 = ik;




   // ☐ Are the lines below correct?       ◯ Yes   ◯ No   ☐ If not, explain.

   localparam logic [31:0] z04;
   assign z04 = pg;




   // ☐ Are the lines below correct?       ◯ Yes   ◯ No   ☐ If not, explain.

   uwire [31:0] z10 = pg;
   assign z10 = ik;




   // ☐ Is the line below correct?       ◯ Yes   ◯ No   ☐ If not, explain.

   uwire [31:0] z13 = ik;



endmodule
```

Staple This Side

Staple This Side

(c) When we run a synthesis program we specify a delay target. In class we often synthesize twice, once with a delay target of 100 ns and a second time with a target of 0.1 ns. What is the harm in specifying a delay target lower (faster) than one needs? Isn't faster better?

☐ Harm in setting delay target too low is:

(d) A 32-bit signed integer, say i, is converted into a 32-bit IEEE 754 floating-point format (8-bit exponent, 23-bit significand) and then back into a 32-bit integer, j.

☐ Is it guaranteed that $i = j$ for all $-2^{31} \leq i < 2^{31}$? ☐ Explain based on the FP representation.

Staple This Side

Staple This Side