Name _____ *Formatted For 2-Sided Printing*

Staple This Side

Digital Design Using HDLs
LSU EE 4755
Midterm Examination

Friday, 30 October 2026, 11:30-12:20 CDT

Problem 1	_____	(10 pts)
Problem 2	_____	(15 pts)
Problem 3	_____	(10 pts)
Problem 4	_____	(10 pts)
Problem 5	_____	(15 pts)
Problem 6	_____	(15 pts)
Problem 7	_____	(15 pts)
Problem 8	_____	(10 pts)
Exam Total	_____	(100 pts)

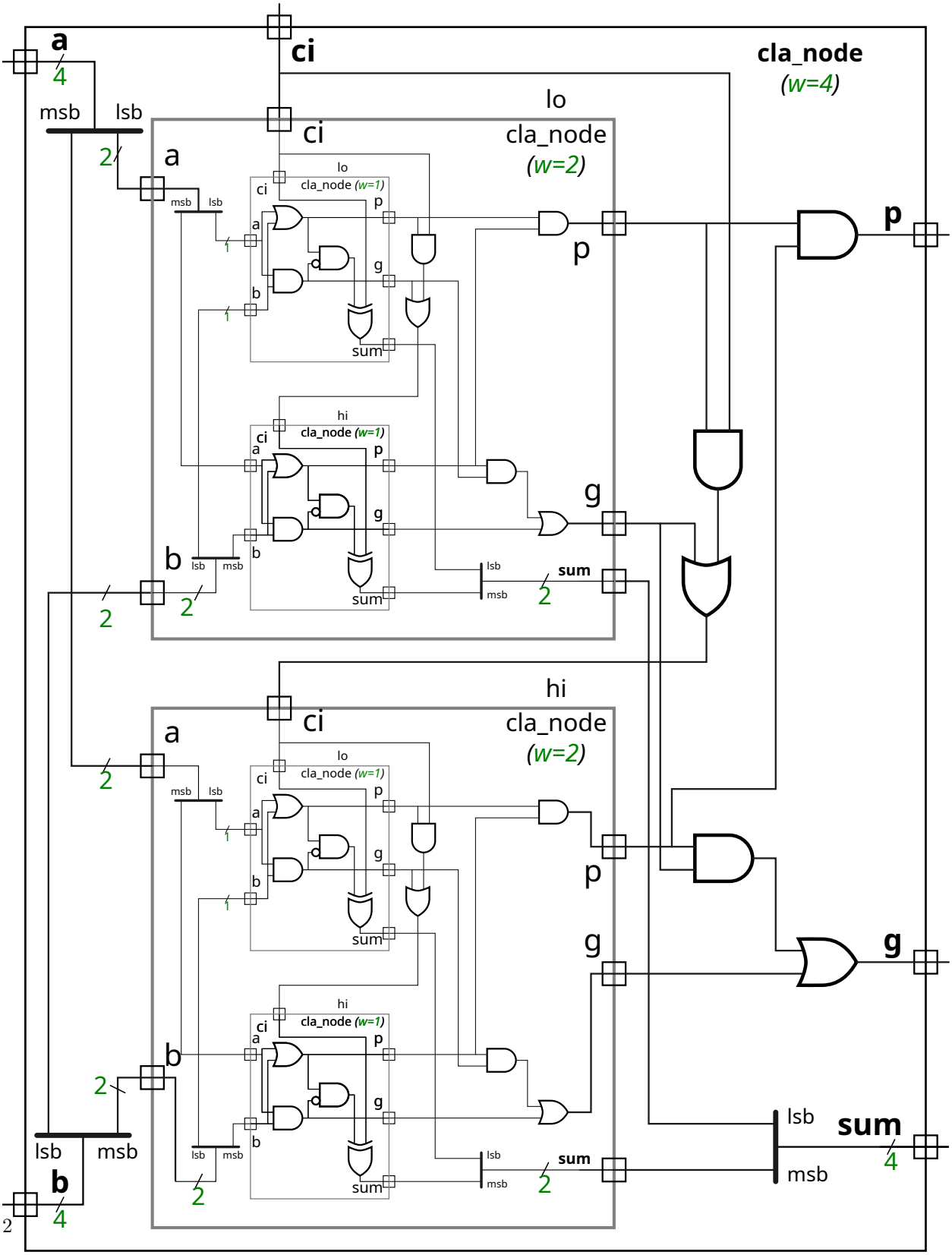
Staple This Side

Alias _____

Good Luck!

Problem 1: [10 pts] Appearing below is the hardware for a $w = 4$ CLA. Optimize the hardware for `cla_node.ci=1`.

☐ Optimize for `cla_node.ci=1`.



Staple This Side

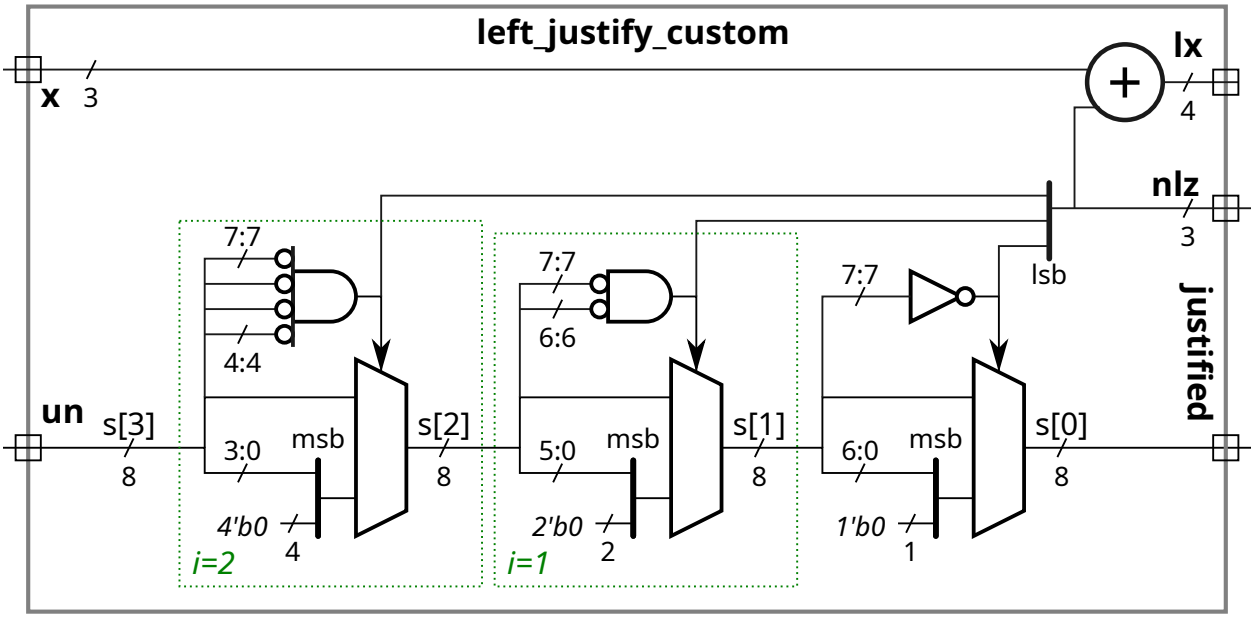
Staple This Side

Problem 2: [15 pts] The module below is based on `left_justify_custom` from Homework 2 and 3.

☐ Show ☐ the arrival time at each wire and show a ☐ critical path. ▷ The adder is a ripple adder and the high bit of its output is the carry out.

☐ Does accounting for constant inputs in the illustrated hardware affect the critical path? ☐ Explain.

Staple This Side



Staple This Side

Problem 3: [10 pts] Appearing below are CLA modules based on the ones used in class. Notice that module `cla` connects port `.cin` of the `cla_node` instantiation to the constant `1'b0`. We want the synthesis program to optimize for this constant, but it might not because it's from a port, not a parameter. This is especially true for the recursive instantiations. For that reason `cla_node` also has parameters that indicate whether the carry-in value should come from the parameter (so it will surely be optimized) or the port. If `cin_is_constant=0` then `cla_node` should use `cin` (the port), otherwise it should use `cin_cval` (the parameter).

☐ Complete `cla_node` so that it uses the appropriate carry-in value. ▷ Note that within the module `cinv` is used as the carry value. ☐ Don't forget about the recursive instantiations.

☐ Don't overdo it. In a correct solution only a few lines need to be modified.

```
module cla #( int w = 16, wb = 1 )
   ( output uwire [w-1:0] sum,  input uwire [w-1:0] a, b );
   // Do not modify this module.
   cla_node #(w,1,0) c(.sum(sum), .p(), .g(), .cin(1'b0), .a(a), .b(b) );
endmodule

module cla_node #( int w = 16, cin_is_constant = 0, cin_cval = 0 )
   ( output uwire [w-1:0] sum,   output uwire p, g,
     input uwire cin,            input uwire [w-1:0] a, b );


   uwire cinv = cin;



   if ( w == 1 ) begin
      assign g = a && b;
      assign p = a || b;
      assign sum = ( !g && p ) ^ cinv;
   end else begin

      localparam int wlo = w/2;
      localparam int whi = w - wlo;

      uwire plo, glo, phi, ghi;
      uwire chi = cinv && plo || glo;



      cla_node #(wlo) lo(sum[wlo-1:0], plo,glo, cinv, a[wlo-1:0],b[wlo-1:0] );



      cla_node #(whi) hi(sum[w-1:wlo], phi,ghi, chi, a[w-1:wlo],b[w-1:wlo] );



      assign p = plo && phi;
      assign g = glo && phi || ghi;
   end
endmodule
```


Staple This Side

Staple This Side

Problem 4: [10 pts] Input `f` to the module `tiger_ftoi` is an IEEE 754-like floating-point number using the specified exponent and fraction sizes, similar to Homework 2.

☐ Complete the module so that it sets output `i` to the *unsigned integer representation* the floating-point number on `f`.

☐ Round toward zero, so for $f < 1$ (including negative values) set $i = 0$.

☐ Solution can use procedural or structural code.

Staple This Side

```
module tiger_ftoi
  #( int w_i = 10, w_exp = 4, w_sig = 8, w_fp = w_exp + w_sig + 1 )
   ( output uwire [w_i-1:0] i,   input uwire [w_fp-1:0] f );

   localparam int bias = ( 1 << w_exp - 1 ) - 1;

   logic the_sign;
   logic [w_exp-1:0] exp;
   logic [w_sig-1:0] sig;
   always_comb { the_sign, exp, sig } = f;
   // The code above has already extracted the sign, exponent (biased),
   // and the significand.
```

Staple This Side

```
endmodule
```

Problem 5: [15 pts] The `clz_node` module is a variation of the tree `clz` presented in class.

☐ Show the inferred hardware for the module with $w > 2$ and w a power of 2.

☐ Show module ports. ☐ Use basic gates, multiplexors, arithmetic units.

☐ Do not confuse elaboration-time computation with computation hardware. ☐ **Do Not Use** `==` nor Verilog operators including concatenation expressions such as {a, b}.

```
module clz_node #( int w = 19, int ww = $clog2(w) )
   ( output uwire [ww-1:0] nlz,   input uwire [w-1:0] a );
   if ( w == 1 ) begin
      assign nlz = ww'(1'b0);
   end else if ( w == 2 ) begin
      assign nlz = !a[1];
   end else begin

      localparam int lhi = $clog2(w) - 1;
      localparam int whi = 1 << lhi;
      localparam int wwhi = lhi;
      localparam int wlo = w - whi;

      uwire [wwhi-1:0] nlz_lo, nlz_hi;

      clz_node #(whi,wwhi) clz_hi( nlz_hi, a[w-1:wlo] );
      clz_node #(wlo,wwhi) clz_lo( nlz_lo, a[wlo-1:0] );

      uwire hi_z = a[w-1:wlo] == whi'(1'b0);
      assign nlz = { hi_z, wwhi'( hi_z ? nlz_lo : nlz_hi ) } ;
   end
endmodule
```

Staple This Side

Staple This Side

Problem 6: [15 pts] Answer the questions below about multi-dimensional packed/unpacked arrays. ☐ Show work for partial credit. ☐ Show bit size in decimal, other values in binary or hexadecimal.

```
module loose_bits_sink_chips;
   logic [4:1][5:0] a [6:0][1:7];


   //  ☐  Number of bits in a:


   //  ☐  Complete the declaration of g.
   //
   //
   uwire         g        =  a[1][2][3];


   //  ☐  Complete the declaration of h.
   //
   //
   uwire         h        =  a[2];

   logic [2:0][8:1] cc;
   logic [3:1][8:1] cf;
   logic [3:0][5:0] d;
   logic [3:0][3:0] e, f;
   int unsigned x;

   initial begin
      cc = 'hfedcba;
      x = cc[1];
      //  ☐  Show value of x:

      cf = 'hfedcba;
      x = cf[1];
      //  ☐  Show value of x:

      d = cc;
      x = d[0];
      //  ☐  Show value of x:

      e = 'hf9ac;
      f = e + 8;
      //  ☐  Show value of f:

      e[1] = e[1] + 9;
     //  ☐  Show value of e (not just e[1]):

   end

endmodule
```

Staple This Side

Staple This Side

Problem 7: [15 pts] The modules in this problem compute a totient. *Note: The totient (rhymes with quotient) of n is the number of integers in $[1, n)$ that are relatively prime to n (don't divide n evenly).*

☐ Explain why the module below is not synthesizable.

```
module totient #( int w = 3 )
   ( output logic [w-1:0] toti,   input uwire [w-1:0] n );
   always_comb begin
      toti = 0;
      for ( int i=2; i<n/2; i++ )
        if ( n / i * i != n ) toti = toti + 1;
   end
endmodule
```

☐ Fix the synthesizability issue using `nmax`. ▷ Don't worry about hardware cost.

```
module totient_syn #( int w = 10 )
   ( output logic [w-1:0] toti,    input uwire [w-1:0] n );
   localparam logic [w-1:0] nmax = ( (w+1)'(1'b1) << w ) - 1;

   always_comb begin

      toti = 0;


      for ( int i=2; i<n/2; i++ )


        if ( n / i * i != n ) toti = toti + 1;


   end
endmodule
```

☐ Why is hardware cost a concern if `totient_syn` were made synthesizable with a small change?

Staple This Side

Staple This Side

☐ What is the cost of the module below *after* optimization? ☐ Explain.

☐ Don't waste time on a lengthy answer.

```
module totient_plan_bb #( int w = 10, logic [w-1:0] n = 4 )
   ( output logic [w-1:0] toti );
   always_comb begin
      toti = 0;
      for ( int i=2; i<n/2; i++ )
        if ( n / i * i != n ) toti = toti + 1;
   end
endmodule
```

Staple This Side

☐ Does module `totient_plan_b`, below, solve the synthesizability issue in `totient`? ☐ Explain.

```
module totient_plan_b #( int w = 10 )
   ( output logic [w-1:0] toti,  input uwire [w-1:0] n );
   totient_plan_bb #(w,n) tpb( toti );
endmodule
```

Staple This Side

Problem 8: [10 pts] Answer each question below.

(*a*) Appearing below are synthesis runs on variations of the Homework 2 left-justify operation.

Module Name	Area	Delay Actual	Delay Target	Synth Time
left_justify_procedural_w4	3995	0.44	3.0 ns	1 s
left_justify_parts_w4	3995	0.44	3.0 ns	1 s
left_justify_one_w4	3995	0.44	3.0 ns	1 s
left_justify_custom_w4	2762	0.40	3.0 ns	1 s
left_justify_procedural_w31	170006	2.94	3.0 ns	48 s
left_justify_parts_w31	86047	2.93	3.0 ns	14 s
left_justify_one_w31	68402	2.98	3.0 ns	11 s
left_justify_custom_w31	59156	3.00	3.0 ns	9 s

☐ In the table above why are three of the modules' delays identical at $w = 4$, but different from each other at $w = 31$?

☐ For the delay target of 3 ns and $w = 31$ which module should be chosen? ☐ Explain.

Staple This Side

Staple This Side

(*b*) In an execution of the Verilog below inputs to `ripple3` are stable up until $t = 100$ when input `a` changes (but the other inputs hold steady). Note that there are three `bfa` instances, `ripple3.f0` `ripple3.f1`, and `ripple3.f2`. Call them `f0`, `f1`, and `f2` for short.

```
module bfa(output uwire c, s, input uwire a, b, cin);
   assign {c,s} = { a & b | a & cin | b & cin, a ^ b ^ cin };
endmodule

module ripple3 #( int w = 3 )
   ( output uwire [w-1:0] sum,   output uwire cout,
     input uwire [w-1:0] a, b,   input uwire cin );

   bfa f0( c0,   sum[0], a[0], b[0], cin );
   bfa f1( c1,   sum[1], a[1], b[1], c0 );
   bfa f2( cout, sum[2], a[2], b[2], c1 );

endmodule
```

☐ Show the order of execution of the `bfa` instances starting at $t = 100$ (which is when `a` changes).

☐ Note that an instance can execute more than one.

☐ In your answer show the points at which the inactive region is copied into the active region.

Start of answer: `f0`, `f1`,

Staple This Side

Staple This Side

This page intentionally blank.

Staple This Side

Staple This Side