Name _____ *Formatted For 2-Sided Printing*

Staple This Side

Digital Design Using HDLs
LSU EE 4755
Midterm Examination

Friday, 27 October 2023, 11:30-12:20 CDT

Staple This Side

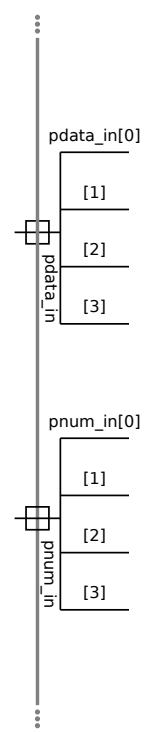
Problem 1	_____	(30 pts)
Problem 2	_____	(25 pts)
Problem 3	_____	(30 pts)
Problem 4	_____	(15 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Problem 1: [30 pts] Appearing below is the permutation module from the solution to Homework 3. Using the illustration of the ports show the inferred hardware for an instantiation with `n=4`. Show the `n=4` instantiation but not what is inside the `n=3` recursive instantiation.

```
module perm
  #( int w = 8, n = 20, wd = $clog2(n) )
   ( output uwire [w-1:0] pdata_out[n],    output uwire [wd-1:0] pnum_out[n],
     output uwire carry_out,
     input uwire [w-1:0] pdata_in[n],      input uwire [wd-1:0] pnum_in[n] );

   if ( n == 1 ) begin

      assign pdata_out[0] = pdata_in[0];
      assign carry_out = 1;
      assign pnum_out[0] = 0;

   end else begin

      uwire [wd-1:0] pos = n - 1 - pnum_in[n-1];
      assign pdata_out[n-1] = pdata_in[pos];
      uwire [w-1:0] prdata_in[n-1];
      for ( genvar i=0; i<n-1; i++ )
        assign prdata_in[i] =  i < pos  ?  pdata_in[i]  :  pdata_in[i+1];

      uwire co;
      perm #(w,n-1,wd) rp( pdata_out[0:n-2], pnum_out[0:n-2], co,
                           prdata_in, pnum_in[0:n-2] );

      uwire [wd-1:0] dnext = pnum_in[n-1] + co;
      assign carry_out = dnext >= n;
      assign pnum_out[n-1] = carry_out ? 0 : dnext;

   end

endmodule
```

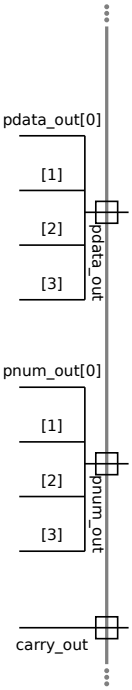


Staple This Side

Staple This Side

☐ Show inferred hardware for `n=4`. Be sure to use ☐ the illustrated module ports and to show ☐ the recursively instantiated module (but not its contents).

☐ Show hardware, ☐ do not confuse elaboration-time computation with computation hardware.



Staple This Side

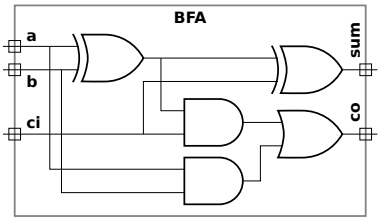
Problem 2: [25 pts] A ripple adder to compute $a + b$ is to be used in situations where a is a constant.

(a) Find the cost and delay of a BFA with input a constant (for use in the ripple adder). A BFA is shown for your convenience.

☐ Show the BFA(s) optimized for input a constant.

☐ Use a truth table to find optimizations not revealed by constant pushing: in a correct solution the delay does not depend upon `a`.

☐ Show simple-model cost of this(these) module(s) and ☐ show simple-model delay(s) of this(these) module(s).



(b) On the facing page show the optimized hardware, cost, LSB delay, and MSB delay of a w-bit ripple adder for computing $a + b + c_{\text{in}}$, where c_{in} is a carry-in bit (`cin` in the diagram) and a is a constant. (☐ See the check box items for details.) Use the illustration on the facing page as a starting point.

☐ Show the hardware optimized for a constant `a` and a non-constant `cin`.

☐ Compute the simple-model cost of this hardware in terms of w.

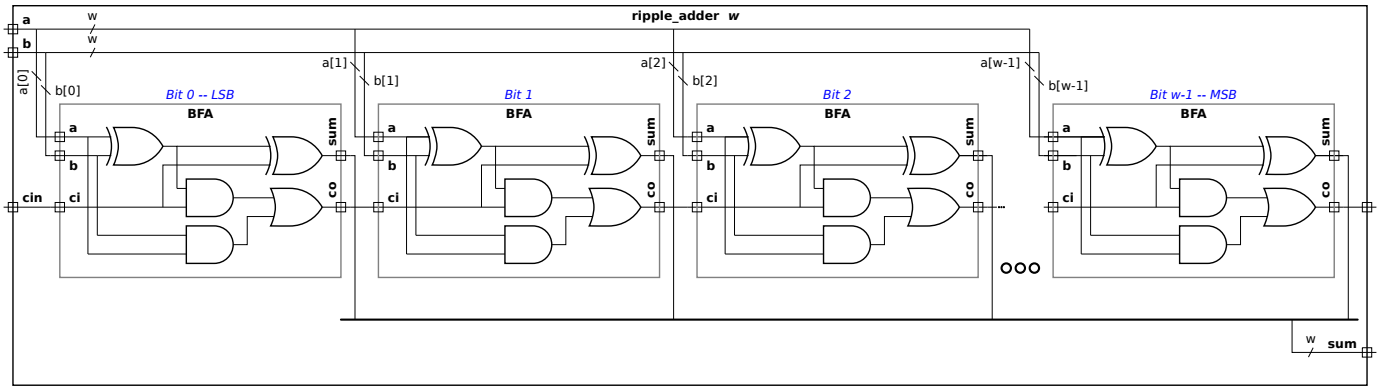
☐ Compute the simple-model delay of the LSB of the sum.

☐ Compute the simple-model delay of the MSB of the sum in terms of w and ☐ show the critical path.

☐ Don't forget that `a` is a constant.

Staple This Side

Staple This Side



(*c*) If `cin` were removed (or set to zero) the cost and delay of the optimized adder would depend on `a`. Explain why, and illustrate with the example of `a=2`.

☐ How are cost and delay dependent on `a` when `cin` removed? ☐ Explain using the example `a=2`.

Staple This Side

Problem 3: [30 pts] Answer the following Verilog questions.

(*a*) The module below makes extensive use of 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[1];


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


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

   initial begin
//  ☐  Which is correct,  ◯ the assignment to g  or  ◯ the assignment to h.  ☐  Explain.
      g = 1;
      h = 1;
   end

endmodule
```

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

(*b*) The module below does not compile.

```
module more_stuff #( int n = 20 ) ( output uwire [31:0] sum, input uwire [31:0] a [ n ] );
   logic [31:0] acc;
   always_comb begin
      acc = a[0];
      for ( int i=1; i<n; i++ )
        my_fixed_adder a1(acc, acc, a[i] );
   end
   assign sum = acc;
endmodule
```

☐ Describe the major problem. ☐ **DO NOT** try to fix the problem.

Staple This Side

Staple This Side

(*c*) The module below is supposed to set $x = a^2 + b^2$.

```
module wrong_way( output logic [31:0] x, input uwire [15:0] a, b );
   logic [31:0] asq;
   uwire [31:0] bsq = b * b;

   initial asq = a * a;
   always_comb x = asq + bsq;

endmodule
```

☐ Explain the problem. ☐ Using sample inputs show the expected output and the actual output.

☐ Fix the problem.

(*d*) The module below does not compile.

```
module my_adder( output uwire [31:0] s, input uwire [31:0] a, b );
   always_comb s = a + b;
endmodule
```

☐ Why won't module above compile? ☐ Fix problem by changing declarations.

(*e*) The module below compiles but does not provide the expected outputs, $p_a = a^2$, $p_b = b^2$, and $p = a^2 + b^2$.

```
module incorrect_way( output logic [31:0] pa,pb,p, input uwire [15:0] a, b );
   wire [31:0] sq;
   assign sq = a * a;
   always_comb pa = sq;
   assign sq = b * b;
   always_comb pb = sq;
   always_comb p = pa + pb;
endmodule
```

☐ What will be the values of outputs `pa`, `pb`, and `p`?

☐ Describe the problem. ☐ Fix it.

Staple This Side

Problem 4: [15 pts] Answer each question below.

(a) A company has two teams, A (very good) and C (slackers) working on modules and a testbench for an important product. Describe the following consequences:

☐ The A team works on the modules and the C team works on the testbench. A possible bad outcome is:

☐ The A team works on the testbench and the C team works on the modules. A possible bad outcome is:

(b) In typical use when running simulation a testbench generates inputs for a module-under-test and the outputs are checked by the testbench to see whether they are correct. After running synthesis we learn how fast the module is. If simulation is computing the module outputs why can't it tell us how fast the module is?

☐ Synthesis can provide timing information and simulation can't because:

(c) A gadget can be build using an ASIC or an FPGA. Describe which is more appropriate for each situation below.

☐ The gadget must be working within a month. ◯ *ASIC* or ◯ *FPGA*. ☐ Explain.

☐ Per-gadget cost must be under $1000. Only ten will be made. ◯ *ASIC* or ◯ *FPGA*. ☐ Explain.

☐ Per-gadget cost must be under $100. Ten thousand will be made. ◯ *ASIC* or ◯ *FPGA*. ☐ Explain.

Staple This Side

Staple This Side