Name _____

Digital Design using HDLs
EE 4755
Midterm Examination

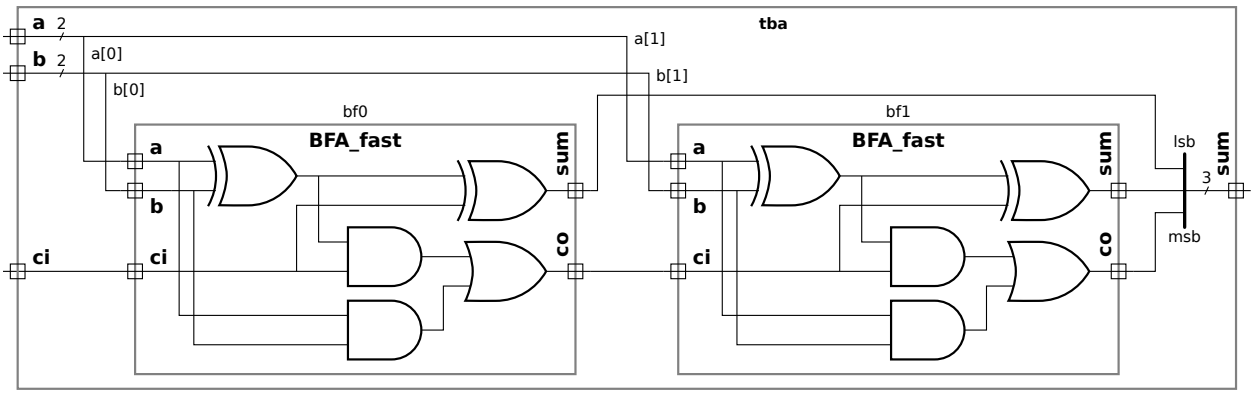
Monday, 16 October 2017 9:30–10:20 CDT

Problem 1	_____	(20 pts)
Problem 2	_____	(20 pts)
Problem 3	_____	(20 pts)
Problem 4	_____	(15 pts)
Problem 5	_____	(10 pts)
Problem 6	_____	(15 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Problem 1: [20 pts] Write a Verilog description of the hardware illustrated below. The description **must include the modules and instantiations as illustrated**. The description can be behavioral or structural, but it must be synthesizable.



☐ Verilog corresponding to illustrated hardware.

☐ Show instantiations, ☐ Verilog for instantiated module(s), ☐ and all module ports.

Problem 2: [20 pts] Appearing below is a partially completed recursive description of an $n = 2^b$-input, w-bit multiplexor, which is a generalized version of the multiplexors appearing in Homework 1. Complete it.

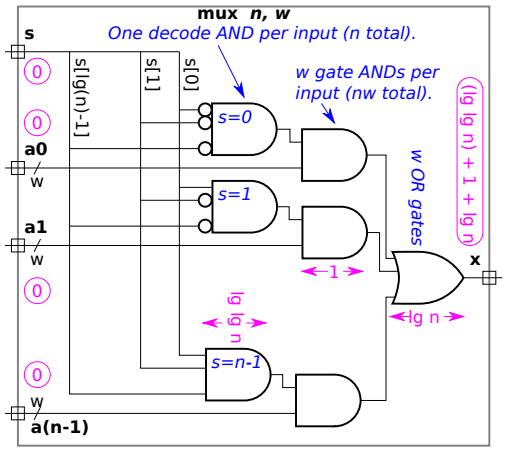
☐ Fill in the condition and code for the terminating case.

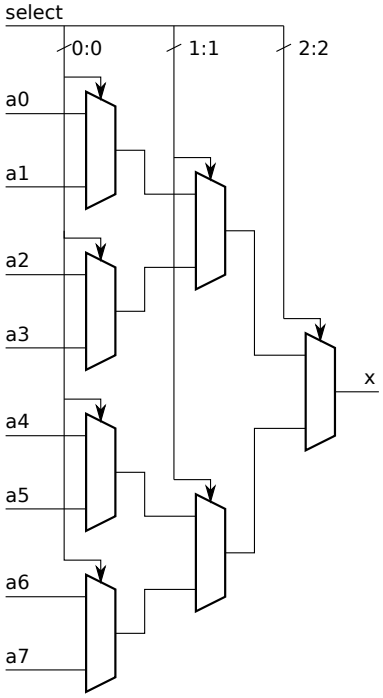
☐ Complete recursive case, including the instantiation port and parameter connections (look for FILL IN).

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

   if (         )  // Terminating Case Condition              <----  ☐  FILL IN
     begin
        // Terminating Case




     end else begin
        // Recursive Case

        uwire [w-1:0] y[2];

        // Instantiate two n/2-input muxen, and connect each to half the inputs.
        //
        //         ----      ----                               <----  ☐  FILL IN
        muxn #( .w(    ), .b(    ) ) mlo( y[0],  sel[b-2:0],  a[   0 : n/2-1 ] );



        //         ----      ----                    -----      <----  ☐  FILL IN
        muxn #( .w(    ), .b(    ) ) mhi( y[1],  sel[     ],  a[ n/2 : n-1   ] );









        // Instantiate one 2-input mux.
        //
        //         ----      ----       --------------------   <----  ☐  FILL IN
        muxn #( .w(    ), .b(    ) ) m2(                    )



   end

endmodule
```

Problem 3: [20 pts] Appearing below to the right is an 8-input multiplexor constructed from 2-input multiplexors using the technique from Homework 1 and from the previous problem. Call a multiplexor constructed this way a *tree mux*. Appearing below to the left is a diagram showing a *flat mux*, the kind usually used in class. The flat mux diagram shows a timing analysis based on the simple model, and some details about cost.

For reference: $\sum_{i=0}^{b-1} a2^i = a(2^b - 1)$. Assume that n is a power of 2.





(*a*) Compute the cost of an n-input, w-bit flat mux using the simple model and without optimization.

☐ Cost of flat mux **in terms of** n **and** w.

(*b*) Compute the cost of an n-input, w-bit tree mux using the simple model.

☐ Cost of tree mux **in terms of** n **and** w. ☐ Describe assumptions made about 2-input mux implementation.

(*c*) Compute the delay of an n-input, w-bit tree mux using the simple model.

☐ Delay of tree mux **in terms of** n **and** w.

Problem 4: [15 pts] Show the hardware that will be synthesized for the modules below.

(*a*) Show the hardware that will be inferred for the module below, including the minimum number of bits in each wire. Assume that `sqrt` is defined in a library somewhere.

```
module wqf
  #( int w = 16 )
   ( output logic signed [2*w-1:0] rad,
     output uwire [31:0] srad,
     input uwire [w-1:0] a, b, c );

   sqrt #(32,2*w) s1(srad,rad);

   always_comb begin

      rad = b*b - 4 * a * c;
      if ( rad < 0 ) rad = 0;

   end

endmodule
```

☐ Show inferred hardware. ☐ Show minimum correct bit widths.

(*b*) Show the hardware that will be inferred for the module below.

```
module sort2 #( int w = 4 )
   ( output logic [w-1:0] x[2], input uwire [w-1:0] a[2] );

   always_comb begin

      for ( int i=0; i<2; i++ ) x[i] = a[i];
      if ( a[0] < a[1] ) begin x[0] = a[1]; x[1] = a[0]; end

   end

endmodule
```

☐ Show inferred hardware.

Problem 5: [10 pts] Answer each question below.

(*a*) The `mux2` module below uses implicit structural code. Modify it so that it uses behavioral (procedural) code.

```
module mux2 #( int w = 16 )
   ( output uwire [w-1:0] x,
     input uwire s, input uwire [w-1:0] a,b );

   assign  x = s == 0 ? a : b;

endmodule
```

☐ Modify so that is procedural. ☐ Change ports if necessary.

(*b*) Modify the module port and parameter declarations below so that the Verilog is correct. Do not modify the contents of the module itself. Note that `opt` is not defined, but that it should be. *Note: In the original exam* `assign` *was omitted from the module body, making the problem impossible to solve.*

```
module sum_or_dff
  #( int w = 16 )
   ( output uwire [w-1:0] x,
     input uwire [w-1:0] a, b );

   if ( opt == 0 ) assign x = a+b; else assign x = a-b;

endmodule
```

☐ Modify port and parameter declarations for correctness.

Problem 6: [15 pts] Answer each question below.

(*a*) Why is `always_comb` preferred over `always @( x or y or .. )` when describing combinational logic?

☐ `always_comb` preferred because ...

☐ What is the risk with `always @( x or y or .. )`?

(*b*) Describe what the technology mapping step of synthesis is, and the kind of optimizations that need to be performed after technology mapping.

☐ Technology mapping is:

☐ Optimizations that must be performed after technology mapping:

(*c*) The module below adds a real and an integer and assigns the sum (in real format) to its output. It is valid Verilog but is not synthesizable by Owr EDA software. So, you call Owr EDA and ask, "why not?". They answer, "because it is impossible to add an integer to a real." Is that the real reason? Explain.

```
module plusri (output real sum, input real a, input [20:0] x);
   assign       sum = a + x;
endmodule
```

☐ Reason a+x not synthesizable by Owr EDA software: