Name _____

Digital Design Using HDLs
LSU EE 4755
Solve-Home Midterm Examination

Friday, 6 Nov 2020 to early Monday, 9 Nov 2020 05:00 CST)

Work on this exam alone. Regular class resources, such as notes, papers, documentation, and code, can be used to find solutions. Outside material that covers the same topics, such as Verilog tutorials, digital logic design guides can also be used. Do not try to directly seek out solutions to any question here. That is, don't Web-search the text of a problem. Do not discuss this exam with classmates or anyone else, except questions or concerns about problems should be directed to Dr. Koppelman.

Warning: Unlike homework assignments collaboration is not allowed on exams. Suspected copying will be reported to the dean of students. The kind of copying on a homework assignment that would result in a comment like "See ee4755xx for grading comments" will be reported if it occurs on an exam. Please do not take advantage of pandemic-forced test conditions to cheat!

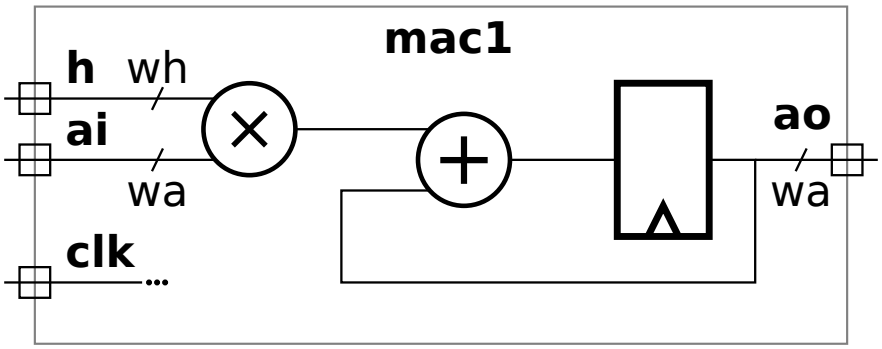
Problem 1	_____	(20 pts)
Problem 2	_____	(20 pts)
Problem 3	_____	(20 pts)
Problem 4	_____	(20 pts)
Problem 5	_____	(20 pts)
Exam Total	_____	(100 pts)

☣ $r \geq 2\,\mathrm{m} \quad \Rightarrow \quad R_e < 1$

Good Luck!

Problem 1: [20 pts] Appearing below are some variations on a multiply accumulate module.

(*a*) Complete the Verilog code below so that it matches the illustration.



☐ Complete the Verilog.

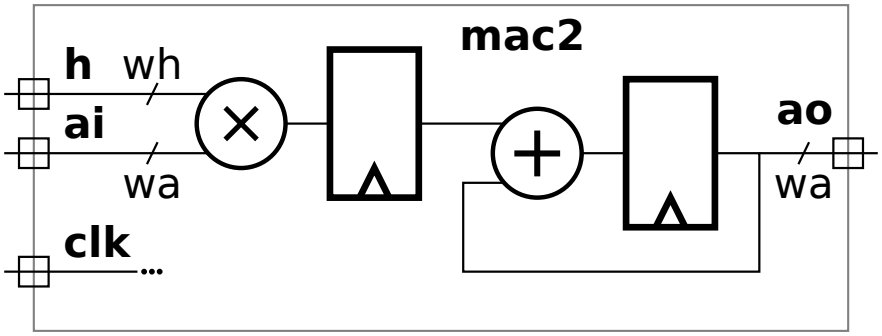
☐ Use parameters for the bit widths `wh` and `wa`.

☐ The registers inferred from the Verilog must match the diagram.

```
module mac1

endmodule
```

(*b*) Complete the Verilog code below so that it matches the illustration, similar to the one on the previous page.



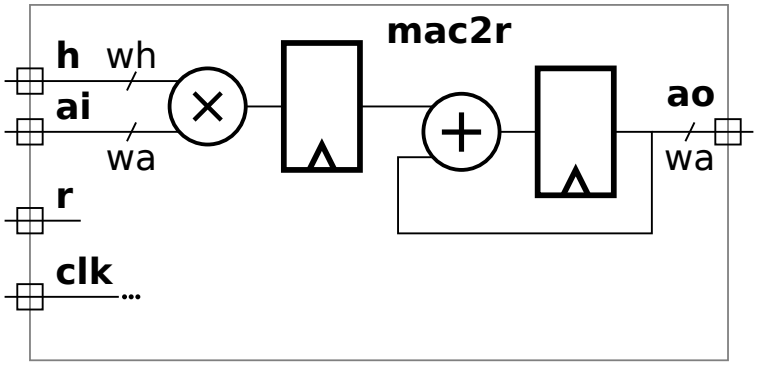
- [ ] Complete the Verilog.
- [ ] Use parameters for the bit widths `wh` and `wa`.
- [ ] The registers inferred from the Verilog must match the diagram.

```
module mac2

endmodule
```

Problem 2: [20 pts] The mac (multiply-accumulate) modules compute a running sum of products. The alert student might have noticed that there is no way to reset the sum. In this problem a reset will be added.

The module below has an input `r` (for reset) which is to work as follows: When `r=1` at a positive edge the product `h*ai` should start a new running sum. That is, that particular `h*ai` should be added to zero. When `r=0` at a positive edge the product `h*ai` should be added to the sum of the previous products. (If `r=0` is always true then the hardware as illustrated works correctly.)



☐ Add hardware to the diagram to implement the reset. ☐ Complete the Verilog to implement the reset.

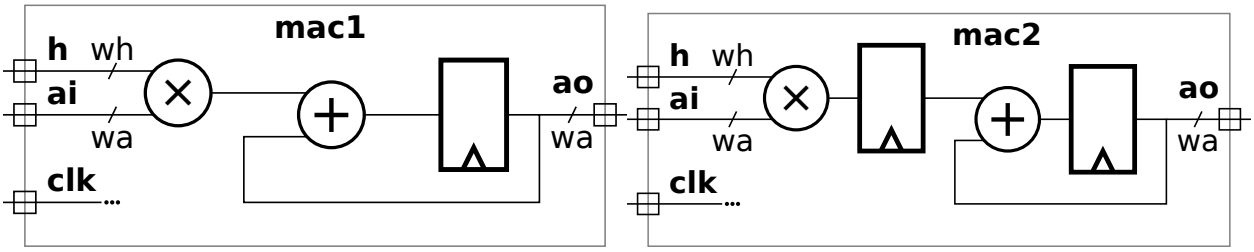
☐ Use parameters for the bit widths `wh` and `wa`.

☐ The registers inferred from the Verilog must match the diagram and ☐ be sure that the reset is applied to the correct value.

```
module mac2r
```

```
endmodule
```

Problem 3: [20 pts] Appearing below are the modules from the previous problem. Suppose that in the multiplier below bit i of the product were computed in time $[4i+2]\,\mathrm{u_t}$ and that a ripple adder were used for the sum. Let w denote the value of `wh` and `wa` (which means `wh==wa`).



(*a*) Find the minimum clock period for each using the simple model, and taking into account cascading. (The clock period is the length of the critical path, including the register delay.)

☐ Find the clock period for `mac1` with cascading. ☐ Don't forget to include the delay of the register.

☐ Find the clock period for `mac2` with cascading. ☐ Don't forget to include the delay of the registers.

(*b*) Find the minimum clock period for each using the simple model assuming that the multiplier output and adder input could not cascade.

☐ Find the clock period for `mac1` without cascading. ☐ Don't forget to include the delay of the register.

☐ Find the clock period for `mac2` without cascading. ☐ Don't forget to include the delay of the registers.

Problem 4: [20 pts] Appearing below is a recursively defined multiplier constructed using bfa (binary full adder) and bha (binary half adder) modules.

```
module mult_tree_bfas #( int wa = 16, int wb = wa, int wp = wa + wb )
   ( output uwire [wp-1:0] prod, input uwire [wa-1:0] a, input uwire [wb-1:0] b );

   if ( wa == 1 ) begin
      assign prod = a ? b : 0;
   end else begin
      // Split a in half and recursively instantiate a module for each half.
      localparam int wn = wa / 2;
      localparam int wx = wb + wn;
      uwire [wx-1:0] prod_lo, prod_hi;

      mult_tree_bfas #(wn,wb) mlo( prod_lo, a[wn-1:0],  b );
      mult_tree_bfas #(wn,wb) mhi( prod_hi, a[wa-1:wn], b );

      assign prod[wn-1:0] = prod_lo[wn-1:0];

      uwire c[wp-1:wn-1];
      assign c[wn-1] = 0;
      for ( genvar i=wn; i<wx; i++ )
        bfa b(  c[i],  prod[i],  prod_lo[i],    prod_hi[i-wn],  c[i-1] );
      for ( genvar i=wx; i<wx+wn; i++ )
        bha b(  c[i],  prod[i],  prod_hi[i-wn],                 c[i-1] );
      localparam int wz = wp - wx - wn;
      if ( wz > 0 ) assign prod[wp-1 :- wz] = 0;
   end
endmodule
```

Show the hardware that will be inferred for two levels of recursion and compute its cost. That is, show three instances of `mult_tree_bfas`: a top-level one, and two recursive instantiations. Show the hardware for the top-level instance and both of the two recursive instantiations. (It is only necessary to show two levels.) Do this for `wa=8` in the top-level module.

☐ Show the inferred hardware.

☐ Be sure to distinguish hardware (such as a `bfa` module) from values computed during elaboration.

☐ Compute the cost of the hardware in your diagram using the simple model. (Work out the cost of a bha by hand.) ☐ The cost should be for two levels, not for hardware going down to the base case.

Problem 5: [20 pts] Answer each question below.

(*a*) Appearing below is a multiply/add module, `nnMADDfp`, that computes its result using a FP add and multiply module. The values on the ports are IEEE 754 floats, and when `wa=32` the format is IEEE 754 single, the same as a SystemVerilog `shortreal`. That is followed by an incomplete testbench module, `testnnMADD`. The testbench module generates random values for the `nnMADDfp` module in variables `ar`, `br`, and `sir`, and computes what the result should be, `sor`.

Add Verilog code to deliver `ar`, `br`, and `sir` to the `nnMADDfp` instance, and to put the output of `nnMADDfp` into `sor_mut` so that `sor_mut` has the correct type of value. Note that one does not need to understand what is inside of `nnMADDfp`, `nnAddfp`, nor `nnMultfp`.

☐ Deliver (whatever that means) `ar`, `br`, and `sir` to `nnMADDfp` instance. ☐ Get output of the `nnMADDfp` instance into variable `sor_mut`.

```
module nnMADDfp #( int wa = 10 )
  ( output uwire [wa-1:0] so, input uwire [wa-1:0] a, b, si);
   uwire [wa-1:0] p;
   nnMultfp #(wa) mu(p, a, b);
   nnAddfp #(wa) ad(so, si, p);
endmodule

module testnnMADD;
   localparam int w = 32, ntests = 100;
   uwire [w-1:0] so;
   logic [w-1:0] a, b, si;
   nnMADDfp #(w) n(so, a, b, si);

   initial begin

      for ( int t=0; t<ntests; t++ ) begin
         shortreal sor, ar, br, sir, sor_mut;
         ar = rand_fp();    // Value to be used as input a to nnMADDfp.
         br = rand_fp();    // Value to be used as input b to nnMADDfp.
         sir = rand_fp();   // Value to be used as input si to nnMADDfp.
         sor = ar * br + sir;




         #1;




         sor_mut =                          ; // <-- DON'T FORGET.

         if ( sor != sor_mut ) handle_incorrect_result();

      end
   end
endmodule
```

(*b*) The module below will not compile or simulate due to multiple assignments to `temperature`, which is declared `uwire`. Changing `uwire` to `wire` will fix the compile problem. Nevertheless, is that the right fix?

```
module more_stuff #( int w = 16 )
   ( output uwire [w-1:0] v, y, input uwire [w-1:0] a, b, c );

   uwire [w-1:0] temperature;

   assign temperature = a + b;
   assign v = temperature >> c;
   assign temperature = a - b;
   assign y = temperature << c;

endmodule
```

☐ What problem remains after changing `temperature` from a `uwire` to a `wire`?

☐ Fix the problem based on what the code looks like its trying to do.

(*c*) An important part of synthesis is optimizing. It is possible to optimize before and again after technology mapping.

☐ What is technology mapping? ☐ Show an example of logic before and after technology mapping. (Make up some technology.)

☐ Describe an optimization that can be done before technology mapping. Provide an example. (This is done all the time in class.)

☐ Describe an optimization that can be done only after technology mapping (or perhaps during). Provide an example, feel free to make things up.