Name ______________________ *Formatted For 2-Sided Printing*

Staple This Side

Digital Design Using HDLs
LSU EE 4755
Midterm Examination

Wednesday, 19 October 2022, 11:30-12:20 CDT

Problem 1	______	(25 pts)
Problem 2	______	(31 pts)
Problem 3	______	(20 pts)
Problem 4	______	(12 pts)
Problem 5	______	(12 pts)
Exam Total	______	(100 pts)

Staple This Side

Alias ______________________

Good Luck!

Problem 1: [25 pts] Answer the following multiplexor questions.

(*a*) Complete module mux4 so that it implements a 4-input multiplexor using instantiations of the 2-input multiplexor shown below. Do not use procedural code.

☐ Complete mux4 so that it implements a 4-input multiplexor ☐ using mux2 instantiations.

☐ Do not use procedural code. ☐ Do not change the ports or default parameters of mux4 or mux2.

☐ Don't forget to declare any objects that are used.

```
module mux4
  #( int w = 3 )
   ( output uwire [w-1:0] x,
     input uwire [1:0] s,      input uwire [w-1:0] a0, a1, a2, a3 );











endmodule

module mux2
  #( int w = 6 )
   ( output uwire [w-1:0] x,
     input uwire s,   input uwire [w-1:0] a0, a1 );
   assign x = s ? a1 : a0;
endmodule
```

(*b*) Module `mux2_bad` only works for `w=1`. Describe the problem and show the correct mux output and the output of `mux2_bad` for `w=4`, `s=0`, `a0=2`, and `a1=4`.

```
module mux2_bad
  #( int w = 4 )
   ( output uwire [w-1:0] x,
     input uwire s,   input uwire [w-1:0] a0, a1 );
   assign x = !s && a0  ||  s && a1;
endmodule
```

☐ In `mux2` (a correct mux) when `w=4`, `s=0`, `a0=2`, and `a1=4`, ☐ output `x=`

☐ In `mux2_bad` when `w=4`, `s=0`, `a0=2`, and `a1=4`, ☐ output `x=`

☐ Explain the problem when `w` is not `1`.

(*c*) Complete module `mux2_1r` below so that it recursively implements a 2-input w-bit mux. All that remains to be done is completing the connections to the two recursive instances, `m1` and `mr`.

```
module mux2_1r
  #( int w = 5 )
   ( output uwire [w-1:0] x,
     input uwire s,   input uwire [w-1:0] a0, a1 );

   if ( w == 1 ) begin
     assign x = !s && a0  ||  s && a1;
   end else begin

      mux2_1r #(1)    m1(


      mux2_1r #(w-1)  mr(


   end

endmodule
```

Problem 2: [31 pts] The `val` output of `atoi_it_m_to_l` is the value of the radix-`r` ASCII-represented number appearing at its input, `str`, and output `nd` is the number of digits. Unlike the Homework 2 Problem 2 module, this module starts at the most-significant digit rather than the least-significant digit.

```
module atoi_it_m_to_l
  #( int r = 11, n = 5, wv = $clog2( r**n ), wd = $clog2(n+1) )
   ( output logic [wv-1:0] val,
     output logic [wd-1:0] nd,
     input uwire [7:0] str [n-1:0] );

   uwire [wv-1:0] vali[n:0];
   uwire is_digit[n:0];
   uwire [wd-1:0] ndi[n:0];
   assign is_digit[n] = 0;
   assign ndi[n] = 0;
   assign vali[n] = 0;
   assign nd = ndi[0];
   assign val = vali[0];

   localparam int wcv = $clog2(r);

   for ( genvar i=n-1; i>=0; i-- ) begin

      // Find Value of Digit i
      uwire [wcv-1:0] vald;
      atoi1 #(r,wcv) a( vald, is_digit[i], str[i] );

      // Multiply (scale) the accumulated sum.
      uwire [wv-1:0] valns;
      mult_by_c #( .w_in(wv), .c(r), .w_out(wv) ) mc( valns, vali[i+1] );

      // Update accumulated value.
      assign vali[i] = is_digit[i] ? valns + vald : 0;
      // Update number of digits.
      assign ndi[i] = !is_digit[i] ? 0 : is_digit[i+1] ? ndi[i+1] : i + 1;

   end

endmodule
```

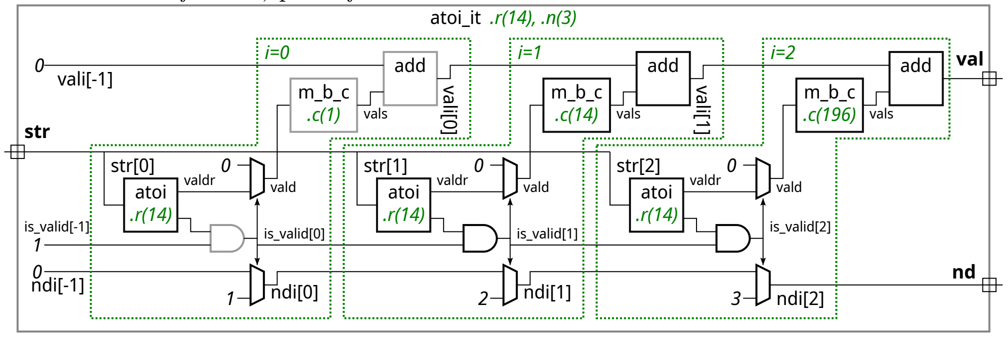
(*a*) Describe how the behavior of the module would change if the loop direction were changed as shown below, but no other changes were made.

```
   for ( genvar i=0; i<n; i++ ) begin
```

☐ Change in behavior with ascending loop:

(*b*) On the next (facing) page show the hardware that will be inferred for an instantiation of `atoi_it_m_to_l` (descending loop version) with `n=3` and `r=10`. Show each instantiation of `atoi1` and `mult_by_c` as a box, do not show their contents. The inferred hardware for `atoi_it` is shown for reference.

For reference, part of Homework 3 Problem 2 solution shown below.



For reference, part of Homework 3 Problem 2 solution shown above.

☐ Show inferred hardware for `atoi_it_m_to_l` for `n=3` and `r=10`.

☐ Show the hardware inferred for the operators, such as `&&` and `?:`.

☐ Do not confuse parameters and ports and omit hardware that does not belong, such as “hardware” to compute values needed at elaboration time.

Staple This Side

Staple This Side

(*c*) Module `atoi_m_to_l` will only show the value of numbers that are right-aligned in `str`, otherwise the value will be shown as zero. For example, for input `str="__123"` the output would be `val=123` and `nd=3`, but for input `str="_123_"` the output would be `val=0` (because the rightmost character is not a digit). Modify the module so the `val` output is the value of the number regardless of its location. If there is more than one number, say `str="__12_345_"`, show the value of the rightmost number, 345 in this case.

☐ Modify so that `val` and `nd` are for numbers whether or not they are right-aligned.

☐ Do not use procedural code.

☐ Avoid costly or slow solutions.

☐ A correct solution only requires a few changes.

Staple This Side

Staple This Side

Staple This Side

```
module atoi_it_m_to_l
  #( int r = 11, n = 5, wv = $clog2( r**n ), wd = $clog2(n+1) )
   ( output logic [wv-1:0] val,
     output logic [wd-1:0] nd,
     input uwire [7:0] str [n-1:0] );

   uwire [wv-1:0] vali[n:0];
   uwire is_digit[n:0];
   uwire [wd-1:0] ndi[n:0];
   assign is_digit[n] = 0;
   assign ndi[n] = 0;
   assign vali[n] = 0;
   assign nd = ndi[0];
   assign val = vali[0];

   localparam int wcv = $clog2(r);

   for ( genvar i=n-1; i>=0; i-- ) begin

      // Find Value of Digit i
      uwire [wcv-1:0] vald;
      atoi1 #(r,wcv) a( vald, is_digit[i], str[i] );






      // Multiply (scale) the accumulated sum.
      uwire [wv-1:0] valns;
      mult_by_c #( .w_in(wv), .c(r), .w_out(wv) ) mc( valns, vali[i+1] );






      // Update accumulated value.
      assign vali[i] = is_digit[i] ? valns + vald : 0;




      // Update number of digits.
      assign ndi[i] = !is_digit[i] ? 0 : is_digit[i+1] ? ndi[i+1] : i + 1;




   end
endmodule
```

Staple This Side

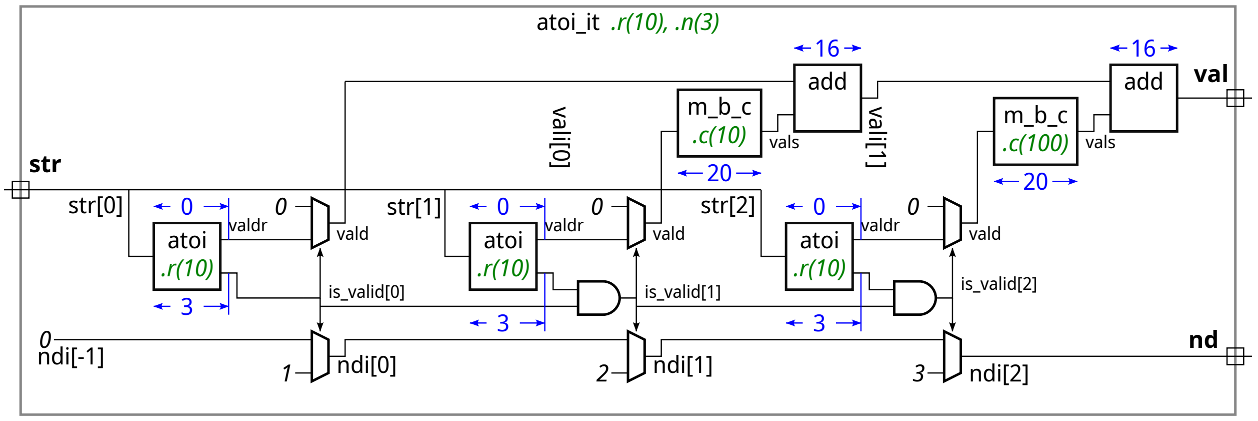
Problem 3: [20 pts] Illustrated below is the hardware for one of the `atoi` modules from Homework 3. The delays for the `add`, `atoi1`, and `mult_by_c` modules are shown in blue. For `atoi` the delay of the value (`valdr`) output is zero and the delay of the `is_digit` (lower) output is 3.

(*a*) Based on the illustrated delays and using the simple model find the delay at each output, `val` and `nd`, and show the critical path to each.

☐ Use the simple model and indicated delays to find the delay at outputs `val` and `nd`.

☐ Show the critical path to both `val` and `nd`.

☐ Take into account constant values.



(*b*) Modify the design to reduce the delay at `val` by moving multiplexors. The modification is simple though will increase cost. Show your modification either on the diagram or in the Verilog code below.

☐ Modify to reduce the delay at `val` by moving multiplexors.

☐ Do not change what the module does.

```
module atoi_it
  #( int r = 11, n = 5, wv = $clog2( r**n ), wd = $clog2(n+1) )
   ( output logic [wv-1:0] val,  output logic [wd-1:0] nd,
     input uwire [7:0] str [n-1:0] );

   uwire [wv-1:0] vali[n-1:-1];
   uwire is_valid[n-1:-1];
   uwire [wd-1:0] ndi[n-1:-1];
   assign is_valid[-1] = 1;
   assign ndi[-1] = 0;
   assign vali[-1] = 0;
   assign nd = ndi[n-1];
   assign val = vali[n-1];
   localparam int wcv = $clog2(r);

   for ( genvar i=0; i<n; i++ ) begin
      uwire [wcv-1:0] valdr;
      uwire is_digit;
      atoi1 #(r,wcv) a( valdr, is_digit, str[i] );  // Find Value of Digit i

      // Determine if this digit continues a sequence of valid digits.
      //
      assign is_valid[i] = is_digit && is_valid[i-1];



      // Replace value with zero if str[i] is not a digit, or if the
      // string of valid digits has already ended.
      //
      uwire [wcv-1:0] vald = is_valid[i] ? valdr : 0;



      // Multiply (scale) the digit value based on its position in the number.
      //
      uwire [wv-1:0] vals;
      mult_by_c #( .w_in(wcv), .c(r**i), .w_out(wv) ) mc( vals, vald );



      // Add the scaled digit to the value accumulated so far.
      //
      add #(wv) a1( vali[i], vali[i-1], vals );



      // Update the number of digits so far.
      //
      assign ndi[i] = is_valid[i] ? i+1 : ndi[i-1];

   end
endmodule
```

Problem 4: [12 pts] Answer each question below.

(*a*) The module below will not compile because of the way the module connections are declared. Fix the problem by changing the declarations.

☐ Change declaration to fix problem.

```
module yucx2
  #( int w = 5 )
   ( output uwire [w-1:0] x,
     input uwire [1:0] s,
     input uwire [w-1:0] a0, a1 );

   always_comb begin
      x = a0;
      if ( s != 0 ) x = a1;
   end

endmodule
```

(*b*) The `mv` output of `findmax` is supposed to be set to the value of the largest of the three inputs. Assuming it compiles and simulates, it still won't work. Identify the problem.

☐ Why won't `mv` be set to the maximum of `a0`, `a1`, `a2`?

☐ Provide an example that illustrates the incorrect behavior.

```
module findmax
  #( int w = 5 )
   ( output logic [w-1:0] mv,
     input uwire [w-1:0] a0, a1, a2 );

   initial mv = 0;
   always_comb if ( mv < a0 ) mv = a0;
   always_comb if ( mv < a1 ) mv = a1;
   always_comb if ( mv < a2 ) mv = a2;

endmodule
```

Staple This Side

Problem 5: [12 pts] Answer each question below.

(*a*) Type `logic` is an example of a four-state type. Name those four states and describe what the non-numeric ones are used for.

☐ Name the four `logic` states.

☐ Describe what the non-numeric ones signify.

(*b*) Most synthesis programs will not synthesize a module that includes a delay, such as the one below. Why not?

```
module madd
  #( int w )
   ( output logic [w-1:0] w,
     input uwire [w-1:0] a, b, c );
   always_comb begin
      w = a * b;
      #5; // Allow enough time for multiplication to finish.
      w = w + a;
   end
endmodule
```

☐ Why isn't a delay synthesizeable?

Staple This Side