Name Solution

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)

Alias Sentient?

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.

The solution appears below. The first two muxen, `m01` and `m23`, connect to the data inputs (`a0`-`a3`), two per mux. Note that both of these muxen use `s[0]` as the select bit. The select connection of the third mux, `m0123`, connects to bit `s[1]`.

```
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 );

   // SOLUTION
   //
   uwire [w-1:0] a01, a23;
   mux2 #(w) m01(   a01, s[0], a0,  a1 );
   mux2 #(w) m23(   a23, s[0], a2,  a3 );
   mux2 #(w) m0123( x,   s[1], a01, a23 );

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=` 2

☑ In `mux2_bad` when `w=4`, `s=0`, `a0=2`, and `a1=4`, ☑ output `x=` 1

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

The problem is that `a0` and `a1` are operands of a logical AND operator, `&&`, and so they will be converted to a Boolean (1-bit) type. That changes both the 2 and 4 in the example to a 1. There would be no problem if `a0` and `a1` were already 1 bit.

(*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`.

The solution is shown below. Note that instance `m1` was declared with `w=1` and `mr` was declared with `w=w-1` as part of the problem. So to complete the module instance `m1` connects with one bit of each of `x`, `a0`, and `a1`. Here bit zero was chosen but any bit position would do. Instance `mr` connects to the remaining `w-1` bits of `x`, `a0`, and `a1`. The select signal is the same for both instances.

Note that there is no practical reason to recursively describe a 2-input multiplexor this way, or to recursively describe a 2-input multiplexor at all.

```
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


      // SOLUTION
      mux2_1r #(1)   m1( x[0],     s, a0[0],     a1[0] );
      mux2_1r #(w-1) mr( x[w-1:1], s, a0[w-1:1], a1[w-1:1] );


   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
```

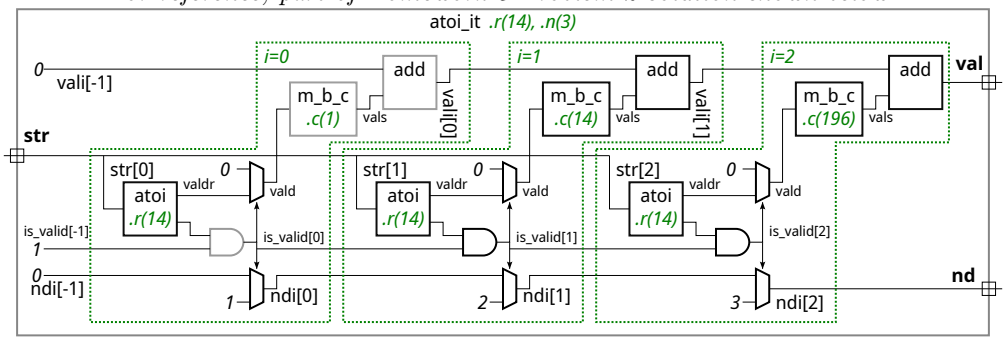
☑ Change in behavior with ascending loop:

There will be no change in behavior. It may be more confusing to a human with the direction of the loop reversed, but the module does exactly the same thing. To see that look at the line assigning `ndi[i]`. It is computed using `ndi[i+1]`. In a procedural language the forward loop would not work because `ndi[i+1]` would not have been computed at iteration `i` when `ndi[i]` is written. But this is Verilog and `assign` is a *continuous assignment* that re-executes whenever its live-in values change, `is_digit[i]`, `is_digit[i+1]`, and `ndi[i+1]` in this case. All the generate loop is doing is describing hardware, each iteration describes one set of hardware. When the hardware for `assign ndi` from iteration `x+1` executes it writes `ndi[x+1]` which results in the `assign`

`ndi` for iteration `x` to execute because `ndi[x+1]` is in the sensitivity list for the assign.

(*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.



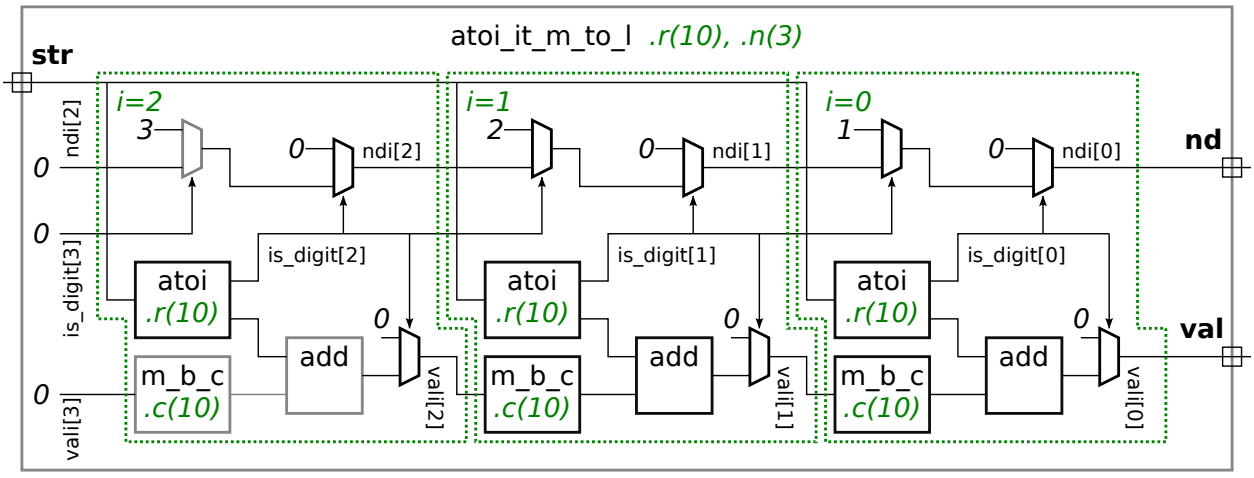
For reference, part of Homework 3 Problem 2 solution shown above.

☑ 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.

Solution appears below. Hardware that can easily be eliminated by optimization appears in gray.



(*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.

Solution appears in the Verilog on the next page.

In the original code, if `is_digit[i]` was false then the value and length were set to zero. But now since there can be non-digit characters to the right of the number we can't set these to zero. So the first case in the expressions assigning `vali[i]` and `ndi[i]` pass the value and length along when `is_digit[i]` is false.

If both `is_digit[i]` and `is_digit[i+1]` are true then a number is continuing at position `i`. For `vali[i]` we need to add on the scaled number from the left (`valns`) and the current digit, `vald`. If `is_digit[i]` is true but `is_digit[i+1]` is false then `vali` is just the value of the current digit, `vald`. Unlike in the original hardware we can't rely on `valns` being zero for this case.

In the original hardware the value of `i+1` was used for `ndi[i]` at the left-most digit. That won't work here because there could be non-digit characters to the right of the number, so we can't use the position of the first non-digit character to compute the length. Instead, when a number is continuing, both `is_digit[i]` and `is_digit[i+1]` are true, the hardware adds `1` to the previous value of the length (`ndi[i+1]`).

```
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;
      /// SOLUTION
      assign vali[i] =
              !is_digit[i]   ? vali[i+1] :
               is_digit[i+1] ? valns + vald : vald;



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

   end
endmodule
```

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.

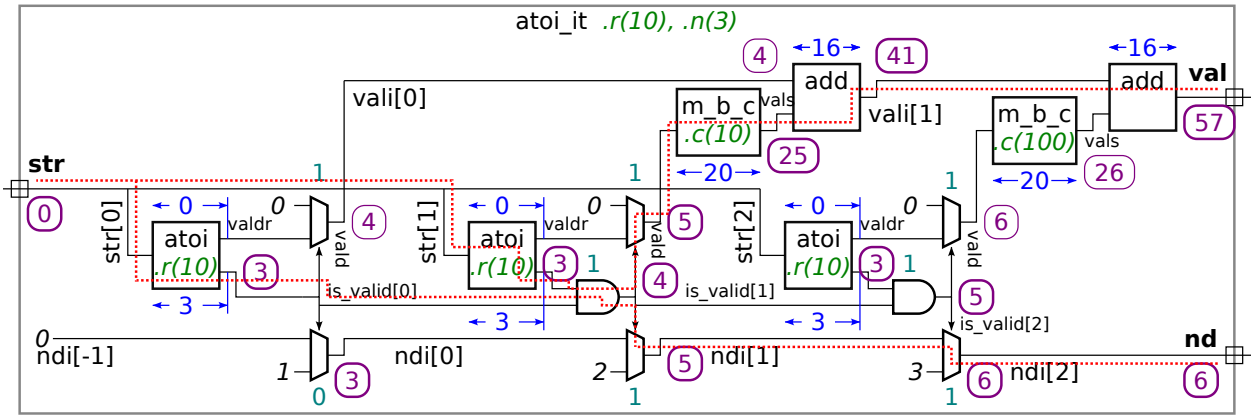
(*a*) Based on the illustrated delays and using the simple model find the delay at each output, `val` and `nd`, and show the critical path to each.

☑ 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.

Solution appears below. Note that the delay of a 2-input mux with one constant input is 1, and the delay with two constant inputs is zero.

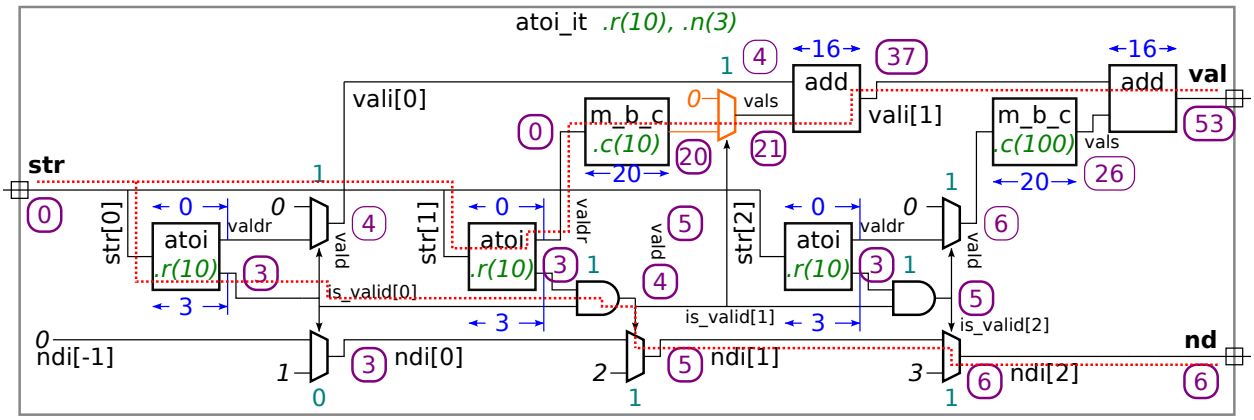


(*b*) Modify the design to reduce the delay at `val` by moving multiplexors. The modification is simple though will increase cost. Show your modification either on the diagram or in the Verilog code below.

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

☑ Do not change what the module does.

The solution appears below, with the moved mux shown in orange. By moving the mux to the output of the `m_b_c` module it can start at $t = 0$ rather than waiting for the mux select signal to arrive.



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.

The solution appears below. Since `x` is assigned proceduraly it must be declared `logic`, which make it a `var` kind rather than a net kind.

```
module yucx2
  #( int w = 5 )
   ( output logic [w-1:0] x,  // SOLUTION: Change port from uwire to logic.
     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`?

Because `mv` is only initialized once, at the beginning of simulation whereas `a0`, `a1`, and `a2` can change any time.

☑ Provide an example that illustrates the incorrect behavior.

At $t = 10$ the inputs are `a0=4`, `a1=7`, `a2=3`. The output will be `mv=7`. Later at $t = 10$ inputs are `a0=3`, `a1=2`, `a2=0`. The output will still be `mv=7` because there is no way for `mv` to be set to a smaller value.

```
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
```

```
module findmax
  #( int w = 5 )
   ( output logic [w-1:0] mv,  input uwire [w-1:0] a0, a1, a2 );
   always_comb begin           // SOLUTION: Possible fix. (Not the best.)
     mv = 0;                   // mv is initialized whenever the a's change.
     if ( mv < a0 ) mv = a0;
     if ( mv < a1 ) mv = a1;
     if ( mv < a2 ) mv = a2;
   end
endmodule
```

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.

They are `0`, `1`, `x`, and `z`.

☑ Describe what the non-numeric ones signify.

State `x` for `var` types can mean uninitialized. For both `var` and `uwire` it can mean an ambiguous results. For net kinds (such as `uwire`) it can mean a bit is driven by more than one driver. State `z` for net types means it is not being driven (in a high impedance state).

(*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?

Though it would be possible for a synthesis program and technology target to provide for delays, it would not be very useful, especially in digital logic design. In the module above the output of the multiplier connects to the input of the adder. A delay has no role to play, since the inferred hardware is just a bunch of connected gates. There is no way to, and no need to, tell the gates that their input values have arrived and so now its time to start working.