LSU EE 4755 Homework 1 Solution Due: 9 September 2015

The questions below can be answered without using EDA software, paper and pencil will suffice. Please turn in the solution on paper. Homework 2 will require the use of Verilog implementations.

Those who are rusty about the correspondence between Verilog code and hardware might want to look at the solution to EE 3755 Fall 2013 Homework 1, at `http://www.ece.lsu.edu/ee3755/2013f/hw01_sol.pdf`.

Problem 1: The routine `shift_right_fixed_amt` uses the `>>` operator to perform the right shift. Perhaps you are wondering if the operation is an arithmetic right shift or a logical right shift. (In a logical right shift the vacated bit positions are always set to zero, in an arithmetic shift they are set to the MSB of the input.) Look up the operation performed by this operator in the SystemVerilog 2012 documentation.

```
module shift_right_fixed_amt
  #( int fsamt = 4 )   // Fixed shift amount.
   ( output wire [15:0] shifted,
     input wire [15:0] unshifted,
     input wire shift );

   // If shift is true shift by fsamt, otherwise don't shift.
   //
   assign    shifted =  shift  ?  unshifted >> fsamt  :  unshifted;

endmodule
```

(*a*) Indicate the section and page in which this information can be found.

Section 11.4.10, on page 233.

(*b*) Show how the module can be modified to perform the other kind of shift (if it's currently arithmetic, make it logical, if it's currently logical make it arithmetic).

Two changes need to be made: The type of the value to be shifted must be changed to signed, and the operator must be changed from >> to >>>. The changed code appears below.

```
module shift_right_fixed_amt_sol
  #( int fsamt = 4 )   // Fixed shift amount.
   ( output wire [15:0] shifted,
     input wire signed [15:0] unshifted,  // SOLUTION, change to signed.
     input wire shift );

   // SOLUTION, change ">>" operator to ">>>".
   //
   assign    shifted =  shift  ?  unshifted >>> fsamt  :  unshifted;

endmodule
```

Problem 2: Appearing below are two variations on a `min_4` module that finds the minimum of four unsigned integers. Both of these modules instantiate the following `min_2` module.

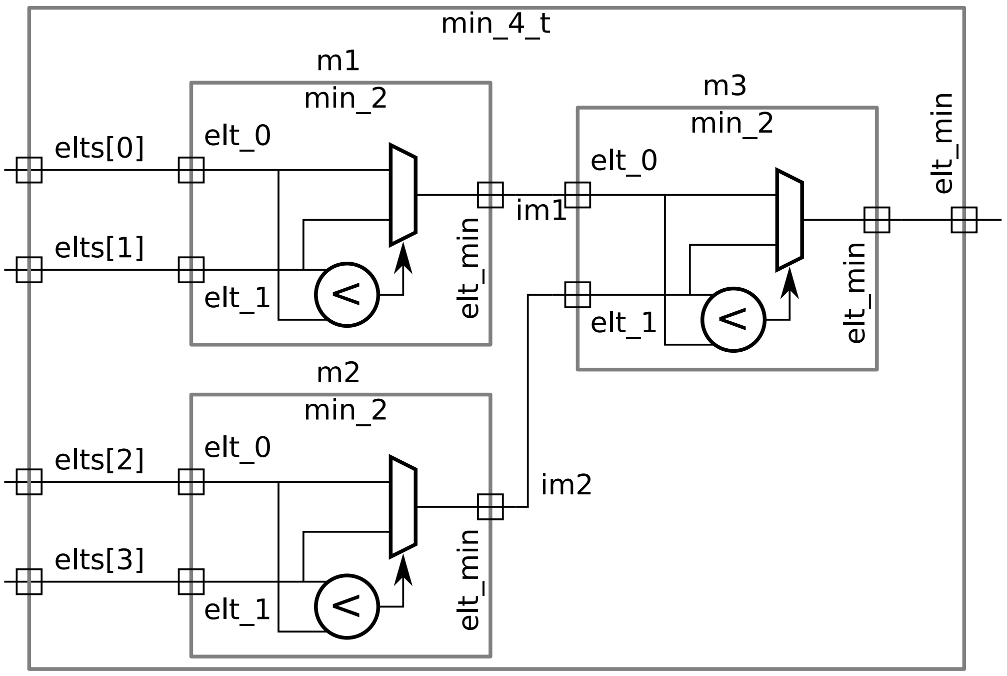
```
module min_2
  #( int elt_bits = 4 )
   ( output [elt_bits-1:0] elt_min,
     input [elt_bits-1:0] elt_0,
     input [elt_bits-1:0] elt_1 );
   assign                 elt_min = elt_0 < elt_1 ? elt_0 : elt_1;
endmodule
```

(*a*) Draw a diagram of the hardware that will be synthesized for the `min_4_t` module below. Your diagram should include two-input multiplexors and a comparison module. To get an idea of what to draw, see the EE 3755 Homework solution mentioned at the top of this assignment.

```
module min_4_t
  #( int elt_bits = 4 )
   ( output [elt_bits-1:0] elt_min,
     input [elt_bits-1:0] elts [4] );

   wire [elt_bits-1:0]    im1, im2;
   min_2 #(elt_bits) m1( im1, elts[0], elts[1] );
   min_2 #(elt_bits) m2( im2, elts[2], elts[3] );
   min_2 #(elt_bits) m3( elt_min, im1, im2 );
endmodule
```

Solution appears below.

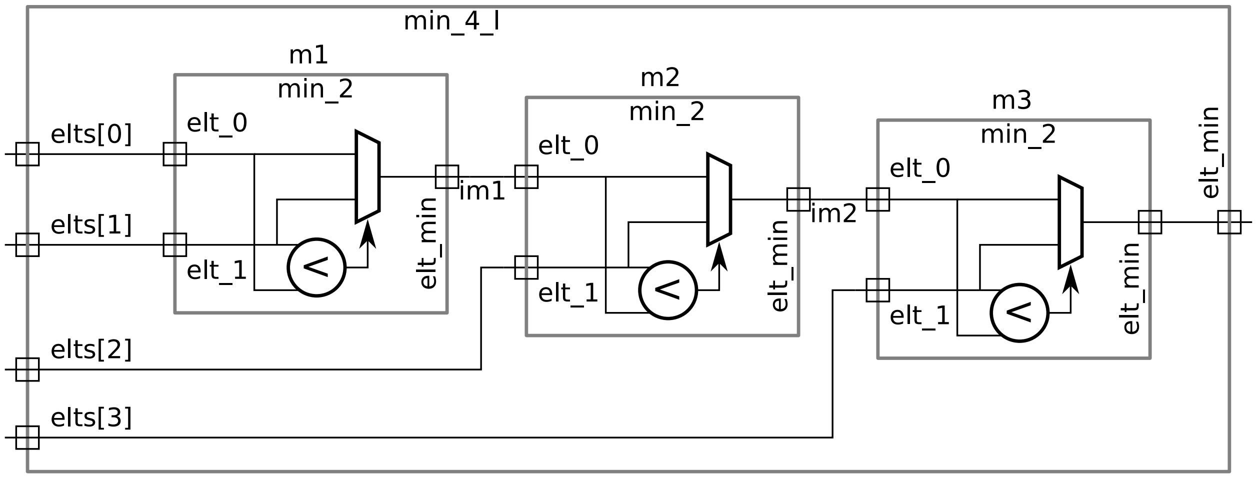


(*b*) Draw a diagram of the hardware that will be synthesized for the `min_4_l` module below. Your diagram should include two-input multiplexors and a comparison module.

```
module min_4_l
  #( int elt_bits = 4 )
   ( output [elt_bits-1:0] elt_min,
     input [elt_bits-1:0] elts [4] );

   wire [elt_bits-1:0]    im1, im2;
   min_2 #(elt_bits) m1( im1, elts[0], elts[1] );
   min_2 #(elt_bits) m2( im2, im1, elts[2] );
   min_2 #(elt_bits) m3( elt_min, im2, elts[3] );
endmodule
```

Solution appears below.



(*c*) Which of the two modules above would you expect to have lower cost? Which would you expect to be faster? Briefly explain.

The cost of the two modules should be the same. Module `min_4_t` should be faster because the longest path through the module is through two `min_2` modules, whereas in `min_4_l` the longest paths is through three `min_2` modules.

Problem 3: The module `min_4_err` below is correct Verilog, but it won't do what we want.

```
module min_4_err
  #( int elt_bits = 4 )
   ( output [elt_bits-1:0] elt_min,
     input [elt_bits-1:0] elts [4] );

   wire [elt_bits-1:0]    im;
   min_2 #(elt_bits) m1( im, elts[0], elts[1] );
   min_2 #(elt_bits) m2( im, im, elts[2] );
   min_2 #(elt_bits) m3( elt_min, im, elts[3] );

endmodule
```

(*a*) Explain why it's correct Verilog yet provides the incorrect result.

The problem is that the output of `m1` and `m2` are both connected to the same net, `im`. This may lead to conflicts, for example, when `m1` wants to set bit `im[0]` to 1 but `m2` wants to set it to 0. The simulator will assign an `x` for such cases. Worse, in `m2` an output and an input are connected to the same loop.

(*b*) Look up `uwire` in the SystemVerilog standard and explain how that might help catching such errors.

Unlike a net of type `wire`, a net of type `uwire` can only be driven by one source. See IEEE Std 1800-2012 Section 6.6.2. A net connected in the same way as `im`, above, would result in a Verilog compiler error.

Problem 4: Appearing below is yet another variation on `min_4`, this one attempting to take advantage of a special case by using generate statements. The module is correctly using generate statements to handle a special case. Do you think the synthesized hardware will be less expensive for the special case *beyond the reduction in cost for using fewer bits.* Hint: Think about what the comparison unit and mux would look like with 1-bit inputs and how such logic can be optimized.

Note: In the original assignment this problem had a typo, which made the Verilog illegal. Further, the phrase above starting "beyond the reduction" was not in the original question, making it difficult to see what was really being asked. The answer below is for the corrected question.

```
module min_4_special1
  #( int elt_bits = 4 )
   ( output [elt_bits-1:0] elt_min,
     input [elt_bits-1:0] elts [4] );

   if ( elt_bits == 1 ) begin

      assign elt_min = elts[0] && elts[1] && elts[2] && elts[3];

   end else begin

      wire [elt_bits-1:0]    im1, im2;

      min_2 #(elt_bits) m1( im1, elts[0], elts[1] );
      min_2 #(elt_bits) m2( im2, im1, elts[2] );
      min_2 #(elt_bits) m3( elt_min, im2, elts[3] );

   end

endmodule
```

The special case is, of course, and AND gate and we expect that the synthesis program can easily handle those. When `elt_bits` is greater than one the synthesis program sees a linear connection of `min_2` modules

Problem 5: The module below handles another special case, in this case the case where the first element is zero.

```
module min_4_special2
  #( int elt_bits = 4 )
   ( output [elt_bits-1:0] elt_min,
     input [elt_bits-1:0] elts [4] );

   wire [elt_bits-1:0]    im1, im2;

   if ( elts[0] == 0 )
     assign elt_min = 0;
   else begin
      min_2 #(elt_bits) m1( im1, elts[0], elts[1] );
      min_2 #(elt_bits) m2( im2, im1, elts[2] );
      min_2 #(elt_bits) m3( elt_min, im2, elts[3] );
   end
endmodule
```

(*a*) Explain why the module is illegal Verilog.

The `if` statement, testing `elts[0]`, is not in procedural code (for example, in an `initial` or `always`), and so it will be interpreted as a generate statement. Generate statements can only access elaborate-time constants, such as parameters and variables declared `genvar`. A module input port, such as `elts`, is definitely not such a constant and so there is an error.

(*b*) Explain why what it's trying to do would be unlikely to help within a larger design. *Hint: Think about critical path.*

Suppose that the delay through `min_4_special2` when `elts[0]==0` is $1\,\mathrm{ns}$ and is $3\,\mathrm{ns}$ in other cases. Suppose that the output of `min_4_special2` is connected to logic that has another $5\,\mathrm{ns}$ of delay. Setting a clock period to $1 + 5 = 6\,\mathrm{ns}$ would result in errors when the special case was not present and setting it to $3 + 5 = 8\,\mathrm{ns}$ would make the special-case hardware unnecessary.

It's not impossible to take advantage of the special case. To do so external logic would need to detect it (an output indicating the special case could be added to `min_4_special2`) and there would have to be some advantage for the special case. One possibility is that for the special case results from the external logic would be captured in one cycle, otherwise it would take two cycles.