

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.
- (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).

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
```

(b) Draw a diagram of the hardware that will be synthesized for the `min_4_1` module below. Your diagram should include two-input multiplexors and a comparison module.

```
module min_4_1
  #( 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
```

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

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.

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

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

end

endmodule
```

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
end
endmodule
```

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