LSU EE 4755 **Homework 3** **Due: 7 October 2015**

Problem 1: Solve EE 4755 Fall 2014 Midterm Exam Problem 4 and Problem 5. The solutions are available, but please make an honest effort to solve them on your own.

Problem 2: The homework Verilog file, `hw04.v` contains two versions of the sequential shifter used in class, those modules are also reproduced below. Module `shift_lt_seq_d_live`, is based on the version written during class and module `shift_lt_seq_d` is the one prepared in advance. Though both work correctly their timing is not identical.

(*a*) Show the hardware that might be synthesized for each module using the default parameters. Include reasonable optimizations, the initially inferred hardware can be omitted. This should be a human-to-human diagram, don't show output of a synthesis program.

(*b*) The two modules differ in their timing. Using your hardware diagrams explain any differences in:

- The register-to-register delay within the module.
- How far in advance of the positive edge module inputs must become stable.
- How long after the positive edge module outputs will be available.

As with the previous part, this should be done by hand though synthesis tools can be used to help solve the problem.

An answer might look like this: *"For register-to-register delay Module A is slower because its critical path has two multipliers, whereas in module B the two multiplications are split between cycles and so at most one multiplier is on the critical path. In module A inputs connect directly to a divider, and so they must arrive long before the positive edge, whereas in module B inputs can arrive just before the positive edge because ...."* Of course, this question does not have a module A or B, nor does it really have multipliers and dividers.

Modules on next page.

```
module shift_lt_seq_d_live
  #( int wid_lg = 6,
     int num_shifters = 1,
     int wid = 1 << wid_lg )
   ( output logic [wid-1:0] shifted,
     output logic ready,
     input [wid-1:0] unshifted,
     input [wid_lg-1:0] amt,
     input start,
     input clk );

   localparam int bits_per_seg = wid_lg / num_shifters;

   logic [num_shifters-1:0] shift;
   wire [wid-1:0] shin[num_shifters-1:-1];
   assign shin[-1] = shifted;

   for ( genvar i=0; i<num_shifters; i++ ) begin
      localparam int fs_amt = 2 ** ( i * bits_per_seg );
      shift_fixed #( wid_lg, fs_amt ) sf( shin[i], shin[i-1], shift[i] );
   end

   logic [num_shifters-1:0][bits_per_seg-1:0] cnt;

   always_ff @( posedge clk ) begin

      if ( start == 1 ) begin
         ready = 0;
         cnt = amt;
         shift = 0;
         shifted = unshifted;
      end else begin
         if ( cnt == 0 ) ready = 1;
         for ( int i=0; i<num_shifters; i++ ) begin
            shift[i] = cnt[i] > 0;
            if ( cnt[i] != 0 ) cnt[i]--;
         end
         shifted = shin[num_shifters-1];
      end

   end

endmodule
```

Another module on next page.

```
module shift_lt_seq_d
  #( int wid_lg = 4,
     int num_shifters = 2,
     int wid = 1 << wid_lg )
   ( output logic [wid-1:0] shifted,
     output wire ready,
     input [wid-1:0] unshifted,
     input [wid_lg-1:0] amt,
     input start,
     input clk );

   localparam int cnt_bits = ( wid_lg + num_shifters - 1 ) / num_shifters;
   logic [num_shifters-1:0][cnt_bits-1:0] cnt;
   wire [wid-1:0] inter_sh[num_shifters-1:-1];
   assign inter_sh[-1] = shifted;

   for ( genvar i = 0; i < num_shifters; i++ ) begin
      localparam int shift_amt = 1 << i * cnt_bits;
      wire       shift = cnt[i] != 0;
      shift_fixed #(wid_lg,shift_amt) sf( inter_sh[i], inter_sh[i-1], shift );
   end

   always_ff @( posedge clk )

      if ( start == 1 ) begin
         shifted = unshifted;
         cnt = amt;
      end else if ( cnt > 0 ) begin
         shifted = inter_sh[num_shifters-1];
         for ( int i=0; i<num_shifters; i++ ) if ( cnt[i] ) cnt[i]--;
      end

   assign ready = cnt == 0;

endmodule
```