LSU EE 4755 **Homework 3** Solution **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.

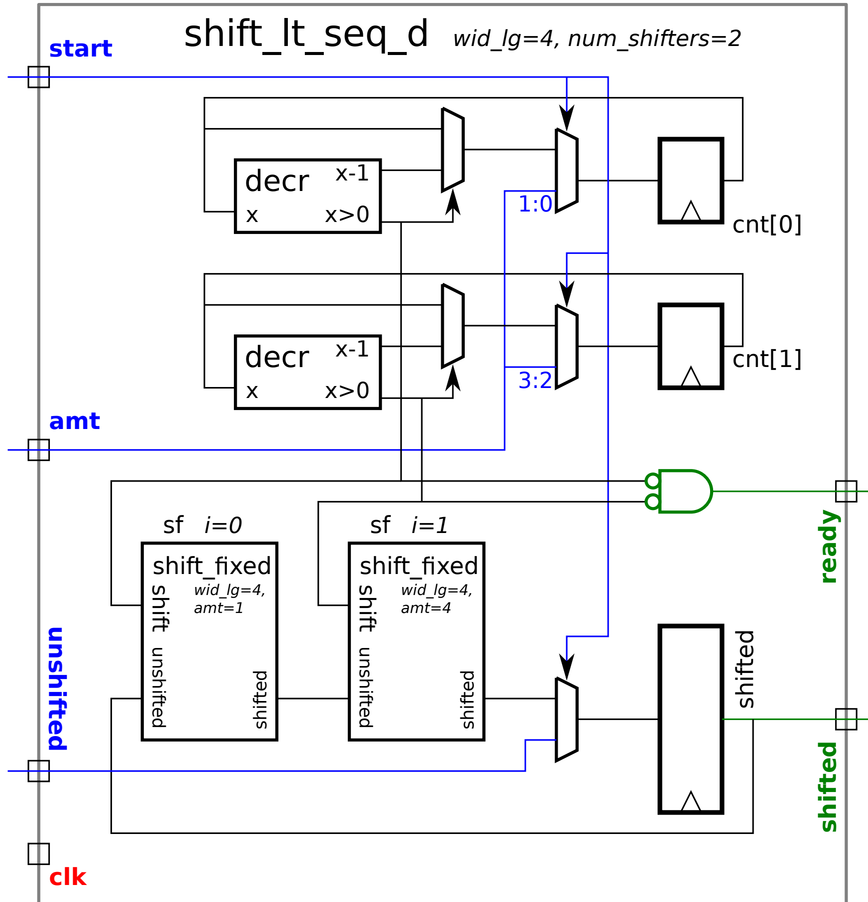
See the posted solutions at `http://www.ece.lsu.edu/koppel/v/2014/mt_sol.pdf`.

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.

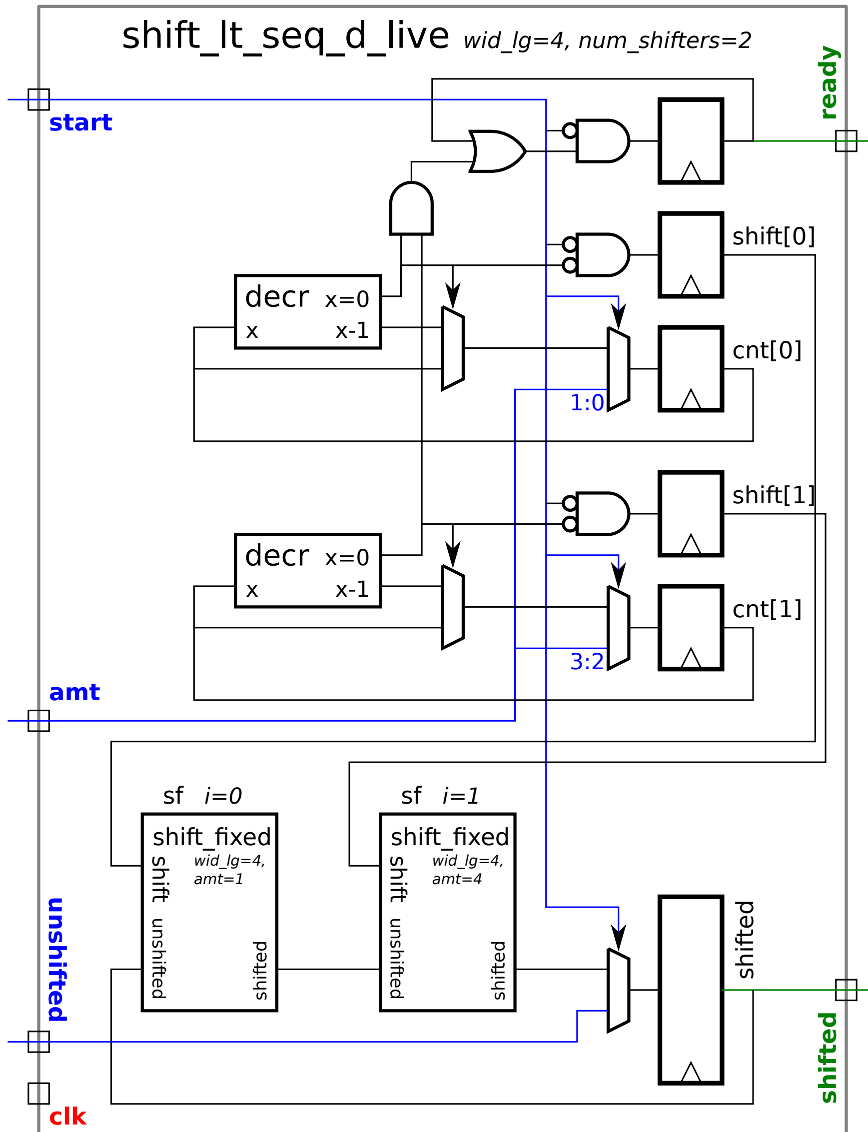
(*a*) Show the hardware that might be synthesized for each module using the default parameters. Include reasonable optimizations, the initially inferred hardware can be omitted. This should be a human-to-human diagram, don't show the output of a synthesis program.

Note: In the original assignment the parameters for the `shift_lt_seq_d_live` *module were not set as intended, that has been corrected in this version of the homework assignment. Both solutions appear below, they are referred to as the* original *and* intended *module. In the intended assignment (this one) both modules have the same parameters, in the original assignment the live module had just one shifter and could shift more bits.*

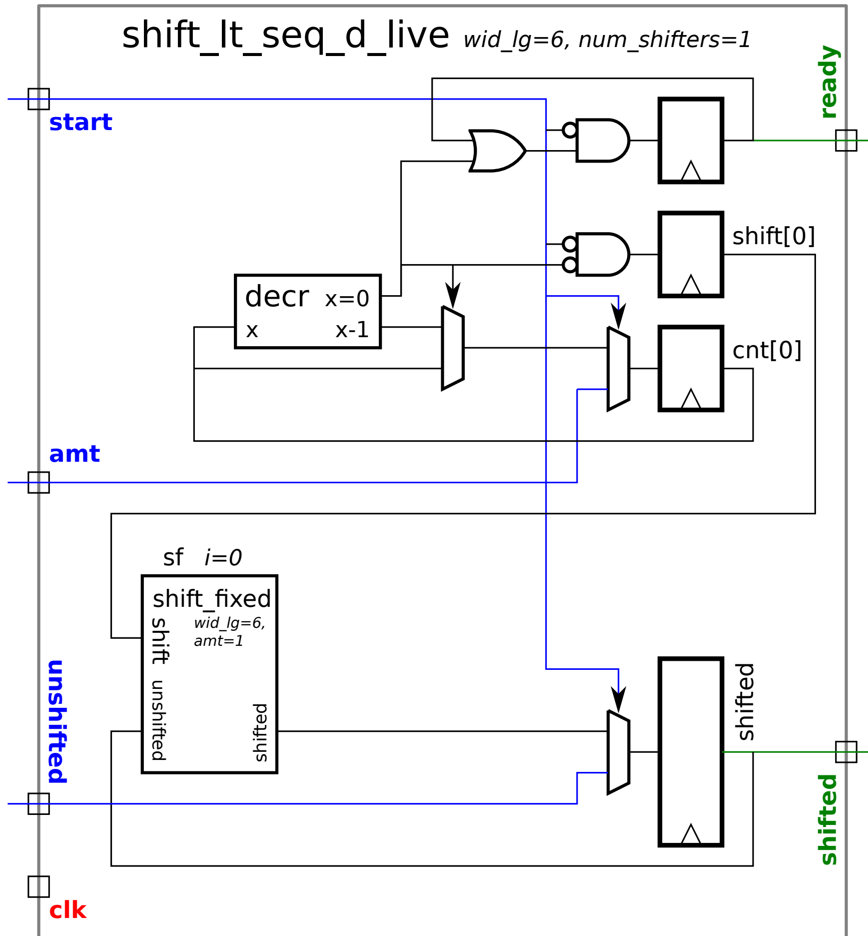
The hardware appears below. In **shift_lt_seq_d_live** the initially inferred multiplexors at the inputs to the **ready** and **shift** registers have been replaced by logic gates. The logic computing the next state of **ready** includes the old value of **ready**. The old value of **ready** isn't really needed, but it's shown because it is probably what the synthesis program would have included.



The intended live module appears below:



The original live module appears below:



(*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.

The following timing will be assumed when comparing the modules. Multiplexor delay is two gate delays from either the select or data inputs. For a two-bit decrementor the `x=0`, `x>0`, and `x-1` outputs are all 1 gate delay (draw a truth table). A six-bit decrementor is assumed to take two gate delays to compute `x=0` and 6 gate delays to compute `x-1`. Since it's essentially a multiplexor the `shift_fixed` modules take two gate delays regardless of the shift amount.

An important difference between the live and prepared module, is that the in the live module the `shift` input to `shift_fixed` comes from a register output, and so it will be available at the beginning of a the clock cycle. In the prepared module the `shift` input is generated by checking if a portion of `cnt` is zero, the check adds a small delay. Though this may sound like a small advantage for the live module, but it may not be because it doesn't use the `shift` signal until the next clock cycle and so it takes one clock cycle longer to perform the shift. If `wid_lg/num_shifters` is large than the extra clock cycle will be a small fraction of the total time and so the live module would be better. If the ratio is small the extra clock cycle will make things slower.

For the assigned problem, in which `shift_lt_seq_d_live` has 1 shifter, the register-to-register critical path in the live module is 10 gate delays, assuming 6 gate delays for the 6-bit subtract. The prepared module, `shift_lt_seq_d`, module has a critical path of 7 delays. Thus, the live module can have a higher clock frequency—that's the good news—but it will take $\frac{2^6}{2^{4/2}-1} = 21.33$ times as many cycles to perform the largest shift.

A concise answer to the assigned problem might be: the register-to-register delay in the live module is much longer because it must decrement a much larger number, six versus two bits. This overcomes any benefit of having one shifter, versus two in the prepared module.

In the intended problem the live module has the same parameters as the prepared module, including two shifters. In that case the critical path is 6 gate delays, 1 gate delay faster than the prepared module. But because it takes one cycle longer the benefit in clock frequency would not be large enough to overcome the disadvantage of requiring one more clock cycle, at least not for the default parameters.

The two modules have equivalent input setup times, two gate delays. So for both, the inputs can arrive near the end of the clock cycle.

In the live module the outputs are available at the beginning of the clock cycle. In the prepared module the `ready` signal is generated using an AND gate connected to the decrementors. Based on the analysis above, the prepared module's ready output is not available until two gate delays after the clock edge.

Modules on next page.

```
module shift_lt_seq_d_live
  #( int wid_lg = 4,        // In original assignment, 6
     int num_shifters = 2,  // In original assignment, 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
```