Name: KEY **Id#**

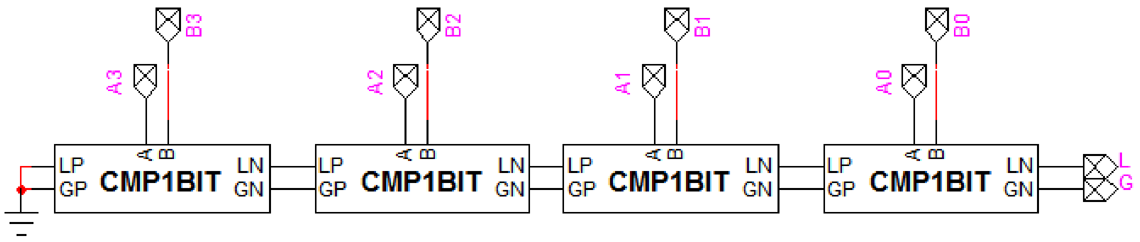
COE 405, Term 131

Design & Modeling of Digital Systems

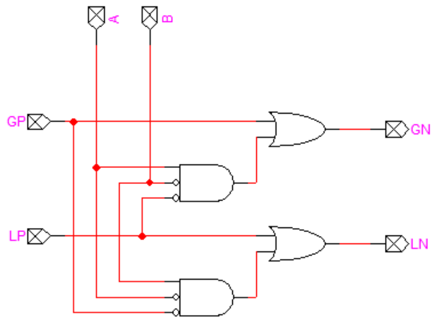
Quiz# 3

Date: Thursday, Nov. 28, 2013

Q.1. It is required to model an n-bit iterative magnitude comparator. A 4-bit comparator is shown below:



The model for a 1-bit comparator is as follows:



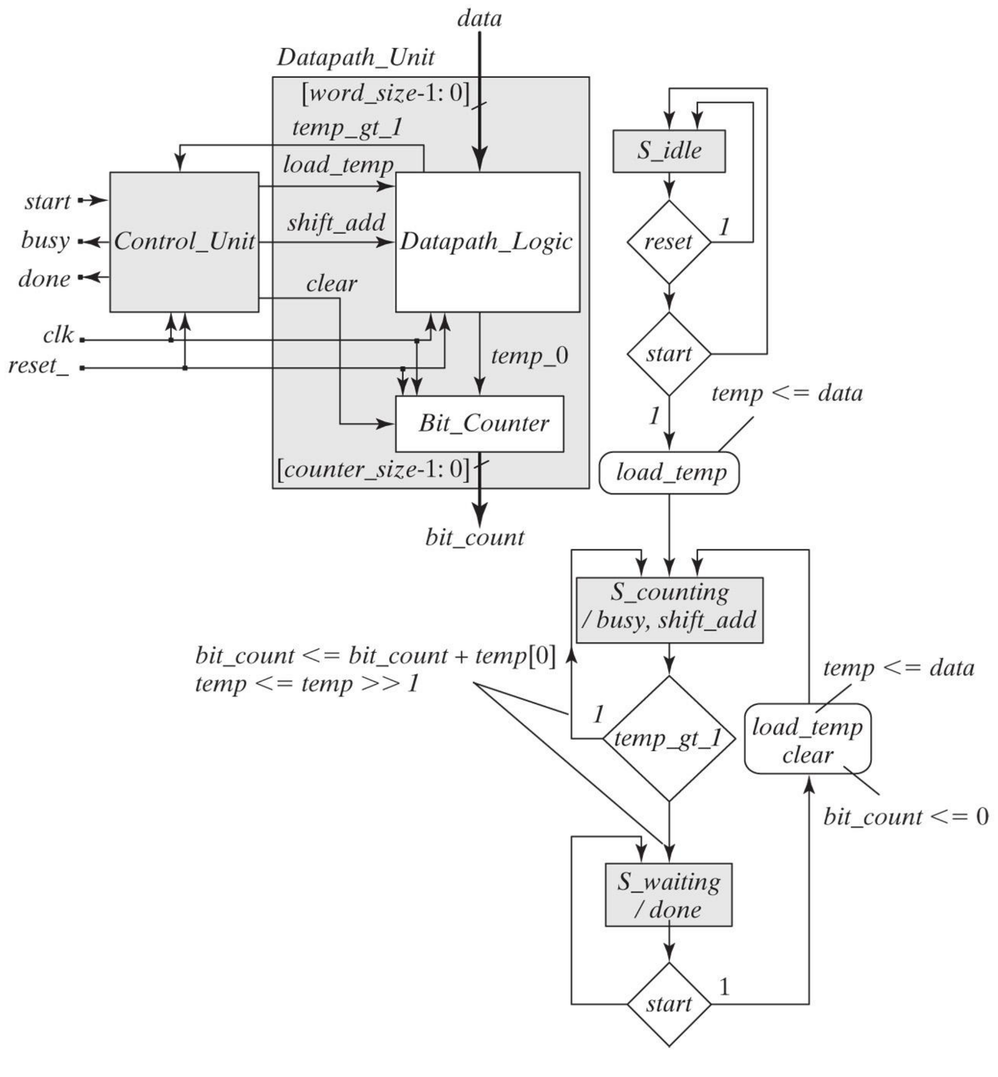
Write a parametrizable Verilog model for modeling an n-bit comparator with a default of n=4.

```
module comparator #(parameter n=4)(
output  reg  G, L,
input [n-1:0] a, b);
integer k;

always @ (a, b) begin: compare_loop

G = 0; L = 0;
   for (k=n-1; k>=0; k=k-1) begin
          G = G | a[k] & ~b[k] & ~L;
          L = L | ~a[k] & b[k] & ~G;
   end

end
endmodule
```

Q.2. The ASMD chart given below describes a state machine that counts 1's in a word and terminates activity as soon as possible. The machine remains in its reset state, *S_idle*, until an external agent asserts *start*. This action asserts the output, *load_temp*, which will cause *data* to be loaded into register *temp* when the state makes a transition to *S_counting* at the next active edge of *clk*. The machine remains in *S_counting* while *temp* contains a 1. Two actions occur concurrently at each subsequent clock: (1) *temp* is shifted towards its LSB and (2) *temp[0]* is added to *bit_count*. When *temp* finally has a 1 in only the LSB, the machine's state moves to *S_waiting*, where *done* is asserted. The state remains in *S_waiting* until *start* is reasserted. Assume that when the synchrnous *reset* input is asserted the machine is reset to the state *S_idle* and *bit_count* and *temp* are initialized to 0.

(i) Write a Verilog model for modeling the data-path unit.

(ii) Write a Verilog model for modeling the control unit using the following state assignment: *S_idle=00*, *S_counting =01*, and *S_waiting=10*.



(i)

```
module OnesCount_DPU #(parameter word_size=4, counter_size=3)(
output reg [counter_size-1: 0] bit_count,
output temp_gt_1,
input [word_size-1:0] data,
input load_temp, shift_add, clear, clock, reset
);

reg [word_size-1:0] temp;

assign temp_gt_1 = | temp[word_size-1:1];

always @ (posedge clock)
  if (load_temp) temp <= data;
  else if (shift_add) temp <= temp >> 1;

always @ (posedge clock)
  if (reset || clear) bit_count <= 0;
  else if (shift_add]) bit_count <= bit_count + temp[0];

endmodule
```

(ii)

```
module OnesCount_CU  (output reg load_temp,
shift_add, clear, done, busy,
input start, temp_gt_1, clock, reset);

parameter S_idle = 2'b00, S_counting = 2'b01, S_waiting = 2'b10;

reg [1:0] state, next_state;

  always @(posedge clock)
   if (reset) state <= S_idle;
   else state <= next_state;

  always @(state, start, temp_gt_1) begin
   load_temp=0; shift_add=0; clear=0; done=0; busy=0;

   case (state)
      S_idle:
              if (start) begin
                load_temp = 1;
                next_state = S_counting;
              end
              else next_state = S_idle;
      S_counting: begin
                  shift_add = 1; busy = 1;
                  if (temp_gt_1) next_state = S_counting;
                  else   next_state = S_waiting;
            end
      S_waiting:  begin
                done = 1;
                if (start) begin
                          load_temp = 1;  clear = 1;
                          next_state = S_counting;
                end
                else next_state = S_waiting;
               end
      default: begin
            next_state = 2'bxx;
            load_temp='bx; shift_add='bx; clear='bx; done='bx; busy='bx;
          end
    endcase
  end
endmodule
```