Lab 9 Nov 5,6,7 Fall 2018

DSD EE3563

Name: Andrew Quick

1. What are the limits on how fast the comparator works? How could the design be modified to make it work faster?

   Propagation delay limits the comparator. Limiting the number of gates increases the speed.

2. Could you extend the design to more bits?

   Yes

3. Note: that if A=1001 and B=1010 what are the results of these various tests:
   If (A && B) = **T** or F ; if(A&B) = **T** or F ;

   Now with A=0000 and B=0000:
   If(A == B) = **T** or F; if (A=B) = T or **F**

4. How many bits of information must be sent from one module to the next of the 1 bit comparator module, how many wires does this take?

   2 inputs and 3 outputs, so at least 3 wires in between each 1-bit comparator.

5. List 2 different ways you could write the Verilog code: List on at the behavioral level and one at the Register Transfer Level (RTL) with the logic equations you developed.

```
module comparator_behavioral (
    input wire A,       // 1-bit input A
    input wire B,       // 1-bit input B
    output reg A_greater, // A > B
    output reg A_less,    // A < B
    output reg A_equal    // A == B
);

always @(*) begin
    // Comparison logic
    if (A > B) begin
        A_greater = 1;
        A_less = 0;
        A_equal = 0;
    end
    else if (A < B) begin
```

```
        A_greater = 0;
        A_less = 1;
        A_equal = 0;
    end
    else begin
        A_greater = 0;
        A_less = 0;
        A_equal = 1;
    end
end

endmodule
```

```
module comparator_rtl (
    input wire A,       // 1-bit input A
    input wire B,       // 1-bit input B
    output wire A_greater, // A > B
    output wire A_less,    // A < B
    output wire A_equal    // A == B
);

// Logic equations for comparison
assign A_greater = A & ~B;    // A is greater if A is 1 and B is 0
assign A_less    = ~A & B;    // A is less if A is 0 and B is 1
assign A_equal   = ~(A ^ B);  // A equals B if A and B are the same

endmodule
```

6. List the truth table for your 1 bit comparator. How many outputs must connect from one cell to the next? How many different outcomes are there for each comparison?

A	**B**	**A > B**	**A < B**	**A == B**
0	**0**	**0**	**0**	**1**
0	**1**	**0**	**1**	**0**
1	**0**	**1**	**0**	**0**
1	**1**	**0**	**0**	**1**

7. Paste in you code:

```
module comparator
 #(
  parameter NUM_SEGMENTS = 8,
  parameter CLK_PER      = 10,
  parameter REFR_RATE    = 1000
 )
 (
  input wire                    clk,
  input wire                    CPU_RESETN,
  input wire [7:0]              SW,
  output logic [NUM_SEGMENTS-1:0] anode,
  output logic [7:0]            cathode
  );

 logic [NUM_SEGMENTS-1:0][7:0] encoded;
 logic [NUM_SEGMENTS-1:0]      digit_point;
 logic                         reset;

 assign reset = !CPU_RESETN;  // Tie reset to CPU_RESETN

 // Comparator Logic
 wire [3:0] B = SW[3:0];
 wire [3:0] A = SW[7:4];
```

```
logic A_greater, A_less, A_equal;


assign A_greater = (A > B);

assign A_less = (A < B);

assign A_equal = (A == B);


// 7-segment display for comparison result

always @(*) begin

  encoded = '0;  // Initialize to zero

  encoded[2] = A;  // Display A

  encoded[0] = B;  // Display B

  if (A_greater)

    encoded[1] = 8'h10; // "G"

  else if (A_less)

    encoded[1] = 8'h11; // "L"

  else

    encoded[1] = 8'h12; // "E"

end


// Instantiate seven_segment module

seven_segment #(

  .NUM_SEGMENTS(NUM_SEGMENTS),

  .CLK_PER(CLK_PER),

  .REFR_RATE(REFR_RATE)

) u_7seg (
    .clk(clk),
    .reset(reset),
    .encoded(encoded),
    .digit_point(digit_point),
    .anode(anode),
    .cathode(cathode)
  );
endmodule
// seven_segment.sv
//
// Encapsulate multiple seven segment displays using the cathode driver plus an
// anode driver.
`timescale 1ns/10ps
module seven_segment
  #
  (
   parameter NUM_SEGMENTS = 8,
   parameter CLK_PER      = 10,  // Clock period in ns
   parameter REFR_RATE    = 1000 // Refresh rate in Hz
   )
  (
   input wire                       clk,
   input wire                       reset, // active high reset
   input wire [NUM_SEGMENTS-1:0][7:0] encoded,
   input wire [NUM_SEGMENTS-1:0]      digit_point,
 output logic [NUM_SEGMENTS-1:0]    anode,
 output logic [7:0]                 cathode
 );

localparam INTERVAL = int'(100000000 / (CLK_PER * REFR_RATE));

logic [$clog2(INTERVAL)-1:0]        refresh_count;
logic [$clog2(NUM_SEGMENTS)-1:0]    anode_count;
logic [NUM_SEGMENTS-1:0][7:0]       segments;

cathode_top ct[NUM_SEGMENTS]
  (
   .clk        (clk),
   .encoded    (encoded),
   .digit_point(digit_point),
   .cathode    (segments)
   );

initial begin
  refresh_count = '0;
  anode_count   = '0;
end

always @(posedge clk) begin
  if (refresh_count == INTERVAL) begin
```

```
      refresh_count        <= '0;
      anode_count          <= anode_count + 1'b1;
    end else refresh_count <= refresh_count + 1'b1;
    anode                  <= '1;
    anode[anode_count]     <= '0;
    cathode                <= segments[anode_count];
    if (reset) begin
      refresh_count        <= '0;
      anode_count          <= '0;
    end
  end

endmodule

// cathode_top.sv
// ------------------------------------
// Drive the cathodes of 7 segment display
// ------------------------------------
//
// input the encoded value from 0-F and generate the cathode signals
`timescale 1ns/10ps
module cathode_top
  (
   input wire        clk,
   input wire [7:0]   encoded,
 input wire         digit_point,
 output logic [7:0] cathode
 );

  always_ff @(posedge clk) begin
cathode[7] <= digit_point;
case (encoded)
  8'h00: cathode[6:0] <= 7'b1000000; // '0'
  8'h01: cathode[6:0] <= 7'b1111001; // '1'
  8'h02: cathode[6:0] <= 7'b0100100; // '2'
  8'h03: cathode[6:0] <= 7'b0110000; // '3'
  8'h04: cathode[6:0] <= 7'b0011001; // '4'
  8'h05: cathode[6:0] <= 7'b0010010; // '5'
  8'h06: cathode[6:0] <= 7'b0000010; // '6'
  8'h07: cathode[6:0] <= 7'b1111000; // '7'
  8'h08: cathode[6:0] <= 7'b0000000; // '8'
  8'h09: cathode[6:0] <= 7'b0010000; // '9'
  8'h0A: cathode[6:0] <= 7'b0001000; // 'A'
  8'h0B: cathode[6:0] <= 7'b0000011; // 'b'
  8'h0C: cathode[6:0] <= 7'b1000110; // 'C'
  8'h0D: cathode[6:0] <= 7'b0100001; // 'd'
  8'h0E: cathode[6:0] <= 7'b0000110; // 'E'
  8'h0F: cathode[6:0] <= 7'b0001110; // 'F'
  8'h10: cathode[6:0] <= 7'b1000010; // 'G'
  8'h11: cathode[6:0] <= 7'b1000111; // 'L'
```

```
    8'h12: cathode[6:0] <= 7'b0000110; // 'E'
    8'h13: cathode[6:0] <= 7'b1001000; // 'H'
    8'h14: cathode[6:0] <= 7'b0001100; // 'P'
    8'h15: cathode[6:0] <= 7'b1110111; // 'r'
    8'h16: cathode[6:0] <= 7'b0000111; // 'T'
    default: cathode[6:0] <= 7'b1111111; // Blank
  endcase
end



endmodule
```