Lab 5 Submission

**8-Bit Comparator with 4-Bit Comparator Modules**

CPE 133 - 03

Michael Hegglin

Jonathan Skelly

**Executive Summary:**

We designed an 8-bit comparator with two 4-bit comparators to output whether or not the two 8-bit inputs where equal, less than, or equal to each other. We implemented this circuit through Verilog, which could then be downloaded to the board.





**Questions:**

**1. Briefly describe whether you could have used a decoder for the “logic box” portion of this lab? Also, provide a justification as to why you would not and/or did not use a decoder for this part of the circuit.**

* We could have used a decoder due to the fact that we could make a truth table for this design. There reason we didn’t use it was due to the fact that it would take a lot more time, and be harder to implement vs just writing our own logic. This also means there’s more room for mistakes going the decoder route.

**2. In your own words, briefly describe the advantage(s) of using a behavioral model as opposed to a gate-level implementation of the comparator.**

* A person can implement the logic through statements instead of direct boolean equations. This saves a lot of time.

**3. Based on the previous question, you probably see that modeling comparators of any size is no big deal using an HDL. Why then did this lab activity ask that you model a larger comparator using two smaller comparators?**

* We used two smaller comparators to show us hierarchical design. A larger comparator does work, and would be easier to implement however.

**4. We all know that the well-known hallmark of a comparator is that it uses EXOR-type gates in its implementation. Does the development board you’re using actually use EXOR-type gates in your design? Briefly but completely explain. FreeRange Digital Design - 19 - Lab Activity Manual**

* No, our development board uses LUTs to create digital gates for the physical board.

**5. The always block is one of Verilog’s concurrent statements. But, the always block contains only sequential statements in the body of the process statement. Describe how this seemingly oxymoron is actually possible.**

* The statements inside the always block are telling the always block when it should be executing the code inside of it. In this sense the concurrent statement implementation of the always block is being delayed by the statements inside of it.

**6. The always block contains a “process sensitivity list”. Briefly describe what this is. Also, state what you should generally place in the process sensitivity list.**

* The process sensitivity list is a controller that controls when all statements in the always block will be evaluated.

**7. In your own words, what is a test vector and what entity generates the vectors for your testbench?**

* A test vector is a list of possible inputs, with their respective outputs, to make sure the circuit is functioning correctly. The initial statement with the logic module we wrote generate the vectors for out test bench.

**8. Briefly explain what it would mean if the output of your simulation did not match the outputs on hardware for the same inputs?**

* It would mean that something is wrong. The problem would be somewhere inside the logic we wrote in Verilog.

**9. We often use hexadecimal notation in digital-land. Can you use hex notation to represent signed binary numbers in RC format? Briefly explain.**

* Yes signed binary numbers can be written in hexadecimal. The use of hexadecimal is to understand binary easier.

**Design Problem:**

**1. Design a circuit that has four 10-bit unsigned binary inputs A, B, C, D. The output of the circuit has the same three outputs as a normal comparator. If the sum of A+C is valid (10-bit result) and the sum of B+D is value (10-bit result), then the output of the circuit reflects the result the comparison of A+C & B+D. If either addition operation generates a carry, all of the three circuit outputs (EQ, LT, & GT) should be zero. Don’t use any MUXes in your solution. Draw the BBD for this circuit as well as the lower-level schematic. Minimize your use of hardware in your design.**





**Source Code:**

**Comporators:**

`timescale 1ns / 1ps  
//////////////////////////////////////////////////////////////////////////////////  
// Company: Ratner Engineering  
// Engineer: James Ratner  
//   
// Create Date: 07/04/2018 02:13:56 PM  
// Design Name:   
// Module Name: comp\_nb  
// Project Name:   
// Target Devices:   
// Tool Versions:   
// Description: n-bit comparator model  
//  
// Usage (instantiation) example for 16-bit comparator   
// (model defaults to 8-bit comparator)  
//  
// comp\_nb #(.n(16)) MY\_COMP (  
// .a (my\_a),   
// .b (my\_b),   
// .eq (my\_eq),   
// .gt (my\_gt),   
// .lt (my\_lt)  
// );   
//   
// Dependencies:   
//   
// Revision:  
// Revision 1.00 - File Created: 07-06-2018  
// Additional Comments:  
//   
//////////////////////////////////////////////////////////////////////////////////  
  
  
module comp\_nb(a, b, eq, lt, gt);   
 input [n-1:0] a,b;   
 output reg eq, lt, gt;   
   
 parameter n = 8;  
   
 always @ (a,b)  
 begin   
 if (a == b)  
 begin   
 eq = 1; lt = 0; gt = 0;   
 end  
 else if (a > b)   
 begin   
 eq = 0; lt = 0; gt = 1;   
 end  
 else if (a < b)   
 begin   
 eq = 0; lt = 1; gt = 0; end  
 else  
 begin   
 eq = 0; lt = 0; gt = 0;   
 end   
 end   
  
endmodule

**Main:**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 10/12/2018 12:43:24 PM

// Design Name:

// Module Name: Logic\_file

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module Logic\_file(a, b, eq, lt, gt);

input [7:0] a, b;

output reg eq, lt, gt;

wire eq1, eq2, lt1, lt2, gt1, gt2;

comp\_nb #(.n(4)) comp1 (

.a (a[7:4]),

.b (b[7:4]),

.eq (eq1),

.lt (lt1),

.gt (gt1)

);

comp\_nb #(.n(4)) comp2 (

.a (a[3:0]),

.b (b[3:0]),

.eq (eq2),

.lt (lt2),

.gt (gt2)

);

always @ (eq1, eq2, lt1, lt2, gt1, gt2)

begin

if(eq1 == 1 & eq2==1)

begin

eq = 1; lt = 0; gt = 0;

end

else if(gt1 == 1)

begin

eq = 0; lt = 0; gt = 1;

end

else if(lt1 == 1)

begin

eq =0; lt = 1; gt = 0;

end

else if(eq1 == 1)

begin

if(gt2 == 1)

begin

eq = 0; lt = 0; gt = 1;

end

else if(lt2 == 1)

begin

eq = 0; lt = 1; gt = 0;

end

end

end

endmodule

**Testbench:**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 10/15/2018 12:43:46 PM

// Design Name:

// Module Name: testbench

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module tb\_comp\_4b( );

reg [7:0] a, b; //- stimulus outputs

wire eq, lt, gt;

Logic\_file comp2 (

.a (a),

.b (b),

.eq(eq),

.lt(lt),

.gt(gt)

);

initial

begin

//- initial values of a & b

a = 'hab;

b = 'hbb;

//- a & b values 20 time units later

#20

a = 8'b11000000;

b = 8'b11000000;

//- a & b values 20 time units later

#20

a = 8'b00001011;

b = 8'b00000001;

//- a & b values 20 time units later

#20

a = 8'b10001110;

b = 8'b11001111;

#20

a = 8'b11011011;

b = 8'b01101001;

end

endmodule

