Lab 2

# 2 Bit Comparator

Errors in Code:

1. Missing semicolon on line 15 in eq2.v.
2. Line 15 in eq2.v had the arguments in the wrong order. Fixed these so that the two test bits are first and then the output bit.
3. Line 18 in eq2.v had the wrong operator for comparison. Changed the line to use the AND operator rather than the XOR operator.
4. Changed eq3 to eq1

Corrected Simulation Output

Graphical user interface

Description automatically generated with low confidence

Output in TCL console after using the “$monitor” command:

Vec 1: 00, Vec 2: 00, Output: 1

Vec 1: 01, Vec 2: 00, Output: 0

Vec 1: 01, Vec 2: 11, Output: 0

Vec 1: 10, Vec 2: 10, Output: 1

Vec 1: 10, Vec 2: 00, Output: 0

Vec 1: 11, Vec 2: 11, Output: 1

No it is not adequate testing since it doesn’t run through all the possible permutations of the inputs. We can’t take for granted that the output from (01, 10) will be the same as (10, 01).

# 2 Bit Greater than Circuit

Truth Table: (Testing if Vec0 > Vec1)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Vector 0 | | Test Vector 1 | | Output |
| A | B | C | D | Y |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

Minimized SOP:

y = AC' + BC'D' + ABD'

Output after Simulation:

Graphical user interface

Description automatically generated

Test Vectors:

Used a nested for loop in Verilog to run through all possible permutations for the two bit comparator. This is sufficient as it exhaustively tests all possible permutations.

For the greater than or equal to circuit, the two bit modules were instantiated and then connected together using a series of gates.

2 Bit Equal Module Verilog Code

// Listing 1.4

module eq2

(

input wire[1:0] a, b, // a and b are the two 2-bit numbers to compare

output wire aeqb // single bit output. Should be high if a adn b the same

);

// internal signal declaration, used to wire outpus of the 1 bit comparators

wire e0, e1;

// body

// instantiate two 1-bit comparators that we already know are tested and work

// named instantiation allows us to change order of ports.

eq1 eq\_bit0\_unit (.i0(a[0]), .i1(b[0]), .eq(e0));

eq1 eq\_bit1\_unit (.i0(a[1]), .i1(b[1]), .eq(e1));

// a and b are equal if individual bits are equal, which comes from the 1-bit comparators

assign aeqb = e0 & e1;

endmodule

2 Bit Greater Than

`timescale 1ns / 10ps

//2 bit Comparator Module. Checks if in0 > in1. Return 1 if true. 0 otherwise.

module bit2\_GTH(input wire [1:0] in0, input wire [1:0] in1, output wire comp);

/\*

y = AC' + BC'D' + ABD'

in0[1] = A

in0[0] = B

in1[1] = C

in1[0] = D

\*/

wire w1, w2, w3;

assign w1 = in0[1] & (~in1[1]);

assign w2 = in0[0] & (~in1[1]) & (~in1[0]);

assign w3 = in0[1] & in0[0] & (~in1[0]);

assign comp = w1 | w2 | w3;

endmodule

8 Bit Greater than or Equal

`timescale 1ns / 1ps

//tests A >= B

module bit8\_GEQ(input [7:0] A, input [7:0] B, output res);

wire eq1\_out, eq2\_out, eq3\_out, eq4\_out, res\_eq;

wire gth1\_out, gth2\_out, gth3\_out, gth4\_out, res\_gth;

eq2 equalA(.a(A[7:6]), .b(B[7:6]), .aeqb(eq1\_out)); //Aeq

eq2 equalB(.a(A[5:4]), .b(B[5:4]), .aeqb(eq2\_out)); //Beq

eq2 equalC(.a(A[3:2]), .b(B[3:2]), .aeqb(eq3\_out)); //Ceq

eq2 equalD(.a(A[1:0]), .b(B[1:0]), .aeqb(eq4\_out)); //Deq

assign res\_eq = eq1\_out & eq2\_out & eq3\_out & eq4\_out;

bit2\_GTH GTHA(.in0(A[7:6]), .in1(B[7:6]), .comp(gth1\_out)); //A

bit2\_GTH GTHB(.in0(A[5:4]), .in1(B[5:4]), .comp(gth2\_out)); //B

bit2\_GTH GTHC(.in0(A[3:2]), .in1(B[3:2]), .comp(gth3\_out)); //C

bit2\_GTH GTHD(.in0(A[1:0]), .in1(B[1:0]), .comp(gth4\_out)); //D

assign res\_gth = gth1\_out | (gth2\_out & eq1\_out) | (gth3\_out & eq2\_out & eq1\_out) | (gth4\_out & eq3\_out & eq2\_out & eq1\_out);

assign res = res\_eq | res\_gth;

endmodule

2 Bit Equal Testbench

`timescale 1 ns/10 ps

module testbench;

// signal declaration

reg [1:0] test\_in0, test\_in1;

wire test\_out;

integer vec1, vec2;

// instantiate the circuit under test

bit2\_GTH uut (.in0(test\_in0), .in1(test\_in1), .comp(test\_out));

// test vector generator

initial

begin

$monitor("Vec 1: %2b, Vec 2: %2b, Output: %1b", test\_in0, test\_in1, test\_out);

// test\_in0 = 2'b10;

// test\_in1 = 2'b10;

for(vec1=2'b00;vec1<=2'b11;vec1=vec1+1)begin

for(vec2=2'b00;vec2<=2'b11;vec2=vec2+1)begin

test\_in0 = vec1;

test\_in1 = vec2;

#200;

end

end

end

endmodule

8 Bit Greater Than or Equal

`timescale 1ns / 1ps

module bit8\_GEQ\_tb;

reg [7:0] test\_in0, test\_in1;

integer vec1, vec2;

wire result;

bit8\_GEQ gate(.A(test\_in0), .B(test\_in1), .res(result));

initial begin

for(vec1=8'b00000000;vec1<=8'b11111111;vec1=vec1+1)begin

for(vec2=8'b00000000;vec2<=8'b11111111;vec2=vec2+1)begin

test\_in0 = vec1;

test\_in1 = vec2;

#200;

end

end

end

endmodule