**Logo

Description automatically generated San Francisco Bay University**

**EE461 Verilog-HDL**

**Homework #4**

**Due day: 3/18/2023**

**Name: Simon Gezae ID: 19830**

**Instruction:**

1. **Push answer sheets/source code to Github**
2. **Please follow the code style rule like programs on handout.**
3. **Overdue homework submission could not be accepted.**
4. **Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**
5. Design a detector in RTL level to detect if all bits are *0*s or all *1*s for an 8-bits input. If all bits are *0*s, one of two outputs, "*zeroflag*" is *1*. If all are *1*s, the other of two outputs, "*oneflag*" is *1*. After that, write the gate level module to compare two designs with the testbench.

Ans:

module detector(in, zeroflag, oneflag);

input [7:0] in;

output zeroflag, oneflag;

assign zeroflag = !(in[7] | in[6] | in[5] | in[4] | in[3] | in[2] | in[1] | in[0]);

assign oneflag = (in[7] & in[6] & in[5] & in[4] & in[3] & in[2] & in[1] & in[0]);

endmodule

module detector(in, zeroflag, oneflag);

input [7:0] in;

output zeroflag, oneflag;

reg zeroflag, oneflag;

wire [7:0] zero = 8'b0000\_0000;

wire [7:0] one = 8'b1111\_1111;

always@(in)begin

zeroflag <= (in==zero);

oneflag <= (in==one);

end

endmodule

Testbench:

module tb();

reg [7:0] in;

wire zeroflag, oneflag;

integer i;

detector u1 (in, zeroflag, oneflag);

initial begin

$display("in\t\toneflag\tzeroflag");

$monitor("%b\t%b\t%b", in, oneflag, zeroflag);

for (i=0; i<3; i=i+1)begin

in = i;

#2;

end

#2 in = 8'b1111\_1111;

#2 $finish;

end

endmodule

Result:

Graphical user interface, text, application, email

Description automatically generated

Edaplayground link: <https://edaplayground.com/x/XPPC>

1. Design 4-bits 2's complement number converter in the gate level module

Design:

module twoscomp(in, out);

input [3:0] in;

output [3:0] out;

wire [3:0] onescomp;

assign onescomp = ~ in;

assign out = onescomp + 1'b1;

endmodule

Testbench:

module tb;

reg [3:0] in;

wire [3:0] out;

twoscomp u1 (in, out);

initial begin

$monitor("in = %b out = %b", in, out);

in = 4'b1111;

#2 in = 4'b0101;

#2 in = 4'b0000;

#2 in = 4'b0001;

#2 in = 4'b1001;

end

endmodule

Result:

Graphical user interface, text, application

Description automatically generated

Edaplayground: <https://edaplayground.com/x/Z_iM>

1. Design 4-1 mux in continuous assign. Write a testbench to assign *x* & *z* to select-bits to observe what you are going to get.

Design:

module mux4(a,b,c,d,sel,out);

input [3:0] a, b, c, d;

input [1:0] sel;

output [3:0] out;

assign out = sel[1]?(sel[0]?d:c):(sel[0]?b:a);

endmodule

Testbench:

module tb;

reg [3:0] a,b,c,d;

reg [1:0] sel;

wire [3:0] out;

mux4 u1 (a,b,c,d,sel,out);

initial begin

$monitor("a=%b b=%b c=%b d=%b sel=%b\tout=%b", a, b, c, d, sel, out);

a=4'b0001; b=4'b0000; c=4'b0000; d=4'b0000; sel = 2'b00;

#2 a=4'b0001; b=4'b0000; c=4'b0000; d=4'b0000; sel = 2'b00;

#2 a=4'b0000; b=4'b0001; c=4'b0000; d=4'b0000; sel = 2'b01;

#2 a=4'b0000; b=4'b0000; c=4'b0001; d=4'b0000; sel = 2'b10;

#2 a=4'b0000; b=4'b0000; c=4'b0000; d=4'b0001; sel = 2'b11;

#2 a=4'b1000; b=4'b1010; c=4'b1100; d=4'b1000; sel = 2'bxx;

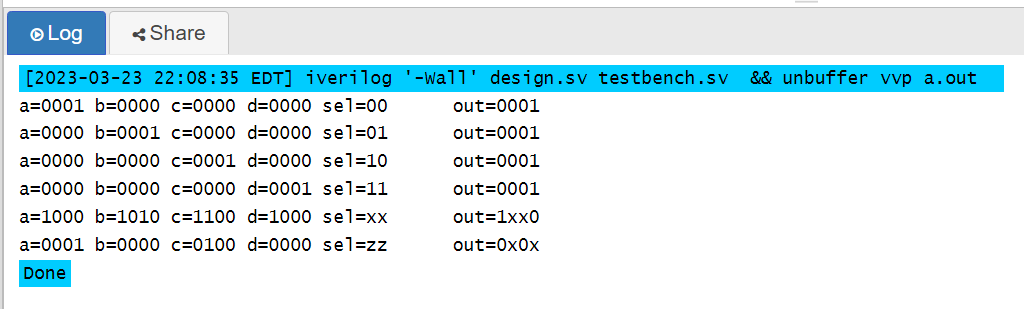
#2 a=4'b0001; b=4'b0000; c=4'b0100; d=4'b0000; sel = 2'bzz;

#2 $finish;

end

endmodule

Result:



Comment: when select is x or z the output is the comparision between the bits of the input when all the input are 1 the output is 1 when there is mixup input the output is x.

Edaplayground: <https://edaplayground.com/x/EsYz>

1. Draw the circuit schematics for the following two always block and compare what the difference is.

*a. always @(a, b, c) begin*

*d = a + b;*

*e = d + c;*

*end*

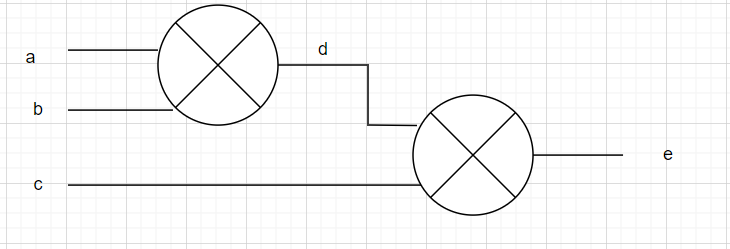
*b. always @(a, b, c) begin*

*e = d + c;*

*d = a + b;*

*end*

*(A)*



(b)

Diagram, engineering drawing

Description automatically generated

Comment: In the first design we have three inputs and d that act as a wire the trasfers the value of a plus b and then it is added with c to get the value of e. In this design there is no condition violation as the variable in sensitivity list are added in order. In the second design we will have problem with d as it is not listed in sensitivity list but used as input. The older value will be used to find e even d is updated when a and b are changed.

1. When designing a 5-to-1 mux by "case" structure, inferred latch will be generated if design isn’t in proper way after logic synthesis. Please fix it and compare new design hardware schematic with original one.

*module mux(a,b,c,d,e,sel,out);*

*input a,b,c,d,e;*

*input[2:0] sel;*

*output out;*

*reg out;*

*always @(a, b, c, d, e, sel) begin*

*case(sel)*

*3'b000: out=a;*

*3'b001: out=b;*

*3'b010: out=c;*

*3'b011: out=d;*

*3'b100: out=e;*

*endcase*

*end*

*endmodule*

new design:

module mux(a,b,c,d,e,sel,out);

input a,b,c,d,e;

input[2:0] sel;

output out;

reg out;

always @(a, b, c, d, e, sel) begin

case(sel)

3'b000: out=a;

3'b001: out=b;

3'b010: out=c;

3'b011: out=d;

3'b100: out=e;

default: out=1'bx;

endcase

end

endmodule

Testbench:

module tb;

reg a,b,c,d,e;

reg [2:0] sel;

wire out;

mux u1 (a,b,c,d,e,sel,out);

initial begin

$monitor("a=%b b=%b c=%b d=%b e=%b sel=%b\tout=%b", a, b, c, d, e, sel, out);

a=4'b0001; b=4'b0000; c=4'b0000; d=4'b0000; e=4'b0000; sel = 2'b00;

#2 a=4'b0001; b=4'b0000; c=4'b0000; d=4'b0000; e=4'b0000; sel = 3'b000;

#2 a=4'b0000; b=4'b0001; c=4'b0000; d=4'b0000; e=4'b0000; sel = 3'b001;

#2 a=4'b0000; b=4'b0000; c=4'b0001; d=4'b0000; e=4'b0000; sel = 3'b010;

#2 a=4'b0000; b=4'b0000; c=4'b0000; d=4'b0001; e=4'b0000; sel = 3'b011;

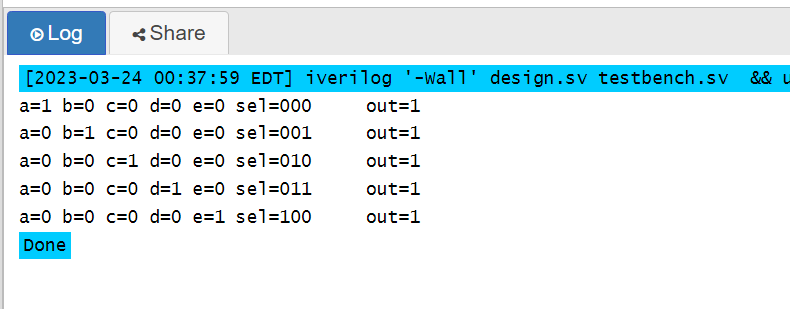
#2 a=4'b0000; b=4'b0000; c=4'b0000; d=4'b0000; e=4'b0001; sel = 3'b100;

#2 $finish;

end

endmodule

Result:



Comment:

In case statement if it does not end up without default statement it infers latch. To fix it default is added to the design.

Edaplayground: <https://edaplayground.com/x/Bn7z>

