Homework 2

Ashmitha Murali

Prof. Alex Yang

Digital Design and HDL EE461

25th September 2025

**Q1. If a = 4’b1111 and b = -5’b00010, write the program to check what the values with 4  
bits are when you want to calculate “a (+/-/\*/%) b”. How about b = -5’b01xz ?**

**Ans. When b is assigned -5b’00010, the 2’s complement of b is calculated which is 5’b11110**

Below is the program and result when output is assigned to 4 bits after all the operations.

**Result:**

**When b= -5’b00010**

a+b =4’b1101

a-b = 4’b0001

a\*b = 4’b0010

a%b = 4’b1111

When b=-5’b01xz all output become 4’bxxxx

**A screenshot of a computer

AI-generated content may be incorrect.**

**Code:**

module arith\_op;

reg [3:0] a;

reg [4:0] b;

wire [3:0] add\_ab, sub\_ab, mul\_ab, mod\_ab;

assign add\_ab = a + b;

assign sub\_ab = a - b;

assign mul\_ab = a \* b;

assign mod\_ab = a % b;

initial begin

//CASE 1

a = 4'b1111;

b = -5'b00010;

#1 $display("CASE 1 : when b=-5'b00010");

$display("a and b in decimal : a=%d b=%d", a, b);

$display("a and b in binary : a=%b b=%b", a, b);

$display("a + b = %b", add\_ab);

$display("a - b = %b", sub\_ab);

$display("a \* b = %b", mul\_ab);

$display("a %% b = %b", mod\_ab);

//CASE 2

b = -5'b01xz;

#1 $display("\nCASE 2 : when b=-5'b01xz");

$display("a + b = %b", add\_ab);

$display("a - b = %b", sub\_ab);

$display("a \* b = %b", mul\_ab);

$display("a %% b = %b", mod\_ab);

$finish;

end

endmodule

**Q2. When a = 2’b1z and b = 3’b11z, verify the values for (a>b), (a>=b), (a<b) and (a<=b)  
by a program. If a = 4’b01xz, check again.**

**Ans.**

**All the comparisons returned x as output since we have ‘z’ and ‘x’ in input.**

**Result:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Code used:**

**module comparison;**

reg [3:0] a;

reg [2:0] b;

wire gt = a>b;

wire ge = a>=b;

wire lt = a<b;

wire le = a<=b;

initial begin

//Case 1

a = 2'b1z;

b = 3'b11z;

#1;

$display("CASE 1: a=2'b1z, b=3'b11z");

$display("(a > b) = %b", gt);

$display("(a >= b) = %b", ge);

$display("(a < b) = %b", lt);

$display("(a <= b) = %b", le);

//Case 2

a = 4'b01xz;

#1;

$display("\nCASE 2: a=4'b01xz, b=3'b11z");

$display("(a > b) = %b", gt);

$display("(a >= b) = %b", ge);

$display("(a < b) = %b", lt);

$display("(a <= b) = %b", le);

$finish;

end

endmodule

Q3. **Write a program to see results for 4 questions on “Equality Operators” page in the  
handout.**

**Ans.**

**Result:  
A white background with black and white text

AI-generated content may be incorrect.**

**Code Used:**

module equality\_test;

reg [3:0] a, b;

wire eq1, neq1, eq2, neq2;

assign eq1 = (a === b);

assign neq1 = (a !== b);

assign eq2 = (a == b);

assign neq2 = (a != b);

initial begin

$monitor("time=%0t | a=%b | b=%b | (a===b) = %b | (a!==b) = %b | (a==b) = %b | (a!=b) = %b", $time, a, b, eq1, neq1, eq2, neq2);

a = 4'b01xz; b = 4'bzx10; #1;

a = 4'b01xz; b = 4'b01xz; #1;

a = 4'b01zz; b = 4'b0100; #1;

a = 4'b01zz; b = 4'b01zz; #1;

$finish;

end

endmodule

**Q4. Verify the results by a program for the following “A’s values”.**

**Ans.**

|  |  |
| --- | --- |
| **A** | **!A** |
| **1’bx** | **1’bx** |
| **1’bz** | **1’bx** |
| **2’b1z** | **1’b0** |
| **2’b0z** | **1’bx** |
| **2’bxz** | **1’bx** |
| **3’bxxx** | **1’bx** |
| **3’b1xx** | **1’b0** |
| **3’b0xx** | **1’bx** |

**Result:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Code:**

module not\_a\_test;

reg a1;

reg [1:0] a2;

reg [2:0] a3;

wire not\_a1;

wire not\_a2;

wire not\_a3;

assign not\_a1 = !a1;

assign not\_a2 = !a2;

assign not\_a3 = !a3;

initial begin

$monitor("time=%0t | a=%b | !a = %b", $time, a1, not\_a1);

a1 = 1'bx; #1;

a1 = 1'bz; #1;

$monitor("time=%0t | a=%b | !a = %b", $time, a2, not\_a2);

a2 = 2'b1z; #1;

a2 = 2'b0z; #1;

a2 = 2'bxz; #1;

$monitor("time=%0t | a=%b | !a = %b", $time, a3, not\_a3);

a3 = 3'bxxx; #1;

a3 = 3'b1xx; #1;

a3 = 3'b0xx; #1;

$finish;

end

endmodule

Q5. **Write a program to see what you will get for 1’bx && 2’bxz , 2’b0x || 1’bz ,  
2’b00 && 2’b1z and 2’b0z || 4’b01xz.**

**Ans.**

**Result:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Code:**

**module test\_logic;**

**reg a1;**

**reg [1:0] b2;**

**reg [1:0] c2;**

**reg d1;**

**reg [1:0] e2;**

**reg [1:0] f2;**

**reg [1:0] g2;**

**reg [3:0] h4;**

**wire r1 = (a1 && b2);**

**wire r2 = (c2 || d1);**

**wire r3 = (e2 && f2);**

**wire r4 = (g2 || h4);**

**initial begin**

**a1 = 1'bx; b2 = 2'bxz; #1;**

**$display("time=%0t | %b && %b = %b ", $time, a1,b2,r1);**

**c2 = 2'b0x; d1 = 1'bz; #1;**

**$display("time=%0t | %b || %b = %b ", $time, c2,d1,r2);**

**e2 = 2'b00; f2 = 2'b1z; #1;**

**$display("time=%0t | %b && %b = %b ", $time, e2,f2,r3);**

**g2 = 2'b0z; h4 = 4'b01xz; #1;**

**$display("time=%0t | %b || %b = %b ", $time, g2,h4,r4);**

**#5 $finish;**

**end**

**endmodule**

**Q6. What are the results in the following operations and verify them by Verilog code?**

**Ans.**

**Result:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Code:**

**module bitwise\_verify;**

**reg [3:0] A = 4'b01xz;**

**reg [3:0] B = 4'bzx01;**

**reg [1:0] C = 2'bz1;**

**wire [3:0] notA, AandB, AorB, AxorB, AxnorB, AxnorC4;**

**wire [1:0] AxnorC2;**

**wire [3:0] C4;**

**assign C4 = {2'b00, C};**

**assign notA = ~A;**

**assign AandB = A & B;**

**assign AorB = A | B;**

**assign AxorB = A ^ B;**

**assign AxnorB = A ^~ B;**

**assign AxnorC4 = A ^~ C4;**

**assign AxnorC2 = A ^~ C;**

**initial begin**

**$display("~%b = %b",A, notA);**

**$display("%b & %b = %b",A,B, AandB);**

**$display("%b | %b = %b",A,B, AorB);**

**$display("%b ^ %b = %b",A,B, AxorB);**

**$display("%b ^~ %b = %b",A,B, AxnorB);**

**$display("%b ^~ %b = %b",A,C4, AxnorC4);**

**$display("(%b ^~ %b) = %b",A,C, AxnorC2);**

**$finish;**

**end**

**endmodule**

**Q7.** **What are you going to get for “& 4’b01xz” , “~| 4’b01xz” , “^ 4’b01xz” and  
“~^ 4’b01xz”**

**Ans.**

**& 4’b01xz = 0**

**~| 4’b01xz = 0**

**^ 4’b01xz = x**

**~^ 4’b01xz = x**

**Result:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Code used:**

**module reduction\_demo;**

**reg [3:0] A = 4'b01xz;**

**wire r\_and = &A;**

**wire r\_nor = ~|A;**

**wire r\_xor = ^A;**

**wire r\_xnor = ~^A;**

**initial begin**

**$display("A = %b", A);**

**$display("&A = %b", r\_and);**

**$display("~|A = %b", r\_nor);**

**$display("^A = %b", r\_xor);**

**$display("~^A = %b", r\_xnor);**

**$finish;**

**end**

**endmodule**

**Q8. What are the new values after bit shifting for “4'b01xz << 1'bz” and “4'b01xz >>2'bxx” ?**

**Ans.** The output of 4'b01xz << 1'bz will be 4’bxxxx

The output of 4'b01xz >>2'bxx will also be 4’bxxxx

If the shift value is x,z or not known, then the output of shift will be x.

**Q9. In this expression A = B ? 4’b1100 : 5’b11ZX0 and if B = 2’b1x, What is A(4-bit  
number)? How about B= 3’b1xz? Write a program to verify your answers.**

**Ans. Both B=2’b1x and B=3’b1xz are logically equal to 1. Hence B always evaluates to 1**

**Result**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Code:**

**module cond\_merge\_tb;**

**reg [2:0] B;**

**wire [4:0] A;**

**assign A = B ? 4'b1100 : 5'b11ZX0;**

**initial begin**

**B = 2'b1x; #1;**

**$display("Case 1: B=%b ", B);**

**$display("A(4bit) = %b ", A[3:0]);**

**B = 3'b1xz; #1;**

**$display("Case 2: B=%b ", B);**

**$display("A(4bit) = %b ", A[3:0]);**

**$finish;**

**end**

**endmodule**

**Q10. Complete the following Verilog modules and display the output strength. Explain why**

**Ans.**

**Completed code:**

**module testStrength1();**

**reg a,b;**

**tri y; // Data type declaration for a, b and y**

**buf (strong1, weak0) g1 (y, a);**

**buf (weak1, strong0) g2 (y, b);**

**initial begin**

**a = 1;**

**b = 1;**

**$display("y = %v, a = %b , b = %b", y, a, b);**

**end**

**endmodule**

**module testStrength2();**

**reg i1, i2, ctrl;**

**tri y; // Data type declaration for a, b and y**

**bufif0 (strong1, weak0) g1 (y, i1, ctrl);**

**bufif0 (strong1, weak0) g2 (y, i2, ctrl);**

**initial begin**

**ctrl = 1'bx;**

**i1 = 0;**

**i1 = 1;**

**$display("y = %v",y);**

**end**

**endmodule**

**Reason for changes**

**Datatypes:**

Assigning datatype of **‘reg’** for a, b, i1, i2 and crtl since these signals are assigned inside initial block and the LHS inside initial block should be reg.

Assigning datatype of ‘**tri’ for y** since it can have multiple outputs connected to this signal.

**Formatter:**

Using ‘**%v’** as formatter for y, since this formatter can print values with drive strength

Assigning %b formatter for a and b since they are basic input bits that do not need drive strength information

**Q11. Design Verilog program for 4 to 1 mux in gate level and write the testbench to verify.**

**Ans.**

**Result:**

**A screenshot of a computer code

AI-generated content may be incorrect.**

**Waveform:**

**A black rectangular object with green lines

AI-generated content may be incorrect.**

**Design code:**

**module fourOneMux (**

**input i0\_i, i1\_i, i2\_i, i3\_i,**

**input s1\_i, s0\_i,**

**output out\_o**

**);**

**wire n1, n0;**

**wire y0, y1, y2, y3;**

**// Inverters**

**not (n1, s1\_i);**

**not (n0, s0\_i);**

**// AND gates for each data input**

**and (y0, i0\_i, n1, n0);**

**and (y1, i1\_i, n1, s0\_i);**

**and (y2, i2\_i, s1\_i, n0);**

**and (y3, i3\_i, s1\_i, s0\_i);**

**// OR gate to produce final output**

**or (out\_o, y0, y1, y2, y3);**

**endmodule**

**Testbench:**

**`include “fourOneMux”**

**`timescale 1ns/1ps**

**module fourOneMuxTB;**

**reg i0, i1, i2, i3;**

**reg s1, s0;**

**wire out;**

**fourOneMux uut (**

**.i0\_i(i0), .i1\_i(i1), .i2\_i(i2), .i3\_i(i3),**

**.s1\_i(s1), .s0\_i(s0),**

**.out\_o(out)**

**);**

**integer k;**

**initial begin**

**$dumpfile("mux4\_gate.vcd");**

**$dumpvars(0, tb\_mux4\_gate);**

**end**

**initial begin**

**i0 = 1'b0;**

**i1 = 1'b1;**

**i2 = 1'b0;**

**i3 = 1'b1;**

**$display("time | s1 s0 | i3 i2 i1 i0 | out");**

**$monitor("%4t | %b %b | %b %b %b %b | %b",**

**$time, s1, s0, i3, i2, i1, i0, out);**

**s1 = 0; s0=0; #10**

**s1 = 0; s0=1; #10**

**s1 = 1; s0=0; #10**

**s1 = 1; s0=1; #10**

**$finish;**

**end**

**endmodule**

**Q12. Write nor gate Verilog module using switch devices and testbench to verify it.**

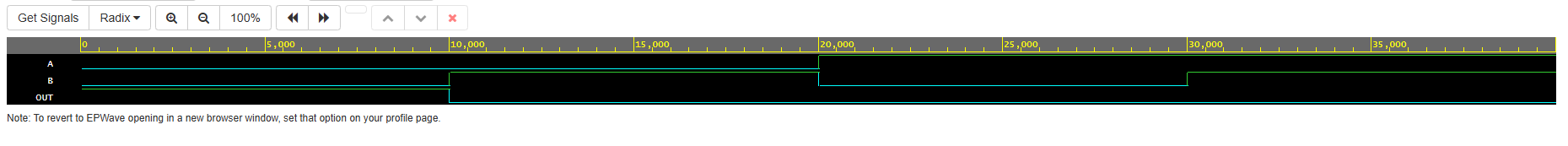
**Ans.**

**Result:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Waveform:**

****

**Design Code:**

**module nor\_cmos (**

**output out,**

**input A,**

**input B**

**);**

**// power rails**

**supply1 VDD;**

**supply0 VSS;**

**wire p\_mid;**

**pmos Pbot(out, p\_mid, A);**

**pmos Ptop(p\_mid, VDD, B);**

**nmos N1(out, VSS, A);**

**nmos N2(out, VSS, B);**

**endmodule**

**Testbench:**

**`timescale 1ns/1ps**

**module tb\_nor\_cmos;**

**reg A, B;**

**wire OUT;**

**nor\_cmos dut (.out(OUT), .A(A), .B(B));**

**initial begin**

**// VCD waveform**

**$dumpfile("nor\_cmos.vcd");**

**$dumpvars(0, tb\_nor\_cmos);**

**$display("time | A B | OUT = ~(A|B))");**

**$display("--------------------------");**

**A=0; B=0; #10 $display("%4t | %b %b | %b", $time,A,B,OUT);**

**A=0; B=1; #10 $display("%4t | %b %b | %b", $time,A,B,OUT);**

**A=1; B=0; #10 $display("%4t | %b %b | %b", $time,A,B,OUT);**

**A=1; B=1; #10 $display("%4t | %b %b | %b", $time,A,B,OUT);**

**$finish;**

**end**

**endmodule**