**Nang Thiri Wutyi**

**20113**

**10/3/2024**

**Verilog HDL HW #2**

1. **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?**

module arithmetic\_operations();

// Declaration of variables

reg signed [3:0] a; // 4-bit signed number

reg [4:0] b; // 5-bit signed number

reg signed [3:0] result\_add, result\_sub, result\_mul, result\_mod;

initial begin

// Initialize value for a

a = 4'b1111; // a = -1 in 2's complement representation

// Test case 1: b = -5'b00010

b = -5'b00010; // b = -2 in 5-bit 2's complement representation

// Perform arithmetic operations

result\_add = a + b; // Addition

result\_sub = a - b; // Subtraction

result\_mul = a \* b; // Multiplication

result\_mod = a % b; // Modulo

// Display the results

$display("Test case 1: b = -5'b00010");

$display("a + b = %d", result\_add);

$display("a - b = %d", result\_sub);

$display("a \* b = %d", result\_mul);

$display("a %% b = %d", result\_mod);

// Test case 2: b with unknown values (-5'b01xz)

b = 5'b01xz; // b contains unknown (x) and high-impedance (z) bits

// Check if b contains unknown or high-impedance values

if (^b === 1'bx) begin

$display("Test case 2: b = 5'b01xz (Contains unknowns, results are invalid)");

end else begin

// Perform arithmetic operations

result\_add = a + b; // Addition

result\_sub = a - b; // Subtraction

result\_mul = a \* b; // Multiplication

result\_mod = a % b; // Modulo

// Display the results

$display("Test case 2: b = 5'b01xz");

$display("a + b = %b (unknowns possible)", result\_add);

$display("a - b = %b (unknowns possible)", result\_sub);

$display("a \* b = %b (unknowns possible)", result\_mul);

$display("a %% b = %b (unknowns possible)", result\_mod);

end

end

endmodule

**Results**

[2024-10-03 21:08:24 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out

Test case 1: b = -5'b00010

a + b = -3

a - b = 1

a \* b = 2

a % b = -1

Test case 2: b = 5'b01xz (Contains unknowns, results are invalid)

Done

1. **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.**

module relational\_operations();

// Declaration of variables

reg [1:0] a\_2bit; // 2-bit number for case 1

reg [2:0] b\_3bit; // 3-bit number for case 1

reg [3:0] a\_4bit; // 4-bit number for case 2

reg [2:0] b\_check; // Temporary variable for comparing 4-bit a with 3-bit b

initial begin

// Test case 1: a = 2'b1z and b = 3'b11z

a\_2bit = 2'b1z;

b\_3bit = 3'b11z;

// Relational operations for case 1

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

if (a\_2bit > b\_3bit)

$display("a > b = True");

else if (a\_2bit === b\_3bit)

$display("a = b = True");

else

$display("a > b = False");

if (a\_2bit >= b\_3bit)

$display("a >= b = True");

else

$display("a >= b = False");

if (a\_2bit < b\_3bit)

$display("a < b = True");

else

$display("a < b = False");

if (a\_2bit <= b\_3bit)

$display("a <= b = True");

else

$display("a <= b = False");

// Test case 2: a = 4'b01xz and b = 3'b11z

a\_4bit = 4'b01xz;

// Relational operations for case 2

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

// Ensure correct comparison size by truncating 4-bit a to match 3-bit b

b\_check = a\_4bit[2:0];

if (b\_check > b\_3bit)

$display("a > b = True");

else if (b\_check === b\_3bit)

$display("a = b = True");

else

$display("a > b = False");

if (b\_check >= b\_3bit)

$display("a >= b = True");

else

$display("a >= b = False");

if (b\_check < b\_3bit)

$display("a < b = True");

else

$display("a < b = False");

if (b\_check <= b\_3bit)

$display("a <= b = True");

else

$display("a <= b = False");

end

endmodule

**Results**

**[2024-10-03 21:12:34 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out**

**Test case 1: a = 2'b1z, b = 3'b11z**

**a > b = False**

**a >= b = False**

**a < b = False**

**a <= b = False**

**Test case 2: a = 4'b01xz, b = 3'b11z**

**a > b = False**

**a >= b = False**

**a < b = False**

**a <= b = False**

**Done**

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

module equalityOperatorsTest;

// Declare the input variables

reg [3:0] a, b; // 4-bit vectors

initial begin

// Case 1: a = 4'b01xz, b = 4'bzx10

a = 4'b01xz;

b = 4'bzx10;

$display("Case 1: a = 4'b01xz, b = 4'bzx10");

$display("a === b = %b", a === b); // Case equality

$display("a !== b = %b", a !== b); // Case inequality

$display("a == b = %b", a == b); // Logical equality

$display("a != b = %b", a != b); // Logical inequality

$display("");

// Case 2: a = 4'b01xz, b = 4'b01xz

a = 4'b01xz;

b = 4'b01xz;

$display("Case 2: a = 4'b01xz, b = 4'b01xz");

$display("a === b = %b", a === b); // Case equality

$display("a !== b = %b", a !== b); // Case inequality

$display("a == b = %b", a == b); // Logical equality

$display("a != b = %b", a != b); // Logical inequality

$display("");

// Case 3: a = 4'b01zz, b = 4'b0100

a = 4'b01zz;

b = 4'b0100;

$display("Case 3: a = 4'b01zz, b = 4'b0100");

$display("a === b = %b", a === b); // Case equality

$display("a !== b = %b", a !== b); // Case inequality

$display("a == b = %b", a == b); // Logical equality

$display("a != b = %b", a != b); // Logical inequality

$display("");

// Case 4: a = 4'b01zz, b = 4'b01zz

a = 4'b01zz;

b = 4'b01zz;

$display("Case 4: a = 4'b01zz, b = 4'b01zz");

$display("a === b = %b", a === b); // Case equality

$display("a !== b = %b", a !== b); // Case inequality

$display("a == b = %b", a == b); // Logical equality

$display("a != b = %b", a != b); // Logical inequality

$display("");

// End the simulation

$finish;

end

endmodule

**Results**

**[2024-10-04 07:52:06 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out**

**Case 1: a = 4'b01xz, b = 4'bzx10**

**a === b = 0**

**a !== b = 1**

**a == b = x**

**a != b = x**

**Case 2: a = 4'b01xz, b = 4'b01xz**

**a === b = 1**

**a !== b = 0**

**a == b = x**

**a != b = x**

**Case 3: a = 4'b01zz, b = 4'b0100**

**a === b = 0**

**a !== b = 1**

**a == b = x**

**a != b = x**

**Case 4: a = 4'b01zz, b = 4'b01zz**

**a === b = 1**

**a !== b = 0**

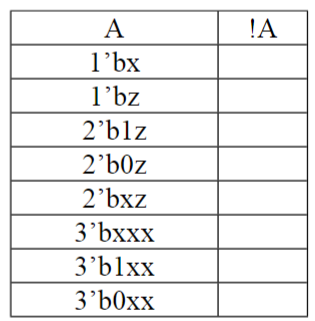
**a == b = x**

**a != b = x**

**design.sv:48: $finish called at 0 (1s)**

**Done**

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



module notOperationTest;

reg [2:0] A; // 3-bit input

reg [1:0] B; // 2-bit input

initial begin

// Test case 1: A = 1'bx

A = 1'bx;

$display("A = 1'bx, !A = %b", !A);

// Test case 2: A = 1'bz

A = 1'bz;

$display("A = 1'bz, !A = %b", !A);

// Test case 3: A = 2'b1z

B = 2'b1z;

$display("A = 2'b1z, !A = %b", !B);

// Test case 4: A = 2'b0z

B = 2'b0z;

$display("A = 2'b0z, !A = %b", !B);

// Test case 5: A = 2'bxz

B = 2'bxz;

$display("A = 2'bxz, !A = %b", !B);

// Test case 6: A = 3'bxxx

A = 3'bxxx;

$display("A = 3'bxxx, !A = %b", !A);

// Test case 7: A = 3'b1xx

A = 3'b1xx;

$display("A = 3'b1xx, !A = %b", !A);

// Test case 8: A = 3'b0xx

A = 3'b0xx;

$display("A = 3'b0xx, !A = %b", !A);

$finish; // End the simulation

end

endmodule

**Results**

**[2024-10-04 07:59:54 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out**

**A = 1'bx, !A = x**

**A = 1'bz, !A = x**

**A = 2'b1z, !A = 0**

**A = 2'b0z, !A = x**

**A = 2'bxz, !A = x**

**A = 3'bxxx, !A = x**

**A = 3'b1xx, !A = 0**

**A = 3'b0xx, !A = x**

**design.sv:39: $finish called at 0 (1s)**

**Done**

1. **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.**

module testExpressions;

// Declare inputs and outputs

reg [0:0] expr1\_1; // For 1'bx

reg [1:0] expr1\_2; // For 2'bxz

reg result1; // Result of expr1

reg [1:0] expr2\_1; // For 2'b0x

reg [0:0] expr2\_2; // For 1'bz

reg result2; // Result of expr2

reg [1:0] expr3\_1; // For 2'b00

reg [1:0] expr3\_2; // For 2'b1z

reg result3; // Result of expr3

reg [1:0] expr4\_1; // For 2'b0z

reg [3:0] expr4\_2; // For 4'b01xz

reg result4; // Result of expr4

initial begin

// Test expression 1: 1'bx && 2'bxz

expr1\_1 = 1'bx; // Initialize input A

expr1\_2 = 2'bxz; // Initialize input B

result1 = expr1\_1 && expr1\_2; // Evaluate expression

// Test expression 2: 2'b0x || 1'bz

expr2\_1 = 2'b0x; // Initialize input A

expr2\_2 = 1'bz; // Initialize input B

result2 = expr2\_1 || expr2\_2; // Evaluate expression

// Test expression 3: 2'b00 && 2'b1z

expr3\_1 = 2'b00; // Initialize input A

expr3\_2 = 2'b1z; // Initialize input B

result3 = expr3\_1 && expr3\_2; // Evaluate expression

// Test expression 4: 2'b0z || 4'b01xz

expr4\_1 = 2'b0z; // Initialize input A

expr4\_2 = 4'b01xz; // Initialize input B

result4 = expr4\_1 || expr4\_2; // Evaluate expression

// Display the results

$display("Result of 1'bx && 2'bxz: %b", result1);

$display("Result of 2'b0x || 1'bz: %b", result2);

$display("Result of 2'b00 && 2'b1z: %b", result3);

$display("Result of 2'b0z || 4'b01xz: %b", result4);

// End the simulation

$finish;

end

endmodule

**Results**

**[2024-10-03 22:05:06 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out**

**Result of 1'bx && 2'bxz: x**

**Result of 2'b0x || 1'bz: x**

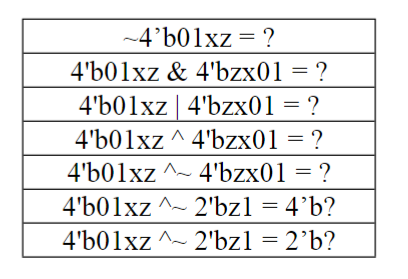
**Result of 2'b00 && 2'b1z: 0**

**Result of 2'b0z || 4'b01xz: 1**

**design.sv:48: $finish called at 0 (1s)**

**Done**

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

****

module test\_operations;

reg [3:0] A, B, C;

reg [1:0] D;

initial begin

A = 4'b01xz;

B = 4'bzx01;

D = 2'bz1;

// Test bitwise operations

$display("~4'b01xz = %b", ~A);

$display("4'b01xz & 4'bzx01 = %b", A & B);

$display("4'b01xz | 4'bzx01 = %b", A | B);

$display("4'b01xz ^ 4'bzx01 = %b", A ^ B);

$display("4'b01xz ~^ 4'bzx01 = %b", A ~^ B);

// Test XNOR and XOR with 2'bz1

$display("4'b01xz ~^ 2'bz1 = %b", A ~^ D);

$display("4'b01xz ^ 2'bz1 = %b", A ^ D);

$finish;

end

endmodule

**Results**

**[2024-10-04 08:04:57 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out**

**~4'b01xz = 10xx**

**4'b01xz & 4'bzx01 = 0x0x**

**4'b01xz | 4'bzx01 = x1x1**

**4'b01xz ^ 4'bzx01 = xxxx**

**4'b01xz ~^ 4'bzx01 = xxxx**

**4'b01xz ~^ 2'bz1 = 10xx**

**4'b01xz ^ 2'bz1 = 01xx**

**design.sv:21: $finish called at 0 (1s)**

**Done**

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

module testBitwiseOperations;

// Declare a 4-bit register with unknown values

reg [3:0] input\_value = 4'b01xz; // 4'b01xz

reg and\_result;

reg nor\_result;

reg xor\_result;

reg xnor\_result;

initial begin

// Perform bitwise AND operation

and\_result = &input\_value; // AND operation

// Perform bitwise NOR operation

nor\_result = ~|input\_value; // NOR operation

// Perform bitwise XOR operation

xor\_result = ^input\_value; // XOR operation

// Perform bitwise XNOR operation

xnor\_result = ~^input\_value; // XNOR operation

// Display the results

$display("Result of & 4'b01xz: %b", and\_result);

$display("Result of ~| 4'b01xz: %b", nor\_result);

$display("Result of ^ 4'b01xz: %b", xor\_result);

$display("Result of ~^ 4'b01xz: %b", xnor\_result);

// End the simulation

$finish;

end

endmodule

**Results**

[2024-10-03 22:10:32 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out

Result of & 4'b01xz: 0

Result of ~| 4'b01xz: 0

Result of ^ 4'b01xz: x

Result of ~^ 4'b01xz: x

design.sv:30: $finish called at 0 (1s)

Done

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

module testBitShiftOperations;

// Declare a 4-bit register with unknown values

reg [3:0] input\_value = 4'b01xz; // 4'b01xz

reg [3:0] left\_shift\_result;

reg [3:0] right\_shift\_result;

// Declare shift amounts

reg [1:0] left\_shift\_amount = 1'bz; // Left shift amount (unknown)

reg [1:0] right\_shift\_amount = 2'bxx; // Right shift amount (unknown)

initial begin

// Perform bitwise left shift operation

left\_shift\_result = input\_value << left\_shift\_amount;

// Perform bitwise right shift operation

right\_shift\_result = input\_value >> right\_shift\_amount;

// Display the results

$display("Result of 4'b01xz << 1'bz: %b", left\_shift\_result);

$display("Result of 4'b01xz >> 2'bxx: %b", right\_shift\_result);

// End the simulation

$finish;

end

endmodule

**Results**

**[2024-10-03 22:14:19 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out**

**Result of 4'b01xz << 1'bz: xxxx**

**Result of 4'b01xz >> 2'bxx: xxxx**

**design.sv:24: $finish called at 0 (1s)**

**Done**

1. **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.**

module test;

reg [1:0] B1;

reg [2:0] B2;

reg [3:0] A;

initial begin

// Case 1: B = 2'b1x

B1 = 2'b1x;

A = B1 ? 4'b1100 : 5'b11ZX0;

$display("B = 2'b1x, A = %b", A); // Display the result

// Case 2: B = 3'b1xz

B2 = 3'b1xz;

A = B2 ? 4'b1100 : 5'b11ZX0;

$display("B = 3'b1xz, A = %b", A); // Display the result

$finish;

end

endmodule

**Results**

**[2024-10-04 08:21:22 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out**

**B = 2'b1x, A = 1100**

**B = 3'b1xz, A = 1100**

**design.sv:17: $finish called at 0 (1s)**

**Done**

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

**module testStrength1(); ... ... // 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 = ..., a =... , b = ...”, y, a, b);**

**end**

**endmodule**

**module testStrength2(); ... ... // 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 = x;**

**i1 = 0;**

**i1 = 1;**

**$display(“y = ...”);**

**end**

**endmodule**

**Module 1**

module testStrength1();

reg a, b;

wire y; // Declaring the wire with driven signals

// Define buffers with different strengths

buf (strong1, weak0) g1 (y, a); // buf with strong1, weak0

buf (weak1, strong0) g2 (y, b); // buf with weak1, strong0

initial begin

// Set initial values for a and b

a = 1;

b = 1;

// Display the values of y, a, and b

#1 $display("y = %b, a = %b, b = %b", y, a, b); // Delay to allow signal propagation

end

endmodule

**Results**

**[2024-10-04 08:12:17 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out**

**y = 1, a = 1, b = 1**

**Done**

**There are two buf (buffer) gates connected to the same output (y).**

**The first buffer g1 drives y using input a with strong strength (called strong1 for 1 and weak0 for 0).**

**The second buffer g2 drives y using input b with weaker strength (called weak1 for 1 and strong0 for 0).**

**Since g1 has stronger strength than g2, and both a and b are 1, the output y will be 1, because g1 dominates.**

**The result will be: y = 1 (since the stronger buffer drives the value of y), and both a and b are 1.**

**Module 2**

module testStrength2();

reg i1, i2, ctrl; // Declare the inputs and control signal

wire y; // Declare the output

// Define bufif0 with different strengths

bufif0 (strong1, weak0) g1 (y, i1, ctrl); // Buffer with strong1, weak0 control

bufif0 (strong1, weak0) g2 (y, i2, ctrl); // Buffer with strong1, weak0 control

initial begin

// Initialize inputs

ctrl = 1'bx; // Unknown control

i1 = 0;

i2 = 1;

#1 $display("y = %b, i1 = %b, i2 = %b, ctrl = %b", y, i1, i2, ctrl); // Delay for signal propagation

end

endmodule

**Results**

**[2024-10-04 08:13:49 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out**

**y = x, i1 = 0, i2 = 1, ctrl = x**

**Done**

**Two bufif0 (tri-state buffer) gates are used to drive the output y.**

**The control signal (ctrl) determines whether the input signals (i1 and i2) are allowed to drive the output y.**

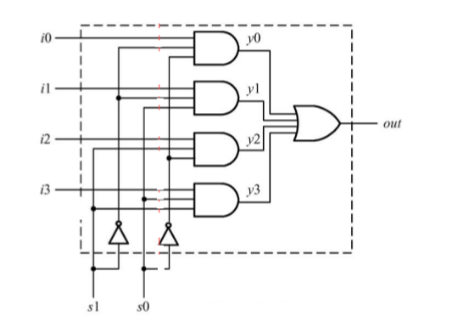
**When ctrl is 0, the corresponding buffer will connect its input to y. If ctrl is 1, the buffer does not affect y, and y is in a high impedance state (z). If ctrl is x, the behavior becomes uncertain.**

**when ctrl = x (unknown), the tri-state buffers will not definitively connect either i1 or i2 to the output y. The output is thus uncertain.**

**Since i1 is 0 and i2 is 1, but ctrl is x, y takes on the value of x (unknown), reflecting that the output cannot be reliably determined.**

**The result will be: y = x (unknown), since the control signal (ctrl) is not a clear 0, preventing the buffers from driving a definitive value to y. The outputs for i1 and i2 are also displayed as 0 and 1, respectively.**

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



**4-to-1 Multiplexer Module (fourOneMux.v)**

module fourOneMux(

input i0\_i, // Input 0

input i1\_i, // Input 1

input i2\_i, // Input 2

input i3\_i, // Input 3

input s0\_i, // Select line 0

input s1\_i, // Select line 1

output out\_o // Output

);

wire not\_s0, not\_s1;

wire y0, y1, y2, y3;

// Inverting select lines

not (not\_s0, s0\_i);

not (not\_s1, s1\_i);

// AND gates for each input

and (y0, i0\_i, not\_s1, not\_s0); // y0 = i0 and not s1 and not s0

and (y1, i1\_i, not\_s1, s0\_i); // y1 = i1 and not s1 and s0

and (y2, i2\_i, s1\_i, not\_s0); // y2 = i2 and s1 and not s0

and (y3, i3\_i, s1\_i, s0\_i); // y3 = i3 and s1 and s0

// OR gate for the output

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

endmodule

**Testbench Module (fourOneMuxTB.v)**

module fourOneMuxTB;

// Inputs

reg i0\_i, i1\_i, i2\_i, i3\_i; // Data inputs

reg s0\_i, s1\_i; // Select lines

wire out\_o; // Output

// Instantiate the 4-to-1 MUX

fourOneMux uut (

.i0\_i(i0\_i),

.i1\_i(i1\_i),

.i2\_i(i2\_i),

.i3\_i(i3\_i),

.s0\_i(s0\_i),

.s1\_i(s1\_i),

.out\_o(out\_o)

);

initial begin

// Monitor the inputs and outputs

$monitor("Time = %0t | i0 = %b, i1 = %b, i2 = %b, i3 = %b, s0 = %b, s1 = %b, out = %b",

$time, i0\_i, i1\_i, i2\_i, i3\_i, s0\_i, s1\_i, out\_o);

// Initialize inputs

i0\_i = 0; i1\_i = 0; i2\_i = 0; i3\_i = 0; s0\_i = 0; s1\_i = 0;

// Test all cases

// Test case 1: Select i0

i0\_i = 1; i1\_i = 0; i2\_i = 0; i3\_i = 0; s0\_i = 0; s1\_i = 0; #10;

// Test case 2: Select i1

i0\_i = 0; i1\_i = 1; i2\_i = 0; i3\_i = 0; s0\_i = 1; s1\_i = 0; #10;

// Test case 3: Select i2

i0\_i = 0; i1\_i = 0; i2\_i = 1; i3\_i = 0; s0\_i = 0; s1\_i = 1; #10;

// Test case 4: Select i3

i0\_i = 0; i1\_i = 0; i2\_i = 0; i3\_i = 1; s0\_i = 1; s1\_i = 1; #10;

// End of the simulation

$finish;

end

endmodule

**Results**

**[2024-10-03 21:32:17 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out**

**Time = 0 | i0 = 1, i1 = 0, i2 = 0, i3 = 0, s0 = 0, s1 = 0, out = 1**

**Time = 10 | i0 = 0, i1 = 1, i2 = 0, i3 = 0, s0 = 1, s1 = 0, out = 1**

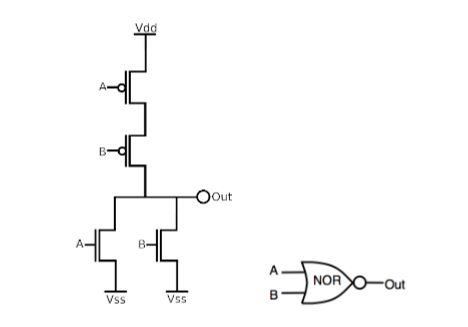
**Time = 20 | i0 = 0, i1 = 0, i2 = 1, i3 = 0, s0 = 0, s1 = 1, out = 1**

**Time = 30 | i0 = 0, i1 = 0, i2 = 0, i3 = 1, s0 = 1, s1 = 1, out = 1**

**testbench.sv:41: $finish called at 40 (1s)**

**Done**

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



**norGate.v**

module norGate(

input A, // Input A

input B, // Input B

output Out // Output

);

supply1 Vdd; // Positive power supply

supply0 Vss; // Ground

wire nA; // Internal wire for NMOS connection

// PMOS transistors

pmos P1(Out, Vdd, A); // PMOS connected to A

pmos P2(Out, Vdd, B); // PMOS connected to B

// NMOS transistors

nmos N1(Out, nA, A); // NMOS connected to A

nmos N2(nA, Vss, B); // NMOS connected to B

endmodule

**norGateTB.v**

module norGateTB;

reg A, B; // Inputs

wire Out; // Output

// Instantiate the NOR gate module

norGate uut (

.A(A),

.B(B),

.Out(Out)

);

initial begin

// Monitor changes in A, B, and Out

$monitor("Time = %0t, A = %b, B = %b, Out = %b", $time, A, B, Out);

// Test case 1: A = 0, B = 0 (Expected Out = 1)

A = 0; B = 0;

#10;

// Test case 2: A = 0, B = 1 (Expected Out = 0)

A = 0; B = 1;

#10;

// Test case 3: A = 1, B = 0 (Expected Out = 0)

A = 1; B = 0;

#10;

// Test case 4: A = 1, B = 1 (Expected Out = 0)

A = 1; B = 1;

#10;

$finish; // End the simulation

end

endmodule

**Results**

[2024-10-03 21:48:10 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out

Time = 0, A = 0, B = 0, Out = 1

Time = 10, A = 0, B = 1, Out = 1

Time = 20, A = 1, B = 0, Out = 1

Time = 30, A = 1, B = 1, Out = 0

testbench.sv:33: $finish called at 40 (1s)

Done