San Francisco Bay University

EE461 Verilog-HDL
Homework #2

Due day: 10/4/2024

Instruction:

a. Push answer sheets/source code to Github
b. Please follow the code style rule like programs on handout.
c. Overdue homework submission could not be accepted.
d. Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)

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?
For 1st case, a = 4’b1111 and b = -5’b00010

```verilog
module arithmetic_ops;
    reg [3:0] a;              // Unsigned 4-bit
    reg signed [4:0] b;       // Signed 5-bit
    reg signed [6:0] sum, diff, prod, mod;  // Wider bit-width for results

    initial begin
        // Input values
        a = 4'b1111;                  // 15 (unsigned)
        b = -5'sb00010;               // -2 (signed)

        // Perform arithmetic with explicit casting
        sum = $signed({1'b0, a}) + b;  // Cast a as signed by zero-extending
        diff = $signed({1'b0, a}) - b;
        prod = $signed({1'b0, a}) * b;
        mod = $signed({1'b0, a}) % b;

        // Display results in decimal and binary
        $display("Inputs:");
        $display("a = %0d (binary: %b), b = %0d (binary: %b)", $signed({1'b0, a}), {1'b0, a}, b, b);
        $display("\nOutputs:");
        $display("Sum  = %0d (binary: %b)", sum, sum);
        $display("Diff = %0d (binary: %b)", diff, diff);
        $display("Prod = %0d (binary: %b)", prod, prod);
        $display("Mod  = %0d (binary: %b)", mod, mod);
    end
endmodule
```

Log Share

```
[2024-12-19 11:21:20 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Inputs:
a = 15 (binary: 01111), b = -2 (binary: 11110)

Outputs:
Sum  = 13 (binary: 0001101)
Diff = 17 (binary: 0010001)
Prod = -30 (binary: 1100010)
Mod  = 1 (binary: 0000001)
Done
```

For 2nd case, with b = -5’b01xz

```
module arithmetic_ops_with_x;
    reg [3:0] a;                // Unsigned 4-bit
    reg signed [4:0] b;         // Signed 5-bit
    reg signed [5:0] sum, diff, prod, mod; // 6 bits for operations to handle overflow

    initial begin
        // Inputs
        a = 4'b1111;            // 15 in unsigned
        b = -5'b01xz;           // -5'b01xz (contains unknowns)

        // Perform arithmetic operations
        sum = a + b;
        diff = a - b;
        prod = a * b;
        mod = a % b;

        // Display results
        $display("Inputs: a = %b (%0d), b = %b (%0d)", a, a, b, b);
        $display("Sum = %b (%0d)", sum, sum);
        $display("Diff = %b (%0d)", diff, diff);
        $display("Prod = %b (%0d)", prod, prod);
        $display("Mod = %b (%0d)", mod, mod);
    end
endmodule
```

Log Share

```
[2024-12-19 11:25:48 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && urbuffer vvp a.out
Inputs: a = 1111 (15), b = xxxxx (x)
Sum = xxxxxx (x)
Diff = xxxxxx (x)
Prod = xxxxxx (x)
Mod = xxxxxx (x)
Done
```

b = -5'b01xz contains an x in its binary representation, which represents an unknown state. Since b is unknown, any operation involving b will propagate the x into the results.

2. 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.

Both Cases included here

```
module relational_ops;
    reg [1:0] a; // 2-bit input
    reg [2:0] b; // 3-bit input

    initial begin
        // Case 1: a = 2'b1z, b = 3'b11z
        a = 2'b1z;
        b = 3'b11z;
        $display("Case 1: a=%b, b=%b", a, b);
        $display("a > b: %b, a >= b: %b, a < b: %b, a <= b: %b", (a > b), (a >= b), (a < b), (a <= b));

        // Case 2: a = 4'b01xz
        a = 4'b01xz;
        $display("Case 2: a=%b", a);
        $display("a > b: %b, a >= b: %b, a < b: %b, a <= b: %b", (a > b), (a >= b), (a < b), (a <= b));
    end
endmodule
```

Log Share

```
[2024-12-19 11:33:20 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && urbuffer vvp a.out
Case 1: a=1z, b=11z
a > b: x, a >= b: x, a < b: x, a <= b: x
Case 2: a=xz
a > b: x, a >= b: x, a < b: x, a <= b: x
Done
```

Case 2

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

design.sv

```
module equality_operators;
    reg [3:0] a, b;

    initial begin
        // Example 1: a = 4'b01xz, b = 4'bzx10
        a = 4'b01xz; b = 4'bzx10;
        $display("Example 1: a = %b, b = %b", a, b);
        $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
        // Example 2: a = 4'b01zz, b = 4'b0100
        a = 4'b01zz; b = 4'b0100;
        $display("\nExample 2: a = %b, b = %b", a, b);
        $display("(a === b) = %b", (a === b));
        $display("(a !== b) = %b", (a !== b));
        $display("(a == b)  = %b", (a == b));
        $display("(a != b)  = %b", (a != b));
        // Example 3: a = 4'b01xz, b = 4'b01xz
        a = 4'b01xz; b = 4'b01xz;
        $display("\nExample 3: a = %b, b = %b", a, b);
        $display("(a === b) = %b", (a === b));
        $display("(a !== b) = %b", (a !== b));
        $display("(a == b)  = %b", (a == b));
        $display("(a != b)  = %b", (a != b));
        // Example 4: a = 4'b01zz, b = 4'b01zz
        a = 4'b01zz; b = 4'b01zz;
        $display("\nExample 4: a = %b, b = %b", a, b);
        $display("(a === b) = %b", (a === b));
        $display("(a !== b) = %b", (a !== b));
        $display("(a == b)  = %b", (a == b));
        $display("(a != b)  = %b", (a != b));
    end
endmodule
```

Log Share

```
[2024-12-19 11:47:30 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && urbuffer vvp a.out
Example 1: a = 01xz, b = zx10
(a === b) = 0
(a !== b) = 1
(a == b)  = x
(a != b)  = x

Example 2: a = 01zz, b = 0100
(a === b) = 0
(a !== b) = 1
(a == b)  = x
(a != b)  = x

Example 3: a = 01xz, b = 01xz
(a === b) = 1
(a !== b) = 0
(a == b)  = x
(a != b)  = x

Example 4: a = 01zz, b = 01zz
(a === b) = 1
(a !== b) = 0
(a == b)  = x
(a != b)  = x
Done
```

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

A	!A
1’bx	
1’bz	
2’b1z	
2’b0z	
2’bxz	
3’bxxx	
3’b1xx	
3’b0xx	

```verilog
module not_ops;
    reg [2:0] A; // Define A as 3-bit wide to accommodate all cases

    initial begin
        // Test cases
        A = 1'bx;    $display("A = %b, !A = %b", A, !A); // A = 1'bx
        A = 1'bz;    $display("A = %b, !A = %b", A, !A); // A = 1'bz
        A = 2'b1z;   $display("A = %b, !A = %b", A, !A); // A = 2'b1z
        A = 2'b0z;   $display("A = %b, !A = %b", A, !A); // A = 2'b0z
        A = 2'bxz;   $display("A = %b, !A = %b", A, !A); // A = 2'bxz
        A = 3'bxxx;  $display("A = %b, !A = %b", A, !A); // A = 3'bxxx
        A = 3'b1xx;  $display("A = %b, !A = %b", A, !A); // A = 3'b1xx
        A = 3'b0xx;  $display("A = %b, !A = %b", A, !A); // A = 3'b0xx
    end
endmodule
```

Log Share

```
[2024-12-19 11:52:15 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
A = 00x, !A = x
A = 00z, !A = x
A = 01z, !A = 0
A = 00z, !A = x
A = 0xz, !A = x
A = xxx, !A = x
A = 1xx, !A = 0
A = 0xx, !A = x
Done
```

5. 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 logical_ops;
    reg [1:0] a1, a2;
    reg [2:0] b1;
    reg [3:0] c1;

    initial begin
        // Test Case 1: 1'bx && 2'bxz
        a1 = 1'bx; b1 = 2'bxz;
        $display("Case 1: a1 = %b, b1 = %b, a1 && b1 = %b", a1, b1, (a1 && b1));

        // Test Case 2: 2'b0x || 1'bz
        a2 = 2'b0x; a1 = 1'bz;
        $display("Case 2: a2 = %b, a1 = %b, a2 || a1 = %b", a2, a1, (a2 || a1));

        // Test Case 3: 2'b00 && 2'b1z
        a2 = 2'b00; b1 = 2'b1z;
        $display("Case 3: a2 = %b, b1 = %b, a2 && b1 = %b", a2, b1, (a2 && b1));

        // Test Case 4: 2'b0z || 4'b01xz
        a2 = 2'b0z; c1 = 4'b01xz;
        $display("Case 4: a2 = %b, c1 = %b, a2 || c1 = %b", a2, c1, (a2 || c1));
    end
endmodule
```

Log Share

```
[2024-12-19 11:56:25 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Case 1: a1 = 0x, b1 = 0xz, a1 && b1 = x
Case 2: a2 = 0x, a1 = 0z, a2 || a1 = x
Case 3: a2 = 00, b1 = 01z, a2 && b1 = 0
Case 4: a2 = 0z, c1 = 01xz, a2 || c1 = 1
Done
```

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

~4’b01xz = ?
4'b01xz & 4'bzx01 = ?
4'b01xz | 4'bzx01 = ?
4'b01xz ^ 4'bzx01 = ?
4'b01xz ^~ 4'bzx01 = ?
4'b01xz ^~ 2'bz1 = 4’b?
4'b01xz ^~ 2'bz1 = 2’b?

```
module bitwise_ops;
    reg [3:0] A, B;  // 4-bit registers
    reg [1:0] C;     // 2-bit register
    reg [3:0] extended_result; // Temporary variable for full-width results
    reg [1:0] truncated_result; // Temporary variable for truncated results

    initial begin
        // Inputs
        A = 4'b01xz;
        B = 4'bzx01;
        C = 2'bz1;

        // Perform bitwise operations and display results
        $display("~A = %b", ~A);                          // ~4'b01xz
        $display("A & B = %b", A & B);                    // 4'b01xz & 4'bzx01
        $display("A | B = %b", A | B);                    // 4'b01xz | 4'bzx01
        $display("A ^ B = %b", A ^ B);                    // 4'b01xz ^ 4'bzx01
        $display("A ^~ B = %b", A ^~ B);                  // 4'b01xz ^~ 4'bzx01

        // Compute extended and truncated results
        extended_result = A ^~ {2'b00, C};                // Zero-extend C and perform XNOR
        truncated_result = extended_result[1:0];          // Truncate to 2 bits
        $display("A ^~ C (4-bit result) = %b", extended_result); // Full 4-bit result
        $display("A ^~ C (2-bit result) = %b", truncated_result); // Truncated 2-bit result
    end
endmodule
```

Log Share

```
[2024-12-19 12:03:37 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
~A = 10xx
A & B = 0x0x
A | B = x1x1
A ^ B = xxxx
A ^~ B = xxxx
A ^~ C (4-bit result) = 10xx
A ^~ C (2-bit result) = xx
Done
```

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

```
module reduction_ops;
    reg [3:0] A;

    initial begin
        A = 4'b01xz;

        // Perform reduction operations and display results
        $display("&A = %b", &A);       // Reduction AND
        $display("~|A = %b", ~|A);     // Reduction NOR
        $display("^A = %b", ^A);       // Reduction XOR
        $display("~^A = %b", ~^A);     // Reduction XNOR
    end
endmodule
```

Log Share

```
[2024-12-19 12:09:54 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
&A = 0
~|A = 0
^A = x
~^A = x
Done
```

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

```
module shift_ops;
    reg [3:0] A;
    reg [1:0] B;

    initial begin
        A = 4'b01xz;

        // Left shift with unknown amount
        $display("A << 1'bz = %b", A << 1'bz);

        // Right shift with unknown amount
        B = 2'bxx;
        $display("A >> B = %b", A >> B);
    end
endmodule
```

Log Share

```
[2024-12-19 12:11:38 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
A << 1'bz = xxxx
A >> B = xxxx
Done
```

Any shift operation with an indeterminate shift amount results in x for all bits.

9. 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 conditional_expr;
    reg [3:0] A;
    reg [1:0] B1;
    reg [2:0] B2;

    initial begin
        // Case 1: B = 2'b1x
        B1 = 2'b1x;
        A = B1 ? 4'b1100 : 5'b11ZX0;  // Truncate 5'b11ZX0 to 4 bits
        $display("Case 1: B = %b, A = %b", B1, A);

        // Case 2: B = 3'b1xz
        B2 = 3'b1xz;
        A = B2 ? 4'b1100 : 5'b11ZX0;  // Truncate 5'b11ZX0 to 4 bits
        $display("Case 2: B = %b, A = %b", B2, A);
    end
endmodule
```

Log Share

```
[2024-12-19 12:15:29 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Case 1: B = 1x, A = 1100
Case 2: B = 1xz, A = 1100
Done
```

10. 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 testStrength1();
    reg a, b;      // Declare inputs as registers
    wire y;        // Output is a wire to be driven by the buffers

    // Buffers with strength modifiers
    buf (strong1, weak0) g1 (y, a);
    buf (weak1, strong0) g2 (y, b);

    initial begin
        a = 1;  // Strong 1 from a
        b = 1;  // Weak 1 from b
        #1;     // Allow signals to resolve
        $display("y = %b, a = %b, b = %b", y, a, b);
    end
endmodule
```

Log Share

```
[2024-12-19 12:26:32 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
y = 1, a = 1, b = 1
Done
```

EXPLANATION OF RESULTS

- Buffers with Strengths:
  - `a` drives `y` with strong 1.
  - `b` drives `y` with weak 1.
- Conflict Resolution
  - When multiple drivers control `y`, the stronger signal dominates.
  - `a`'s strong 1 overrides `b`'s weak 1.
- Output Summary:
  - `y = 1` (strong driver from `a`), `a = 1`, `b = 1`.

```
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 testStrength2();
    reg ctrl, i1, i2;  // Inputs and control signal
    wire y;            // Output is a wire

    // Tristate buffers with strength modifiers
    bufif0 (strong1, weak0) g1 (y, i1, ctrl);
    bufif0 (strong1, weak0) g2 (y, i2, ctrl);

    initial begin
        ctrl = 1'bx;  // Control is unknown
        i1 = 0;       // First input is 0
        i2 = 1;       // Second input is 1
        #1;           // Allow signals to resolve
        $display("y = %b, ctrl = %b, i1 = %b, i2 = %b", y, ctrl, i1, i2);
    end
endmodule
```

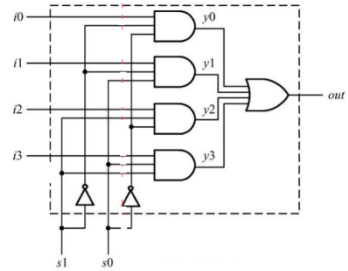
Log Share

```
[2024-12-19 12:21:13 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
y = x, ctrl = x, i1 = 0, i2 = 1
Done
```

EXPLANATION OF RESULTS

- Control Signal (ctrl = x): An unknown ctrl makes the tri-state buffers' behavior ambiguous, resulting in indeterminate output (x).
- Buffer Behavior: With ctrl = x, Verilog cannot decide between driving (0) or disabling (z), leaving y unresolved.
- Inputs (i1 = 0, i2 = 1):
  - Defined inputs cannot influence y when ctrl is indeterminate.
- Output (y = x):
  - The unresolved control signal propagates as an indeterminate output (x).

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




```
module fourOneMux(
                    i0_i,
                    i1_i,
                    i2_i,
                    i3_i,
                    s0_i,
                    s1_i,
                    out_o
);
                ... ... ;
endmodule

`include "foutOneMux"
module fourOneMuxTB;
                ... ... ;
endmodule
```

Note: please save two modules in two different files with the same name as module under the one directory.

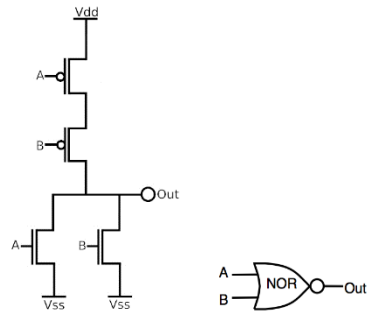
```verilog
// Testbench Module (fourOneMuxTB)
module fourOneMuxTB;

    // Declare inputs as regs and output as wire
    reg i0, i1, i2, i3;     // MUX inputs
    reg s0, s1;             // Select lines
    wire out;               // MUX output

    // Instantiate the multiplexer
    fourOneMux uut (
        .i0_i(i0),
        .i1_i(i1),
        .i2_i(i2),
        .i3_i(i3),
        .s0_i(s0),
        .s1_i(s1),
        .out_o(out)
    );

    // Testbench logic
    initial begin
        // Display header
        $display("s1 s0 | i3 i2 i1 i0 | out");

        // Apply test cases
        {i3, i2, i1, i0} = 4'b1000; {s1, s0} = 2'b00;
#10; $display("%b  %b  |  %b  |  %b", s1, s0, {i3, i2, i1, i0}, out);
        {i3, i2, i1, i0} = 4'b1000; {s1, s0} = 2'b01;
#10; $display("%b  %b  |  %b  |  %b", s1, s0, {i3, i2, i1, i0}, out);
        {i3, i2, i1, i0} = 4'b1000; {s1, s0} = 2'b10;
#10; $display("%b  %b  |  %b  |  %b", s1, s0, {i3, i2, i1, i0}, out);
        {i3, i2, i1, i0} = 4'b1000; {s1, s0} = 2'b11;
#10; $display("%b  %b  |  %b  |  %b", s1, s0, {i3, i2, i1, i0}, out);

        $finish;
    end
endmodule
```

```verilog
// Multiplexer Module (fourOneMux)
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
);

    // Internal wires
    wire not_s0, not_s1;
    wire y0, y1, y2, y3;

    // Generate inverted select lines
    not u1(not_s0, s0_i);
    not u2(not_s1, s1_i);

    // AND gates for each input
    and g0(y0, i0_i, not_s1, not_s0);  // y0 = i0 & ~s1 & ~s0
    and g1(y1, i1_i, not_s1, s0_i);   // y1 = i1 & ~s1 & s0
    and g2(y2, i2_i, s1_i, not_s0);   // y2 = i2 & s1 & ~s0
    and g3(y3, i3_i, s1_i, s0_i);     // y3 = i3 & s1 & s0

    // OR gate to combine outputs
    or g4(out_o, y0, y1, y2, y3);     // out_o = y0 | y1 | y2 | y3

endmodule
```

Log Share

```
[2024-12-19 12:47:18 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
s1 s0 | i3 i2 i1 i0 | out
0  0  |  1000  |  0
0  1  |  1000  |  0
1  0  |  1000  |  0
1  1  |  1000  |  1
testbench.sv:31: $finish called at 40 (1s)
Done
```

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




```verilog
module nor_gate_tb;

    // Inputs and output
    reg A, B;          // Test inputs
    wire Out;          // NOR gate output

    // Instantiate the NOR gate
    nor_gate uut (
        .A(A),
        .B(B),
        .Out(Out)
    );

    // Testbench logic
    initial begin
        $display("A B | Out");
        $monitor("%b %b | %b", A, B, Out);

        // Apply test cases
        A = 0; B = 0; #10;  // NOR(0,0) = 1
        A = 0; B = 1; #10;  // NOR(0,1) = 0
        A = 1; B = 0; #10;  // NOR(1,0) = 0
        A = 1; B = 1; #10;  // NOR(1,1) = 0

        $finish;
    end

endmodule
```

```verilog
module nor_gate (
    input A,       // Input A
    input B,       // Input B
    output Out     // Output of NOR gate
);

    // Internal wires
    wire w_pmos, w_nmos;

    // PMOS devices
    pmos p1(w_pmos, 1'b1, A);  // PMOS with source connected to Vdd
    pmos p2(Out, w_pmos, B);   // PMOS with intermediate wire

    // NMOS devices
    nmos n1(Out, 1'b0, A);     // NMOS with source connected to GND
    nmos n2(Out, 1'b0, B);     // NMOS with source connected to GND

endmodule
```

Log Share

```
[2024-12-19 12:55:30 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && urbuffer vvp a.out
A B | Out
0 0 | 1
0 1 | 0
1 0 | 0
1 1 | 0
testbench.sv:25: $finish called at 40 (1s)
Done
```