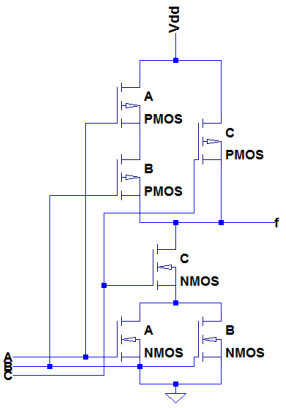
San Francisco Bay University

EE461 Verilog-HDL
Homework #3

1. Design a static logic gate f =$\overline{(a+b)c}$ in Verilog modeling by PMOS/NMOS devices based on the logic schematic as follows. And then write the testbench to verify the design to cover all input combinations and some x/z.



Module for static logic gate

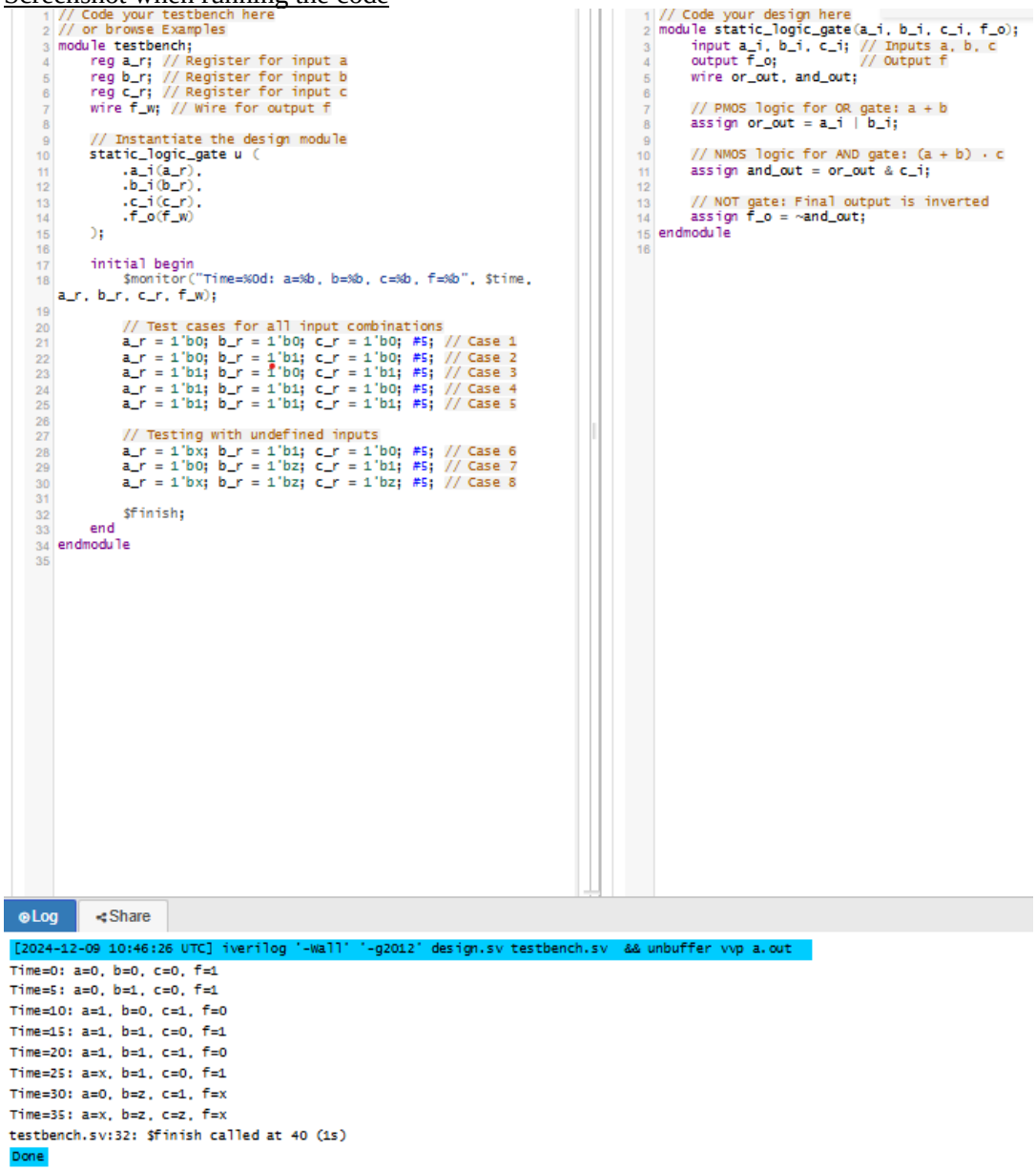
```
module static_logic_gate(a_i, b_i, c_i, f_o);
    input a_i, b_i, c_i; // Inputs a, b, c
    output f_o;          // Output f
    wire or_out, and_out;

    // PMOS logic for OR gate: a + b
    assign or_out = a_i | b_i;

    // NMOS logic for AND gate: (a + b) · c
    assign and_out = or_out & c_i;

    // NOT gate: Final output is inverted
    assign f_o = ~and_out;
endmodule
```

Module for testbench

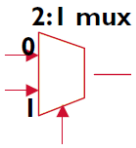
```
module testbench;
    reg a_r; // Register for input a
    reg b_r; // Register for input b
    reg c_r; // Register for input c
    wire f_w; // Wire for output f

    // Instantiate the design module
    static_logic_gate u (
        .a_i(a_r),
        .b_i(b_r),
        .c_i(c_r),
        .f_o(f_w)
    );
```

```
    initial begin
        $monitor("Time=%0d: a=%b, b=%b, c=%b, f=%b", $time, a_r,
b_r, c_r, f_w);

        // Test cases for all input combinations
        a_r = 1'b0; b_r = 1'b0; c_r = 1'b0; #5; // Case 1
        a_r = 1'b0; b_r = 1'b1; c_r = 1'b0; #5; // Case 2
        a_r = 1'b1; b_r = 1'b0; c_r = 1'b1; #5; // Case 3
        a_r = 1'b1; b_r = 1'b1; c_r = 1'b0; #5; // Case 4
        a_r = 1'b1; b_r = 1'b1; c_r = 1'b1; #5; // Case 5

        // Testing with undefined inputs
        a_r = 1'bx; b_r = 1'b1; c_r = 1'b0; #5; // Case 6
        a_r = 1'b0; b_r = 1'bz; c_r = 1'b1; #5; // Case 7
        a_r = 1'bx; b_r = 1'bz; c_r = 1'bz; #5; // Case 8

        $finish;
    end
endmodule
```

Screenshot when running the code



2. Design a 2-to-1 mux UDP, and write testbench to assign selection bit to 0, 1 and X/Z to verify you design.



Screenshot

testbench.sv

```
module testbench;
    reg a, b, sel; // Inputs
    wire out;      // Output

    // Instantiate UDP
    mux_2to1 uut (out, a, b, sel);

    initial begin
        $monitor("Time=%0d: a=%b, b=%b, sel=%b, out=%b",
$time, a, b, sel, out);

        // Test Cases
        a = 0; b = 1; sel = 0; #5;  // Select a
        a = 1; b = 0; sel = 1; #5;  // Select b
        a = 0; b = 1; sel = 1; #5;  // Select b
        a = 1; b = 0; sel = 0; #5;  // Select a
        a = 1; b = 1; sel = 1'bx; #5; // Undefined
select
        a = 0; b = 0; sel = 1'bz; #5; // Undefined
select

        $finish;
    end
endmodule
```

design.sv

```
primitive mux_2to1(out, a, b, sel);
    output out;
    input a, b, sel;

    table
        0  ?   0   : 0;
        1  ?   0   : 1;
        ?  0   1   : 0;
        ?  1   1   : 1;
        0  ?   x   : x;
        1  ?   x   : x;

    endtable
endprimitive
```

Log Share

```
[2024-12-10 05:19:40 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Time=0: a=0, b=1, sel=0, out=0
Time=5: a=1, b=0, sel=1, out=0
Time=10: a=0, b=1, sel=1, out=1
Time=15: a=1, b=0, sel=0, out=1
Time=20: a=1, b=1, sel=x, out=x
Time=25: a=0, b=0, sel=z, out=x
testbench.sv:19: $finish called at 30 (1s)
Done
```

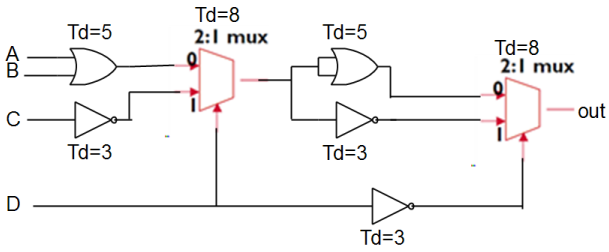
Code for design module

```
primitive mux_2to1(out, a, b, sel);
    output out;
    input a, b, sel;

    table
        0  ?   0   : 0;
        1  ?   0   : 1;
        ?  0   1   : 0;
        ?  1   1   : 1;
        0  ?   x   : x;
        1  ?   x   : x;

    endtable
endprimitive
```

Testbench

```
module testbench;
    reg a, b, sel; // Inputs
    wire out;      // Output

    // Instantiate UDP
    mux_2to1 uut (out, a, b, sel);

    initial begin
        $monitor("Time=%0d: a=%b, b=%b, sel=%b, out=%b", $time, a,
b, sel, out);

        // Test Cases
        a = 0; b = 1; sel = 0; #5;  // Select a
        a = 1; b = 0; sel = 1; #5;  // Select b
        a = 0; b = 1; sel = 1; #5;  // Select b
        a = 1; b = 0; sel = 0; #5;  // Select a
        a = 1; b = 1; sel = 1'bx; #5; // Undefined select
        a = 0; b = 0; sel = 1'bz; #5; // Undefined select

        $finish;
    end
endmodule
```

3. Create top module to instantiate mux UDPs from the above primitive and system gates with delay time as below and simulate it by testbench with only several inputs values. What is the delay time of longest path from inputs to output in hand calculation if we don't consider the delay from the selection bit to output in two muxes? If *D* is either *1* or *0*, what is the possible delay time of longest path from the inputs to output?



Screenshot

```
module testbench;
    reg A, B, C, D; // Inputs
    wire out;       // Output

    // Instantiate Top Module
    top_module uut (.A(A), .B(B), .C(C), .D(D),
.out(out));

    initial begin
        $monitor("Time=%0d: A=%b, B=%b, C=%b, D=%b,
out=%b", $time, A, B, C, D, out);

        // Test Cases
        A = 0; B = 0; C = 0; D = 0; #10;
        A = 1; B = 0; C = 1; D = 1; #10;
        A = 0; B = 1; C = 1; D = 0; #10;
        A = 1; B = 1; C = 0; D = 1; #10;
        $finish;
    end
endmodule
```

```
module top_module(input A, B, C, D, output out);
    wire or_out1, not_out1, mux1_out, or_out2, not_out2,
not_out3;

    // First Layer Gates
    assign or_out1 = A | B;         // OR Gate 1 (Td = 5)
    assign not_out1 = ~C;           // NOT Gate 1 (Td = 3)

    // First MUX
    mux_2to1 mux1 (mux1_out, or_out1, not_out1, D); // Td = 8

    // Second Layer Gates
    assign or_out2 = mux1_out | mux1_out; // OR Gate 2 (Td =
5)
    assign not_out2 = ~mux1_out;          // NOT Gate 2 (Td =
3)

    // Third Layer NOT Gate
    assign not_out3 = ~D; // NOT Gate 3 for D inversion (Td =
3)

    // Second MUX
    mux_2to1 mux2 (out, or_out2, not_out2, not_out3); // Td =
8
endmodule

// Include the UDP definition
primitive mux_2to1(out, a, b, sel);
    output out;
    input a, b, sel;

    table
        0  ?   0   : 0;
        1  ?   0   : 1;
        ?  0   1   : 0;
        ?  1   1   : 1;
        0  ?   x   : x;
        1  ?   x   : x;

    endtable
endprimitive
```

```
Log  Share
[2024-12-10 07:32:58 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Time=0: A=0, B=0, C=0, D=0, out=1
Time=10: A=1, B=0, C=1, D=1, out=0
Time=20: A=0, B=1, C=1, D=0, out=0
Time=30: A=1, B=1, C=0, D=1, out=1
testbench.sv:16: $finish called at 40 (1s)
Done
```

Module

```
module top_module(input A, B, C, D, output out);
    wire or_out1, not_out1, mux1_out, or_out2, not_out2, not_out3;

    // First Layer Gates
    assign or_out1 = A | B;         // OR Gate 1 (Td = 5)
    assign not_out1 = ~C;           // NOT Gate 1 (Td = 3)

    // First MUX
    mux_2to1 mux1 (mux1_out, or_out1, not_out1, D); // Td = 8

    // Second Layer Gates
    assign or_out2 = mux1_out | mux1_out; // OR Gate 2 (Td = 5)
    assign not_out2 = ~mux1_out;          // NOT Gate 2 (Td = 3)

    // Third Layer NOT Gate
    assign not_out3 = ~D; // NOT Gate 3 for D inversion (Td = 3)

    // Second MUX
    mux_2to1 mux2 (out, or_out2, not_out2, not_out3); // Td = 8
endmodule

// Include the UDP definition
primitive mux_2to1(out, a, b, sel);
    output out;
    input a, b, sel;

    table
        0  ?   0   : 0;
        1  ?   0   : 1;
        ?  0   1   : 0;
        ?  1   1   : 1;
        0  ?   x   : x;
        1  ?   x   : x;

    endtable
endprimitive
```

Testbench

```
module testbench;
    reg A, B, C, D; // Inputs
    wire out;       // Output

    // Instantiate Top Module
    top_module uut (.A(A), .B(B), .C(C), .D(D), .out(out));

    initial begin
        $monitor("Time=%0d: A=%b, B=%b, C=%b, D=%b, out=%b",
$time, A, B, C, D, out);

        // Test Cases
        A = 0; B = 0; C = 0; D = 0; #10;
        A = 1; B = 0; C = 1; D = 1; #10;
```

```
        A = 0; B = 1; C = 1; D = 0; #10;
        A = 1; B = 1; C = 0; D = 1; #10;
        $finish;
    end
endmodule
```

Longest Path Delay from Inputs to Output (Excluding Selection Delay in MUXes)

Path with the Highest Delay:

- Longest path: A/B→OR1→MUX1→OR2→MUX2→OUT
- Delay contributions:
    - OR1: Td=5
    - MUX1: Td=8
    - OR2: Td=5
    - MUX2: Td=8
- Total Delay:

Ttotal=5+8+5+8=26 time units

Longest Path Delay Considering D=1or D=0

- The select signal D (or its inversion ~D) is not included in the delay calculation
- The delay remains the same: Ttotal=26 time units, regardless of whether D=1or D=0

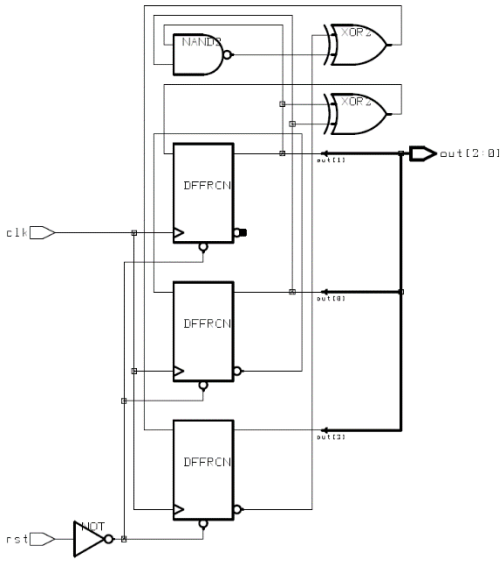
4. Fix the bugs in the following example primitive, and explain why.

```
#include "basicPrimitive.v"
primitive example(a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r);
    output[1:0]     a, r;
    input           b, d, e, f, g, h, i, j, k, l, m, n, o, p, q;
    inout           c;

    reg[1:0]        a, r;

    table
    //a: c:  r:     b  d   e  f  g  h  i  j  k  l  m  n  o  p  q
   00: 1: 01:       0  1   0  1  0  1  0  1  0  1  0  1  0  1  0
    1: 1: 11:       0  1   0  1  0  1  0  1  0  1  0  1  0  1  ?
   00: 0: 01:       0  1   0  1  0  1  0  1  0  1  0  1  0  1  0
      ... ...
    endtable

    basicPrimitive  u0(a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r);

endprimitive
```

Corrected code

```
//The #include "basicPrimitive.v" directive is removed from the solution because Verilog does not
//natively support #include directives, unlike C/C++. If basicPrimitive.v contains required
//primitives or modules, the primitive definition should be defined directly in the file.

primitive example(output [1:0] a, r,
            input b, d, e, f, g, h, i, j, k, l, m, n, o, p, q);
//The output [1:0] a, r remains as the only declaration for a and r. Removed reg[1:0] as primitives
//do not support internal state
// Removed the c signal as inout since primitives do not support bidirectional signals, they only
accept input and output.

   table
//Table Formatting
//Properly aligned inputs and outputs for clarity
//Removed misplaced colons (:) and separated inputs and outputs with a single colon (:)

// a  r  b  d  e  f  g  h  i  j  k  l  m  n  o  p  q : a  r
   00 01 0  1  0  1  0  1  0  1  0  1  0  1  0  1  0 : 00 01;
   01 11 0  1  0  1  0  1  0  1  0  1  0  1  0  1  ? : 01 11;
   00 01 0  1  0  1  0  1  0  1  0  1  0  1  0  1  0 : 00 01;
    endtable
//Table Logic now reflects a valid mapping of inputs to outputs
//Each row defines Input combinations for b,d,e,…,q and corresponding outputs a,r

//Removed incorrect basicPrimitive u0(...) line since primitives are not instantiated like modules,
//they are directly invoked within a design

endprimitive
```

5. Generate UDP of D-Flip Flop with reset, and then build 3-bits counter in top module by instantiating UDP of DFF and NOT/NAND/XOR gates in terms of following schematic. After that, write testbench to observe output results.



Module

```
module DFFRCN(output reg q, input d, clk, rst);
    always @(posedge clk or negedge rst) begin
        if (!rst)
            q <= 0;  // Reset the flip-flop
        else
            q <= d;  // Update q with d on the rising clock edge
    end
endmodule

module counter_3bit(input clk, rst, output [2:0] out);
    wire q0, q1, q2;        // Outputs of the flip-flops
    wire nand_out, xor1_out, xor2_out; // Intermediate signals
    wire not_out;           // Inverted reset signal

    // NOT Gate for reset
    assign not_out = ~rst;

    // Flip-flops
    DFFRCN dff0(q0, xor1_out, clk, not_out);
    DFFRCN dff1(q1, xor2_out, clk, not_out);
    DFFRCN dff2(q2, nand_out, clk, not_out);

    // Logic gates
    assign nand_out = ~(q1 & q2);  // NAND gate
    assign xor1_out = q0 ^ q1;     // XOR gate for flip-flop 0
    assign xor2_out = q1 ^ q2;     // XOR gate for flip-flop 1

    // Outputs
    assign out = {q2, q1, q0}; // Combine the flip-flop outputs
endmodule
```

Testbench

```
module testbench;
    reg clk, rst;      // Clock and reset inputs
    wire [2:0] out;    // 3-bit output of the counter

    // Instantiate the counter module
    counter_3bit uut (
        .clk(clk),
        .rst(rst),
        .out(out)
    );

    // Clock generation
    always #5 clk = ~clk;  // Toggle clock every 5 time units

    initial begin
        // Monitor the output
        $monitor("Time=%0d | clk=%b | rst=%b | out=%b", $time,
clk, rst, out);

        // Initialize inputs
        clk = 0; rst = 1;

        // Test sequence
        #10 rst = 0;  // Release reset
        #50 rst = 1;  // Assert reset
        #10 rst = 0;  // Release reset again
        #100 $finish; // End simulation
    end
endmodule
```

Screenshot

testbench.sv

```
module testbench;
    reg clk, rst;       // Clock and reset inputs
    wire [2:0] out;     // 3-bit output of the counter

    // Instantiate the counter module
    counter_3bit uut (
        .clk(clk),
        .rst(rst),
        .out(out)
    );

    // Clock generation
    always #5 clk = ~clk;  // Toggle clock every 5 time
units

    initial begin
        // Monitor the output
        $monitor("Time=%0d | clk=%b | rst=%b | out=%b",
$time, clk, rst, out);

        // Initialize inputs
        clk = 0; rst = 1;
```

design.sv

```
module DFFRCN(output reg q, input d, clk, rst);
    always @(posedge clk or negedge rst) begin
        if (!rst)
            q <= 0;  // Reset the flip-flop
        else
            q <= d;  // Update q with d on the rising clock edge
    end
endmodule

module counter_3bit(input clk, rst, output [2:0] out);
    wire q0, q1, q2;        // Outputs of the flip-flops
    wire nand_out, xor1_out, xor2_out; // Intermediate signals
    wire not_out;           // Inverted reset signal

    // NOT Gate for reset
    assign not_out = ~rst;

    // Flip-flops
    DFFRCN dff0(q0, xor1_out, clk, not_out);
    DFFRCN dff1(q1, xor2_out, clk, not_out);
    DFFRCN dff2(q2, nand_out, clk, not_out);

    // Logic gates
```

Log Share

```
[2024-12-10 11:45:28 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
Time=0 | clk=0 | rst=1 | out=000
Time=5 | clk=1 | rst=1 | out=000
Time=10 | clk=0 | rst=0 | out=000
Time=15 | clk=1 | rst=0 | out=100
Time=20 | clk=0 | rst=0 | out=100
Time=25 | clk=1 | rst=0 | out=110
Time=30 | clk=0 | rst=0 | out=110
Time=35 | clk=1 | rst=0 | out=001
Time=40 | clk=0 | rst=0 | out=001
Time=45 | clk=1 | rst=0 | out=101
Time=50 | clk=0 | rst=0 | out=101
Time=55 | clk=1 | rst=0 | out=111
Time=60 | clk=0 | rst=1 | out=000
Time=65 | clk=1 | rst=1 | out=000
Time=70 | clk=0 | rst=0 | out=000
Time=75 | clk=1 | rst=0 | out=100
Time=80 | clk=0 | rst=0 | out=100
Time=85 | clk=1 | rst=0 | out=110
Time=90 | clk=0 | rst=0 | out=110
Time=95 | clk=1 | rst=0 | out=001
Time=100 | clk=0 | rst=0 | out=001
Time=105 | clk=1 | rst=0 | out=101
Time=110 | clk=0 | rst=0 | out=101
Time=115 | clk=1 | rst=0 | out=111
Time=120 | clk=0 | rst=0 | out=111
Time=125 | clk=1 | rst=0 | out=000
Time=130 | clk=0 | rst=0 | out=000
Time=135 | clk=1 | rst=0 | out=100
Time=140 | clk=0 | rst=0 | out=100
Time=145 | clk=1 | rst=0 | out=110
Time=150 | clk=0 | rst=0 | out=110
Time=155 | clk=1 | rst=0 | out=001
Time=160 | clk=0 | rst=0 | out=001
Time=165 | clk=1 | rst=0 | out=101
testbench.sv:26: $finish called at 170 (1s)
Time=170 | clk=0 | rst=0 | out=101
Done
```