**Homework 3**

Ashmitha Murali

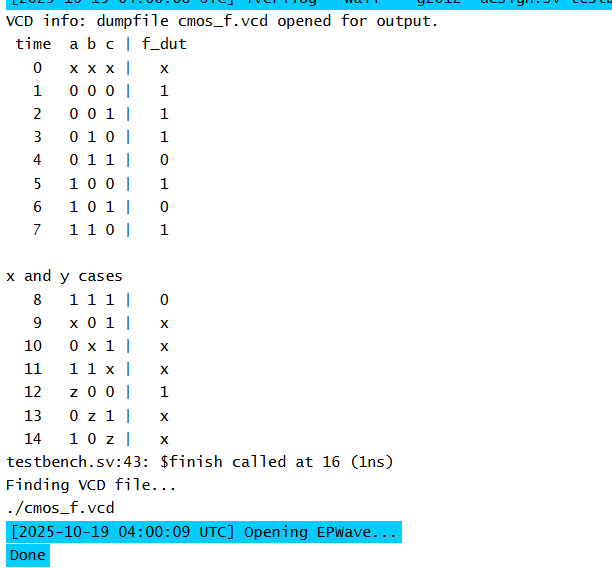
Prof. Alex Yang

Digital Design and HDL EE461

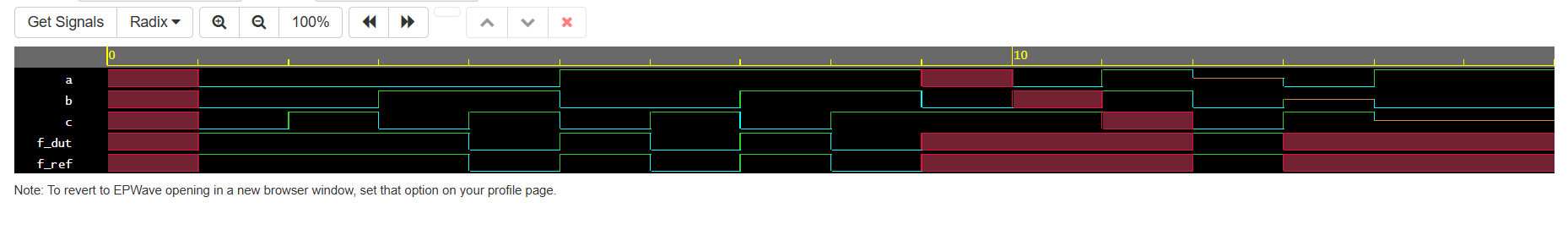
18th October 2025

**Q1.**

**Result:**

****

**Waveform:**

****

**Design Code:**

module cmos\_f\_not\_or\_and (

input wire a,

input wire b,

input wire c,

output wire f

);

supply1 VDD;

supply0 GND;

//pull up

wire p\_mid;

pmos p\_c (f, VDD, c);

pmos p\_a (p\_mid, VDD, a);

pmos p\_b (f, p\_mid, b);

//pull down

wire n\_bot;

nmos n\_a (n\_bot, GND, a);

nmos n\_b (n\_bot, GND, b);

nmos n\_c (f, n\_bot, c);

endmodule

**Testbench Code:**

module tb\_f;

reg a, b, c;

wire f\_dut;

// DUT

cmos\_f\_not\_or\_and dut(.a(a), .b(b), .c(c), .f(f\_dut));

initial begin

// Waveform dump

$dumpfile("cmos\_f.vcd");

$dumpvars(0, tb\_f);

$monitor("%4t %b %b %b | %b",$time,a,b,c,f\_dut);

$display(" time a b c | f\_dut");

#1 a = 0; b = 0; c = 0;

#1 a = 0; b = 0; c = 1;

#1 a = 0; b = 1; c = 0;

#1 a = 0; b = 1; c = 1;

#1 a = 1; b = 0; c = 0;

#1 a = 1; b = 0; c = 1;

#1 a = 1; b = 1; c = 0;

#1 a = 1; b = 1; c = 1;

$display("\nx and y cases");

// Some x/z stimulus to observe 4-state behavior

#1 a = 1'bx; b = 0; c = 1;

#1 a = 0; b = 1'bx; c = 1;

#1 a = 1; b = 1; c = 1'bx;

#1 a = 1'bz; b = 0; c = 0;

#1 a = 0; b = 1'bz; c = 1;

#1 a = 1; b = 0; c = 1'bz;

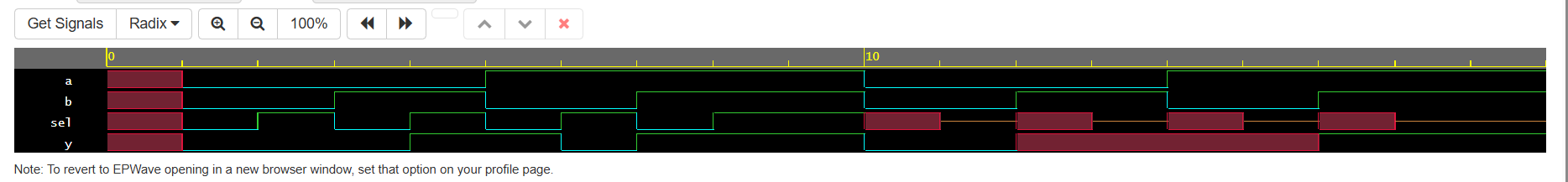
#2 $finish;

end

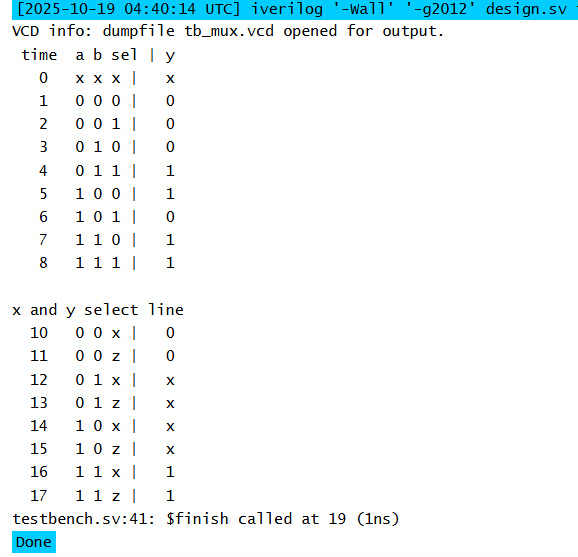
endmodule

**Q2.**

**Waveform:**

****

**Result:**



**Design Code:**

primitive udp\_mux2 (y, a, b, sel);

output y;

input a, b, sel;

table

// a b sel : y

0 ? 0 : 0;

1 ? 0 : 1;

? 0 1 : 0;

? 1 1 : 1;

0 0 x : 0;

1 1 x : 1;

0 1 x : x;

1 0 x : x;

endtable

endprimitive

module mux2\_udp (

input wire a,

input wire b,

input wire sel,

output wire y

);

udp\_mux2 U (y, a, b, sel);

endmodule

**Testbench Code:**

module tb\_mux;

reg a, b, sel;

wire y;

// DUT

mux2\_udp dut(.y(y), .a(a), .b(b), .sel(sel));

initial begin

$dumpfile("tb\_mux.vcd");

$dumpvars(0, tb\_mux);

$monitor("%4t %b %b %b | %b",$time,a,b,sel,y);

$display(" time a b sel | y");

#1 a = 0; b = 0; sel = 0;

#1 a = 0; b = 0; sel = 1;

#1 a = 0; b = 1; sel = 0;

#1 a = 0; b = 1; sel = 1;

#1 a = 1; b = 0; sel = 0;

#1 a = 1; b = 0; sel = 1;

#1 a = 1; b = 1; sel = 0;

#1 a = 1; b = 1; sel = 1;

#1 $display("\nx and y select line");

#1 a = 0; b = 0; sel = 1'bx;

#1 a = 0; b = 0; sel = 1'bz;

#1 a = 0; b = 1; sel = 1'bx;

#1 a = 0; b = 1; sel = 1'bz;

#1 a = 1; b = 0; sel = 1'bx;

#1 a = 1; b = 0; sel = 1'bz;

#1 a = 1; b = 1; sel = 1'bx;

#1 a = 1; b = 1; sel = 1'bz;

#2 $finish;

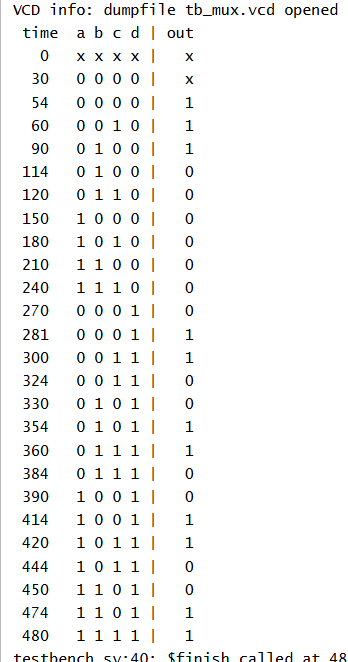
end

endmodule

**Q3.**

The longest path is from or\_gate-> mux->or\_gate->mux which results in longest path of (5+8+5+8 = 26). Hence, the delay time of the longest path is **26ns.**

If we consider that D is either 0 or 1, then the output would go only through one or gate and one not gate. The output would never go through 2 ‘or’ gates and thus the worst case path would be or\_gate-> mux->not\_gate->mux or not\_gate-> mux->or\_gate->mux (5+8+3+8 =24). Hence the time delay when D is 0 or 1 is **24ns.**

**Result:  
**

**Design Code:**

`timescale 1ns/1ns

primitive udp\_mux2 (y, a, b, sel);

output y;

input a, b, sel;

table

// a b sel : y

0 ? 0 : 0;

1 ? 0 : 1;

? 0 1 : 0;

? 1 1 : 1;

0 0 x : 0;

1 1 x : 1;

0 1 x : x;

1 0 x : x;

endtable

endprimitive

module top\_two\_mux\_net (

input wire A, B, C, D,

output wire OUT

);

wire ab\_or, m1\_out, rhs\_inv, rhs\_or, d\_inv, c\_inv;

or #(5) g\_or (ab\_or, A, B);

not #(3) g\_c (c\_inv, C);

udp\_mux2 #(8) M1 (m1\_out, ab\_or, c\_inv, D);

// Right-hand gates

or #(5) g\_or2 (rhs\_or, m1\_out, m1\_out);

not #(3) g\_inv (rhs\_inv, m1\_out);

not #(3) d\_invt (d\_inv, D);

udp\_mux2 #(8) M2 (OUT, rhs\_or, rhs\_inv, d\_inv);

endmodule

**Testbench code:**

`timescale 1ns/1ns

module tb\_mux;

reg a, b, c, d;

wire out;

// DUT

top\_two\_mux\_net dut(.OUT(out), .A(a), .B(b), .C(c), .D(d));

initial begin

// Waveform dump

$dumpfile("tb\_mux.vcd");

$dumpvars(0, tb\_mux);

$monitor("%4t %b %b %b %b | %b",$time,a,b,c,d,out);

$display(" time a b c d | out");

#30 a = 0; b = 0; c = 0; d=0;

#30 a = 0; b = 0; c = 1; d=0;

#30 a = 0; b = 1; c = 0; d=0;

#30 a = 0; b = 1; c = 1; d=0;

#30 a = 1; b = 0; c = 0; d=0;

#30 a = 1; b = 0; c = 1; d=0;

#30 a = 1; b = 1; c = 0; d=0;

#30 a = 1; b = 1; c = 1; d=0;

#30 a = 0; b = 0; c = 0; d=1;

#30 a = 0; b = 0; c = 1; d=1;

#30 a = 0; b = 1; c = 0; d=1;

#30 a = 0; b = 1; c = 1; d=1;

#30 a = 1; b = 0; c = 0; d=1;

#30 a = 1; b = 0; c = 1; d=1;

#30 a = 1; b = 1; c = 0; d=1;

#30 a = 1; b = 1; c = 1; d=1;

#2 $finish;

end

endmodule

**Q4. The primitive has the following issues:**

1. A primitive can’t have multibit outputs. Here, ‘a’ and ‘r’ are defined as 2 bits which is not allowed in UDP.
2. UDP cannot have inout ports. It should either be input or output. Here ‘c’ is defined as inout which is wrong.
3. We should not instantiate anything inside a UDP. We should not use the ‘basicPrimitive’ block inside UDP.
4. The table syntax is invalid. The table should have inputs: outputs. Here the table order is wrong.

**Corrected UDP:**primitive example(r, c, b, d, e, f, g, h, i, j, k, l, m, n, o, p, q);

output r; // single-bit only

input c, b, d, e, f, g, h, i, j, k, l, m, n, o, p, q;

// Combinational table: inputs (in the \*declared\* order) : output ;

table

// c b d e f g h i j k l m n o p q : r

0 ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? : 0;

1 ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? : 1;

x ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? : x;

z ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? : x;

endtable

endprimitive

**Q5.**

**Design and primitive code:**primitive udp\_dff\_async\_rstn (q, d, clk, rst\_n);

output q;

input d, clk, rst\_n;

reg q;

table

// d clk rst\_n : q(t) : q(t+)

? ? 0 : ? : 0;

? (01) 1 : ? : d;

? ? x : ? : x;

? (0x) 1 : ? : -;

? (1x) 1 : ? : -;

? ? 1 : ? : -;

endtable

endprimitive

module counter3\_udp (

input wire clk,

input wire rst,

output wire [2:0] q

);

wire rst\_n;

wire [2:0] q\_bar;

not not\_rst (rst\_n, rst);

assign qbar = ~q;

//gates

wire xor1\_out, xor2\_out, nand\_out;

xor xor1\_g(xor1\_out, q\_bar[2], nand\_out);

xor xor2\_g(xor2\_out, q[0], q[1]);

nand nand\_g(nand\_out, q[0], q[1]);

//DFF

udp\_dff\_async\_rstn ff0(q[0], qbar[0], qbar[0], clk, rst\_n);

udp\_dff\_async\_rstn ff1(q[1], qbar[1], xor2\_out, clk, rst\_n);

udp\_dff\_async\_rstn ff2(q[2], qbar[2], xor1\_out, clk, rst\_n);

endmodule

**Testbench Code:**

module tb\_counter3\_udp;

reg clk, rst;

wire [2:0] q;

counter3\_udp dut(.clk(clk), .rst(rst), .q(q));

initial begin

clk = 0;

forever #5 clk = ~clk;

end

initial begin

$display(" time rst | q2 q1 q0");

$monitor("%6t %b | %b %b %b", $time, rst, q[2], q[1], q[0]);

rst = 1'b1; #12;

rst = 1'b0;

repeat (20) @(posedge clk);

rst = 1'b1; #7;

rst = 1'b0;

repeat (10) @(posedge clk);

$finish;

end

endmodule