Embedded Systems

CS429 Lab6

Name: Dipean Dasgupta **ID:202151188**

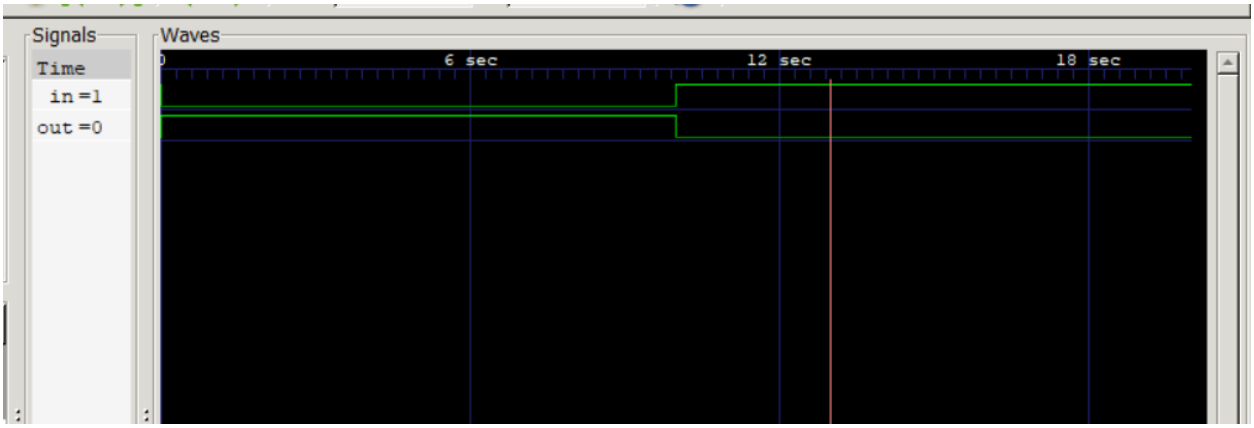
Task1: Design NOT, NOR, NAND, XOR and XNOR gates using switch level modeling.

Not Gate

```
module not_gate (
  input in,
  output out
);
  supply1 vdd;
  supply0 gnd;
  pmos p1 (out, vdd, in);
  nmos n1 (out, gnd, in);
endmodule

module not_gate_tb;
  reg in;
  wire out;
  not_gate dut (
    .in(in),
    .out(out)
  );
  initial begin
    $dumpfile("not_gate.vcd");
    $dumpvars(0, not_gate_tb);
    in = 0;
    #10;
    in = 1;
    #10;
    if (out != !in) begin
      $display("Error: Output of NOT gate is incorrect.");
    end
    $finish;
  end
endmodule
```

Output

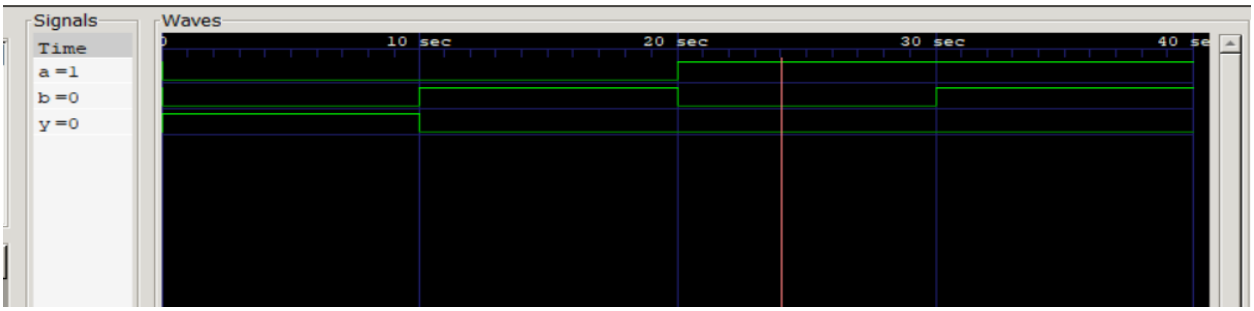


NOR Gate

```
module nor_gate (
  input a, b,
  output y
);
  wire w1;
  supply1 vdd;
  supply0 gnd;
  nmos n1 (y, gnd, a);
  nmos n2 (y, gnd, b);
  pmos p1 (w1, vdd, b);
  pmos p2 (y, w1, a); // Corrected "out" to "y"
endmodule

module nor_gate_tb;
  reg a, b;
  wire y;
  nor_gate dut (
    .a(a),
    .b(b),
    .y(y)
  );
  initial begin
    $dumpfile("nor_gate.vcd");
    $dumpvars(0, nor_gate_tb);
    a = 0;
    b = 0;
    #10;
    if (y != 1) $display("Test case 1 failed");
    a = 0;
    b = 1;
    #10;
    if (y != 0) $display("Test case 2 failed");
    a = 1;
    b = 0;
    #10;
    if (y != 0) $display("Test case 3 failed");
    a = 1;
    b = 1;
    #10;
    if (y != 0) $display("Test case 4 failed");
    $finish;
  end
endmodule
```

Output

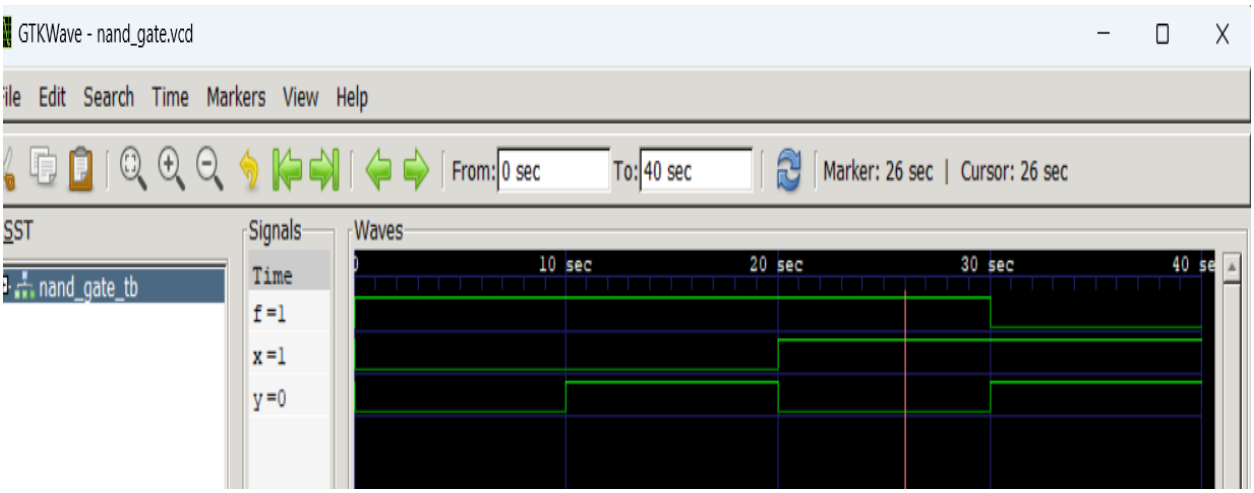


NAND Gate

```
module nand_gate (
    input x,y , output f
);
supply1 vdd;
supply0 gnd;
wire a;
pmos p1 (f, vdd, x);
pmos p2 (f, vdd, y);
nmos n1 (f, a, x);
nmos n2 (a, gnd, y);
endmodule

module nand_gate_tb;
reg x,y;
wire f;
nand_gate dut ( .x(x), .y(y), .f(f));
initial begin
    $dumpfile("nand_gate.vcd");
    $dumpvars(0, nand_gate_tb);
    x = 0;  y = 0;
    #10;
    if (f !== 1) $display("Test case 1 failed");
    x = 0;
    y = 1;
    #10;
    if (f !== 1) $display("Test case 2 failed");
    x = 1;
    y = 0;
    #10;
    if (f !== 1) $display("Test case 3 failed");
    x = 1;
    y = 1;
    #10;
    if (f !== 0) $display("Test case 4 failed");
    $finish;
end
endmodule
```

Output :

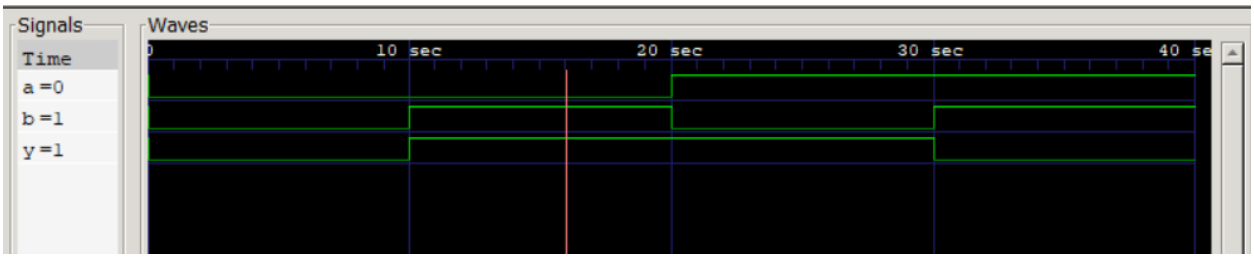


XOR Gate

```
module xor_gate (
  input a, b,
  output y
);
  wire w1;
  supply1 vdd;
  supply0 gnd;
  pmos p1(y,b,a);
  nmos n1(y,w1,a);
  nmos c1(y,a,w1);
  pmos c2(y,a,b);
  pmos p2(w1,vdd,b);
  nmos n2(w1,gnd,b);
endmodule

module xor_gate_tb;
  reg a, b;
  wire y;
  xor_gate uut (.a(a), .b(b), .y(y));
  initial begin
    $dumpfile("xor_gate.vcd");
    $dumpvars(0, xor_gate_tb);
    a = 0; b = 0;
    #10;
    if (y != 0) $display("Test case 1 failed");
    a = 0;
    b = 1;
    #10;
    if (y != 1) $display("Test case 2 failed");
    a = 1;
    b = 0;
    #10;
    if (y != 1) $display("Test case 3 failed");
    a = 1;
    b = 1;
    #10;
    if (y != 0) $display("Test case 4 failed");
    $finish;
  end
endmodule
```

Output :



XNOR Gate:

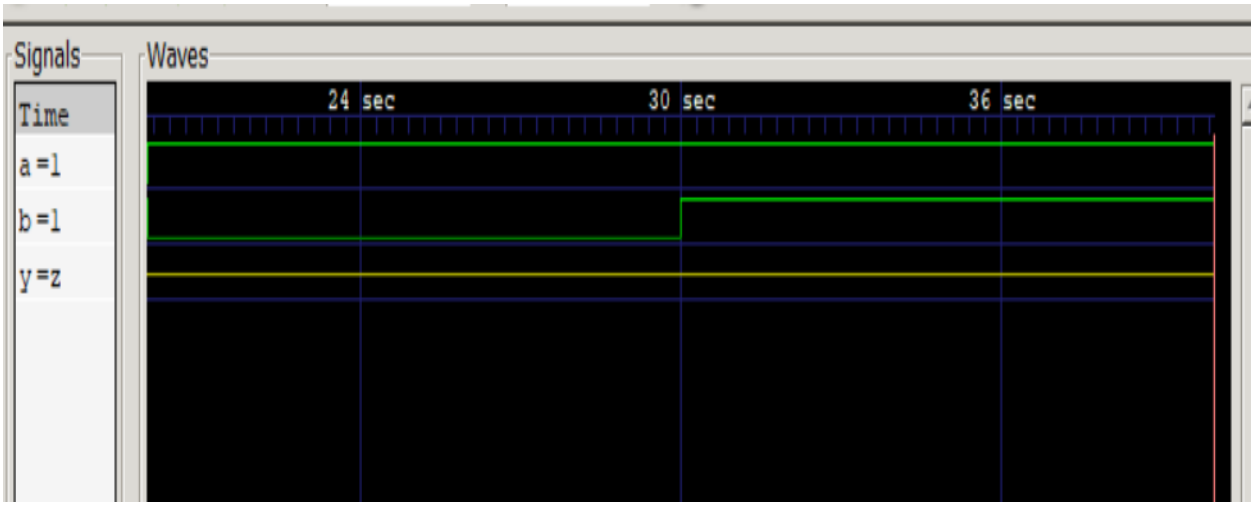
```
module xnor_gate (
  input a, b,
  output y
);
  wire w1, w2, w3;
  supply1 vdd;
  supply0 gnd;

  nmos n1 (w1, gnd, a);
```

```
  nmos n2 (w2, gnd, b);
  pmos p1 (w3, vdd, a);
  pmos p2 (w3, vdd, b);

  wire w4;
  pmos p3 (w4, vdd, w1);
  pmos p4 (y, vdd, w4);
endmodule
// test bench
module xnor_gate_tb;
  reg a, b;
  wire y;
  xnor_gate uut (.a(a), .b(b), .y(y));
  initial begin
    $dumpfile("xnor_gate.vcd");
    $dumpvars(0, xnor_gate_tb);
    a = 0;b = 0;
    #10;
    if (y != 1) $display("Test case 1 failed");
    a = 0;
    b = 1;
    #10;
    if (y != 0) $display("Test case 2 failed");
    a = 1;
    b = 0;
    #10;
    if (y != 0) $display("Test case 3 failed");
    a = 1;
    b = 1;
    #10;
    if (y != 1) $display("Test case 4 failed");
    $finish;
  end
endmodule
```

Output:



Task2: Design NOT, NOR, NAND, XOR and XNOR gates using switch level modeling with less number of transistor count than in conventional CMOS.

NOT Gate

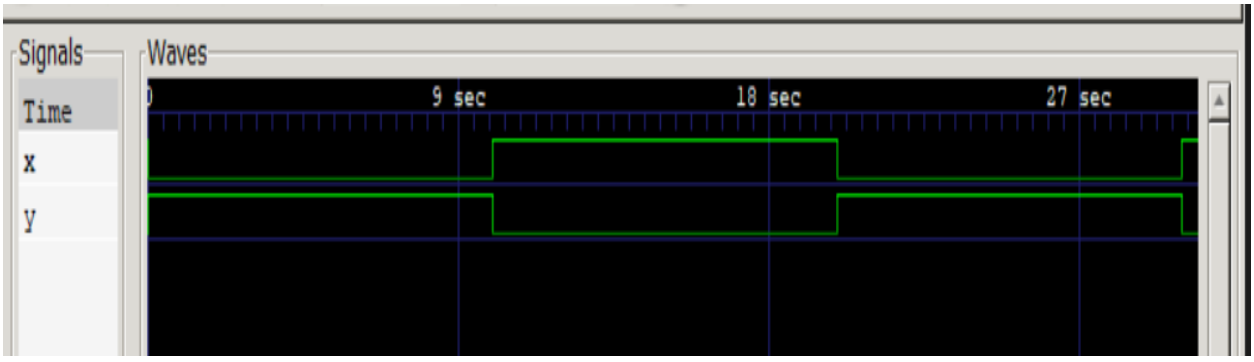
```
// Not gate
module not_gate(input x, output y);
  supply0 gnd;
  supply1 vdd;
  pmos p1 (y,vdd,x);
  nmos n1 (y,gnd,x);
endmodule

// testbech code
module not_gate_tb();
  reg x;
  wire y;
  not_gate uut (x,y);
  initial begin
    $dumpfile("not_gate_sm2.vcd");
    $dumpvars(0,not_gate_tb);
    x = 0;
    #10 x = 1;
    #10 x = 0;
    #10 x = 1;
    #1 $finish;
  end
  // display values
  initial begin
    $monitor("x = %b, y = %b", x, y);
  end
endmodule
```

Output

```
[Running] not_gate_pass.v
VCD info: dumpfile not_gate_sm2.vcd opened for output.
x = 0, y = 1
x = 1, y = 0
x = 0, y = 1
x = 1, y = 0
[Done] exit with code=0 in 0.352 seconds
```

Waveform



NOR Gate

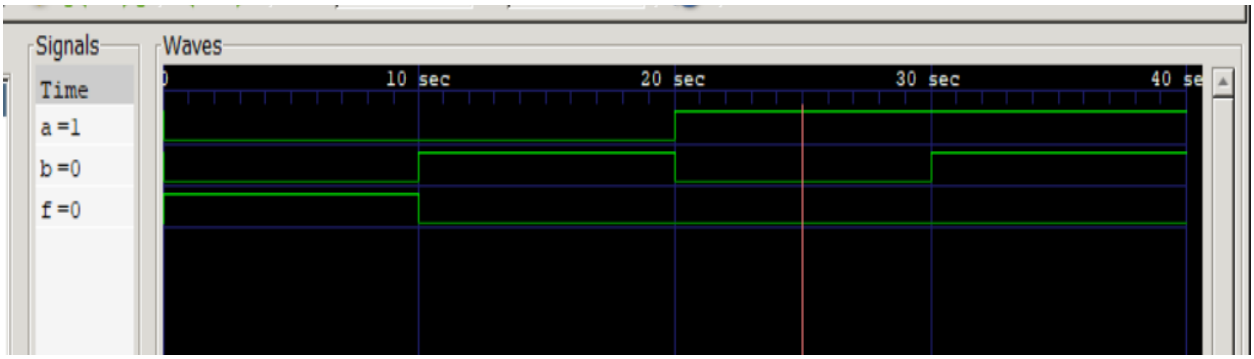
```
// Nor gate
module nor_gate(input a, input b, output f);
  wire anot,bnot;
  not g1(anot,a);
  not g2(bnot,b);
  nmos n1(f,1'b0,b);
  nmos n2(f,anot,bnot);
endmodule

//Test bench
module nor_gate_tb();
  reg a,b;
  wire f;
  nor_gate n3(a,b,f);
  initial begin
    $dumpfile("nor_gate_sm2.vcd");
    $dumpvars(0,nor_gate_tb);
    {a,b}=2'b00;
    #10 {a,b}=2'b01;
    #10 {a,b}=2'b10;
    #10 {a,b}=2'b11;
    #10 $finish;
  end
  // display values
  initial begin
    $monitor("a = %b, b = %b, f = %b", a, b, f);
  end
endmodule
```

Output

```
[Running] nor_gate_pass.v
VCD info: dumpfile nor_gate_sm2.vcd opened for output.
a = 0, b = 0, f = 1
a = 0, b = 1, f = 0
a = 1, b = 0, f = 0
a = 1, b = 1, f = 0
[Done] exit with code=0 in 0.33 seconds
```

Waveform



NAND Gate

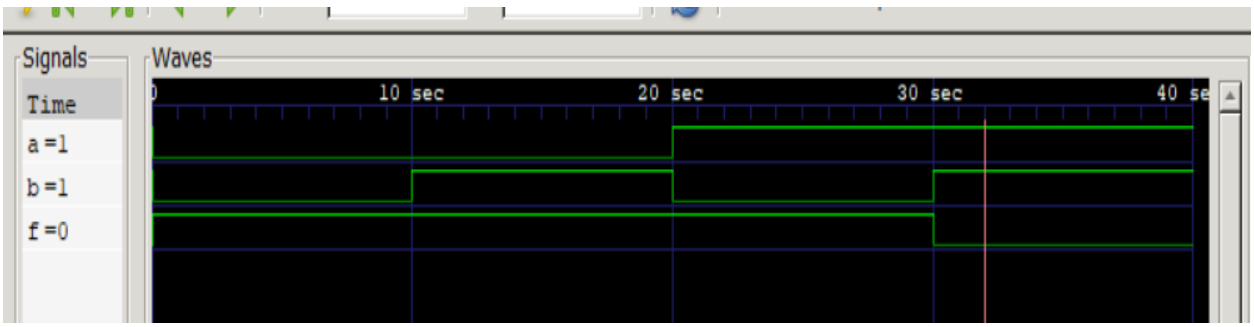
```
// Nand gate
module nand_gate (input a, input b, output f);
  wire anot,bnot;
  not g1(anot,a);
  not g2(bnot,b);
  nmos n1(f,anot,b);
  nmos n2(f,1'b1,bnot);
endmodule

//Test bench
module nand_gate_tb();
  reg a,b;
  wire f;
  nand_gate n3(a,b,f);
  initial begin
    $dumpfile("nand_gate_sm2.vcd");
    $dumpvars(0,nand_gate_tb);
    {a,b}=2'b00;
    #10 {a,b}=2'b01;
    #10 {a,b}=2'b10;
    #10 {a,b}=2'b11;
    #10 $finish;
  end
  // display values
  initial begin
    $monitor("a = %b, b = %b, f = %b", a, b, f);
  end
endmodule
```

Output

```
[Running] nand_gate_pass.v
VCD info: dumpfile nand_gate_sm2.vcd opened for output.
a = 0, b = 0, f = 1
a = 0, b = 1, f = 1
a = 1, b = 0, f = 1
a = 1, b = 1, f = 0
[Done] exit with code=0 in 0.361 seconds
```

GTK Wave



XOR Gate

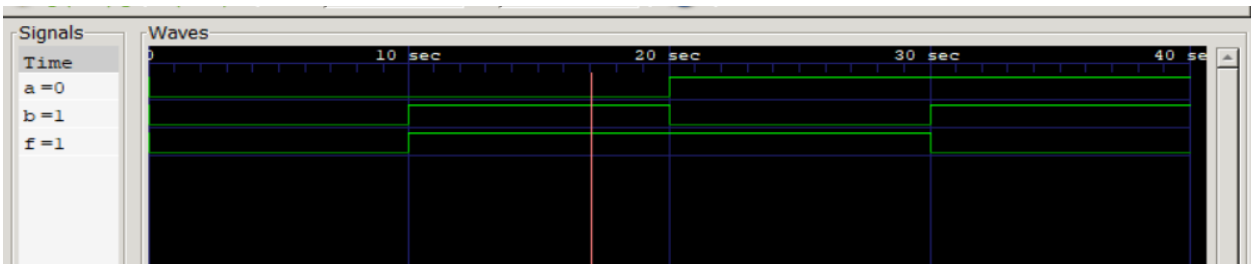
```
// Xor gate
module xor_gate(input a, input b, output f);
  wire anot,bnot;
  not g1(anot,a);
  not g2(bnot,b);
  nmos n1(f,anot,b);
  nmos n2(f,a,bnot);
endmodule

// testbench code
module xor_gate_tb;
  reg a;
  reg b;
  wire f;
  xor_gate uut (.a(a),.b(b),.f(f));
  initial begin
    $dumpfile("xor_gate_sm2.vcd");
    $dumpvars(0,xor_gate_tb);
    {a,b}=2'b00;
    #10 {a,b}=2'b01;
    #10 {a,b}=2'b10;
    #10 {a,b}=2'b11;
    #10 $finish;
  end
  // display values
  initial begin
    $monitor("a = %b, b = %b, f = %b", a, b, f);
  end
endmodule
```

Output

```
[Running] xor_gate_pass.v
VCD info: dumpfile xor_gate_sm2.vcd opened for output.
a = 0, b = 0, f = 0
a = 0, b = 1, f = 1
a = 1, b = 0, f = 1
a = 1, b = 1, f = 0
[Done] exit with code=0 in 0.354 seconds
```

Waveform



XNOR

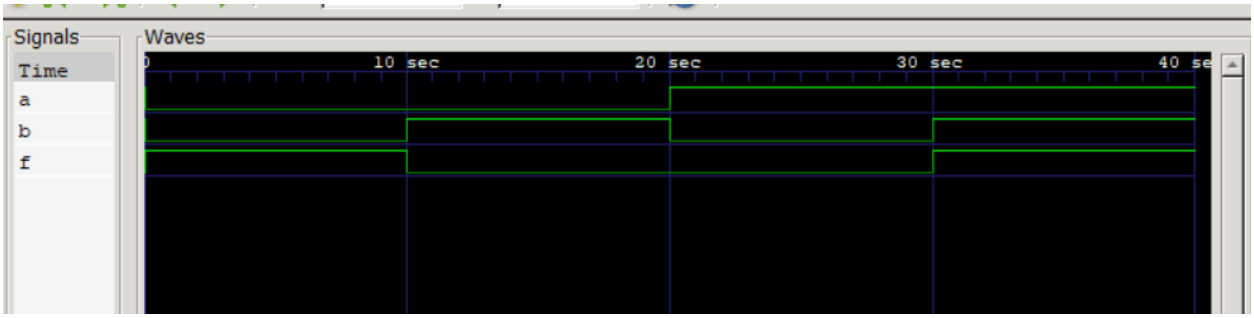
```
// Xnor gate
module xnor_gate(input a, input b, output f);
  wire anot,bnot;
  not g1(anot,a);
  not g2(bnot,b);
  nmos n1(f,a,b);
  nmos n2(f,anot,bnot);
endmodule

// testbench code
module xnor_gate_tb;
  reg a;
  reg b;
  wire f;
  xnor_gate uut (.a(a),.b(b),.f(f));
  initial begin
    $dumpfile("xnor_gate_sm2.vcd");
    $dumpvars(0,xnor_gate_tb);
    {a,b}=2'b00;
    #10 {a,b}=2'b01;
    #10 {a,b}=2'b10;
    #10 {a,b}=2'b11;
    #10 $finish;
  end
  // display values
  initial begin
    $monitor("a = %b, b = %b, f = %b", a, b, f);
  end
endmodule
```

Output

```
VCD info: dumpfile xnor_gate_sm2.vcd opened for output.
a = 0, b = 0, f = 1
a = 0, b = 1, f = 0
a = 1, b = 0, f = 0
a = 1, b = 1, f = 1
[Done] exit with code=0 in 0.359 seconds
```

GTK Wave



Task3: Implement 2:1 and 4:1 multiplexers with switch level modeling.

2:1 Multiplexor model

```
module multiplexer_2to1_switch_level (
    input wire a, // Input data bit 0
    input wire b, // Input data bit 1
    input wire sel, // Selection input
    output wire y // Output data
);

    // NMOS and PMOS transistors for the 2:1 multiplexer
    nmos n1 (n1_out, sel, a);
    nmos n2 (n2_out, sel, b);
    pmos p1 (p1_out, sel, a);
    pmos p2 (p2_out, sel, b);

    // Internal wires to store NMOS and PMOS outputs
    wire n1_out, n2_out, p1_out, p2_out;

    // Output assignment using transmission gates
    assign y = (n1_out | p1_out) & (~n2_out & ~p2_out);

endmodule

// Testbench for 2:1 multiplexor

module tb_multiplexer_2to1_switch_level;
    reg a, b, sel;
    wire y;

    multiplexer_2to1_switch_level uut (
        .a(a),
        .b(b),
        .sel(sel),
        .y(y)
    );

    initial begin
        $monitor("Time=%0t, a=%b, b=%b, sel=%b, y=%b", $time, a, b, sel, y);

        // Test case 1: Select input 'a'
        a = 1'b1;
        b = 1'b0;
```

```
        sel = 1'b0;
        #5;
        // Test case 2: Select input 'b'
        a = 1'b0;
        b = 1'b1;
        sel = 1'b1;
        #5;
        // Test case 3: Select input 'a'
        a = 1'b1;
        b = 1'b0;
        sel = 1'b0;
        #5;
        // Test case 4: Select input 'b'
        a = 1'b0;
        b = 1'b1;
        sel = 1'b1;
        #5;
        $finish;
    end
endmodule
```

4:1 multiplexors

```
module multiplexer_4to1_switch_level (
    input wire [3:0] data, // 4-bit input data
    input wire [1:0] sel,  // 2-bit selection input
    output wire y // Output data
);

    // NMOS and PMOS transistors for the 4:1 multiplexer
    nmos n1 (n1_out, sel[1], data[0]);
    nmos n2 (n2_out, sel[1], data[1]);
    nmos n3 (n3_out, sel[1], data[2]);
    nmos n4 (n4_out, sel[1], data[3]);

    nmos n5 (n5_out, sel[0], n1_out);
    nmos n6 (n6_out, sel[0], n2_out);
    nmos n7 (n7_out, sel[0], n3_out);
    nmos n8 (n8_out, sel[0], n4_out);

    pmos p1 (p1_out, sel[1], data[0]);
    pmos p2 (p2_out, sel[1], data[1]);
    pmos p3 (p3_out, sel[1], data[2]);
    pmos p4 (p4_out, sel[1], data[3]);
```

```
    pmos p5 (p5_out, sel[0], p1_out);
    pmos p6 (p6_out, sel[0], p2_out);
    pmos p7 (p7_out, sel[0], p3_out);
    pmos p8 (p8_out, sel[0], p4_out);
    // Internal wires to store NMOS and PMOS outputs
    wire n1_out, n2_out, n3_out, n4_out, n5_out, n6_out, n7_out, n8_out;
    wire p1_out, p2_out, p3_out, p4_out, p5_out, p6_out, p7_out, p8_out;
    // Output assignment using transmission gates
    assign y = (n5_out | p5_out | n6_out | p6_out | n7_out | p7_out | n8_out |
p8_out);
endmodule
// Testbench for 4:1 Multiplexer
module tb_multiplexer_4to1_switch_level;
    reg [3:0] data;
    reg [1:0] sel;
    wire y;
    multiplexer_4to1_switch_level uut (
        .data(data),
        .sel(sel),
        .y(y)
    );
    initial begin
        $monitor("Time=%0t, data=%b, sel=%b, y=%b", $time, data, sel, y);
        // Test case 1: Select input 0 (data[0])
        data = 4'b0001;
        sel = 2'b00;
        #5;
        // Test case 2: Select input 1 (data[1])
        data = 4'b0010;
        sel = 2'b01;
        #5;
        // Test case 3: Select input 2 (data[2])
        data = 4'b0100;
        sel = 2'b10;
        #5;
        // Test case 4: Select input 3 (data[3])
        data = 4'b1000;
        sel = 2'b11;
        #5;
        $finish;
    end
endmodule
```

Task4: Design SRAM

(ii)Using strength in Verilog

Code:

```
module sram(
    input [1:0] data_in,
    input [1:0] address,
    input [1:0] write,
    output [1:0] data_out
);
    tri w1,w2,w3;

    bufif g1(w1,data_in,write);
    tranif g2(w2,w1,address);
    not(pull0,pull1) g3(w3,w2), g4(w2,w3);
    buf g5(data_out,w1);
endmodule

module tb_sram();

    reg [1:0] data_in;
    reg [1:0] address;
    reg [1:0] write;
    wire [1:0] data_out;

    sram dut (
        .data_in(data_in),
        .address(address),
        .write(write),
        .data_out(data_out)
    );

    initial begin
        // Test Case 1:
        data_in = 2'b11;
        address = 2'b00;
        write = 2'b01;
        #10;

        // Test Case 2
        data_in = 2'b01;
        address = 2'b01;
        write = 2'b01;
        #10;
```

```
        // Test Case 3: Read data from address 0
        data_in = 2'bzz; // Don't care for read operation
        address = 2'b00;
        write = 2'b10;
        #10;

        // Test Case 4: Read data from address 1
        data_in = 2'bzz;
        address = 2'b01;
        write = 2'b10;
        #10;
        $finish;
    end
endmodule
```

Lab Exercise:

Implement capacitor using nmos switch primitive in Verilog HDL. Write testbench to check its functionality.

Code:

```
module capacitor_nmos (
    input data,gate;
    output out
);
      trireg out;
      nmos gl(out, data, control);
endmodule
module tb_capacitor_nmos();
    reg data, gate;
    wire out;
    capacitor_nmos dut (
        .data(data),
        .gate(gate),
        .out(out)
    );
    initial begin
        // Test Case 1:
        data = 1'b1;
        gate = 1'b1;
```

```
        #10;
        // Test Case 2:
        data = 1'b1;
        gate = 1'b0;
        #10;
        // Test Case 3:
        data = 1'b0;
        gate = 1'b1;
        #10;
        // Test Case 4
        data = 1'b0;
        gate = 1'b0;
        #10;
        $finish;
    end
endmodule
```

Lab Ex2: Design 4-bit Full Adder using switch level modeling.

Code:

```
module nmos_transistor (output wire Y, input A, input B, input C);
    assign Y = (A & B & C);
endmodule

module pmos_transistor (output wire Y, input A, input B, input C);
    assign Y = (A | B | C);
endmodule

module full_adder_4bit (
    input wire [3:0] A, B,
    input wire Cin,
    output wire [3:0] Sum,
    output wire Cout
);
    wire [3:0] X, Y, Z;

    // NMOS and PMOS transistors
    nmos_transistor nmos1 (X[0], A[0], B[0], Cin);
    nmos_transistor nmos2 (X[1], A[1], B[1], Cin);
    nmos_transistor nmos3 (X[2], A[2], B[2], Cin);
    nmos_transistor nmos4 (X[3], A[3], B[3], Cin);

    pmos_transistor pmos1 (Y[0], A[0], B[0], Cin);
    pmos_transistor pmos2 (Y[1], A[1], B[1], Cin);
```

```
    pmos_transistor pmos3 (Y[2], A[2], B[2], Cin);
    pmos_transistor pmos4 (Y[3], A[3], B[3], Cin);

    nmos_transistor nmos5 (Z[0], X[0], Y[0], Cout);
    nmos_transistor nmos6 (Z[1], X[1], Y[1], Cout);
    nmos_transistor nmos7 (Z[2], X[2], Y[2], Cout);
    nmos_transistor nmos8 (Z[3], X[3], Y[3], Cout);

    pmos_transistor pmos5 (Cout, X[0], Y[0], Z[0]);

    assign Sum = Z;

endmodule
// Testbench

module testbench_full_adder;
    reg [3:0] A, B;
    reg Cin;
    wire [3:0] Sum;
    wire Cout;

    // Instantiate the 4-bit Full Adder
    full_adder_4bit dut (
        .A(A),
        .B(B),
        .Cin(Cin),
        .Sum(Sum),
        .Cout(Cout)
    );

    // Clock generation for simulation
    reg clk;
    always #5 clk = ~clk;

    // Testbench stimulus
    initial begin
        $dumpfile("full_adder_4bit.vcd");
        $dumpvars(0, testbench_full_adder);

        A = 4'b0000;
        B = 4'b0000;
        Cin = 0;
        #10;

        A = 4'b1010;
```

```
        B = 4'b0110;
        Cin = 1;
        #10;

        A = 4'b1101;
        B = 4'b0101;
        Cin = 0;
        #10;

        $finish;
    end

    // Clock driver
    always begin
        #2 clk = 0;
        #2 clk = 1;
    end
endmodule
```

OUTPUT: