Lab Assignment 4

Name- Aye Myat Thiri Khin

ID-20293

**Q :** Design the following 2-bits RTL level combinational logic modules in continuous assignment and always blocks for each gate and write the testbench to verify your design covering all the testcases by showing the waveforms,

* 1. 1-bit NOT gate
  2. 2-bits AND gate
  3. 2-bits OR gate
  4. 2-bits NAND gate
  5. 2-bits NOR gate
  6. 2-bits XOR gate
  7. 2-bits XNOR gate

Ans:

(1)

1. bit NOT gate

module not\_gate (

input a,

output y

);

assign y = ~a;

endmodule

module test\_not\_gate;

reg a;

wire y;

not\_gate uut (

.a(a),

.y(y)

);

initial begin

$dumpfile("not\_gate.vcd"); // For VCD waveform generation

$dumpvars(0, test\_not\_gate);

// Test all input cases

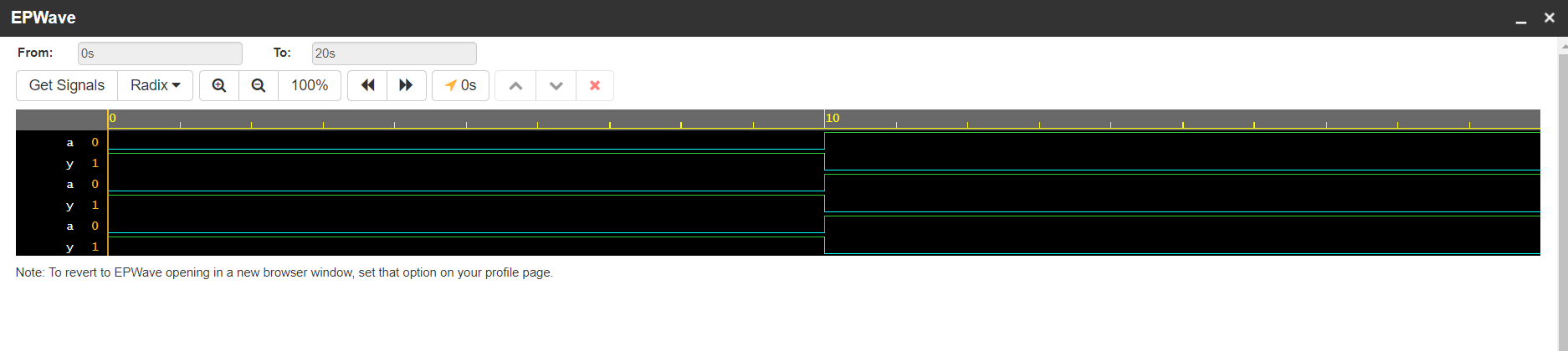
a = 1'b0; #10;

a = 1'b1; #10;

$finish;

end

endmodule



(2)

1. bits AND gate

module and\_gate\_2bit (

input [1:0] a, b,

output [1:0] y

);

assign y = a & b;

endmodule

module test\_and\_gate\_2bit;

reg [1:0] a, b;

wire [1:0] y;

and\_gate\_2bit uut (

.a(a),

.b(b),

.y(y)

);

initial begin

$dumpfile("and\_gate\_2bit.vcd");

$dumpvars(0, test\_and\_gate\_2bit);

// Apply all input combinations

a = 2'b00; b = 2'b00; #10;

a = 2'b01; b = 2'b01; #10;

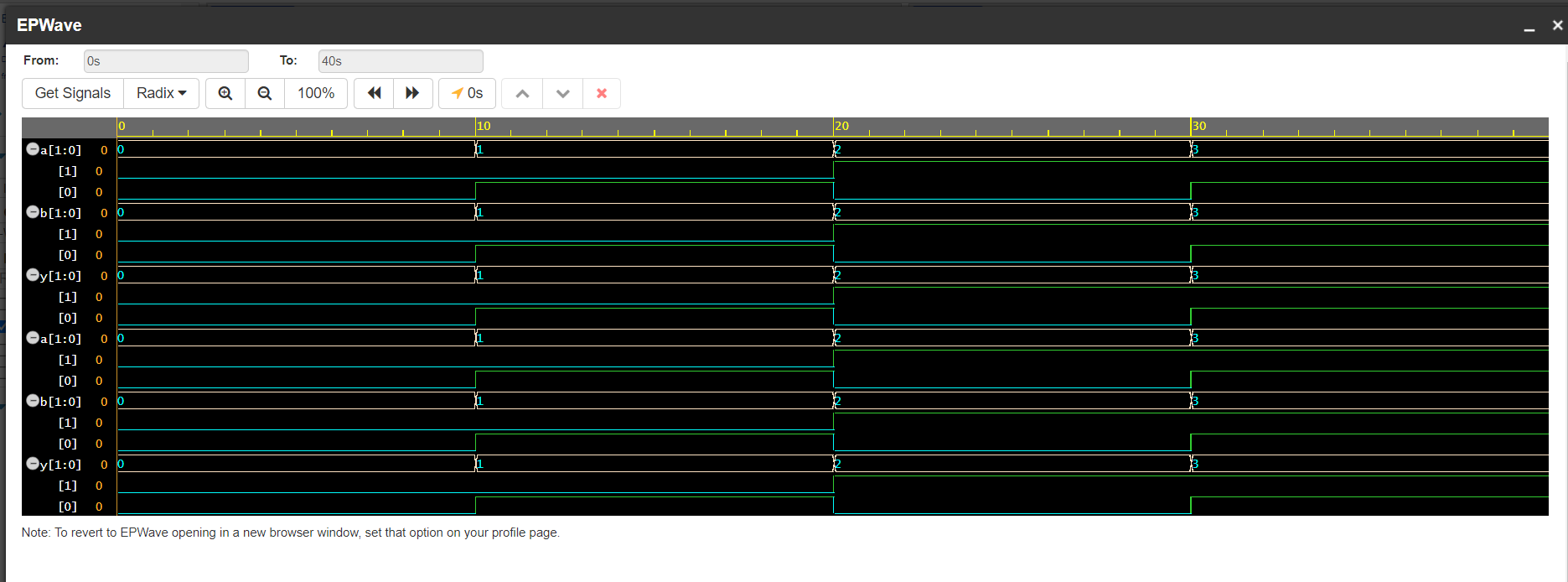
a = 2'b10; b = 2'b10; #10;

a = 2'b11; b = 2'b11; #10;

$finish;

end

endmodule



(3)

2-bits OR gate

module or\_gate\_2bit (

input [1:0] a, b,

output [1:0] y

);

assign y = a | b;

endmodule

module test\_or\_gate\_2bit;

reg [1:0] a, b;

wire [1:0] y;

or\_gate\_2bit uut (

.a(a),

.b(b),

.y(y)

);

initial begin

$dumpfile("or\_gate\_2bit.vcd");

$dumpvars(0, test\_or\_gate\_2bit);

// Test all cases

a = 2'b00; b = 2'b00; #10;

a = 2'b01; b = 2'b01; #10;

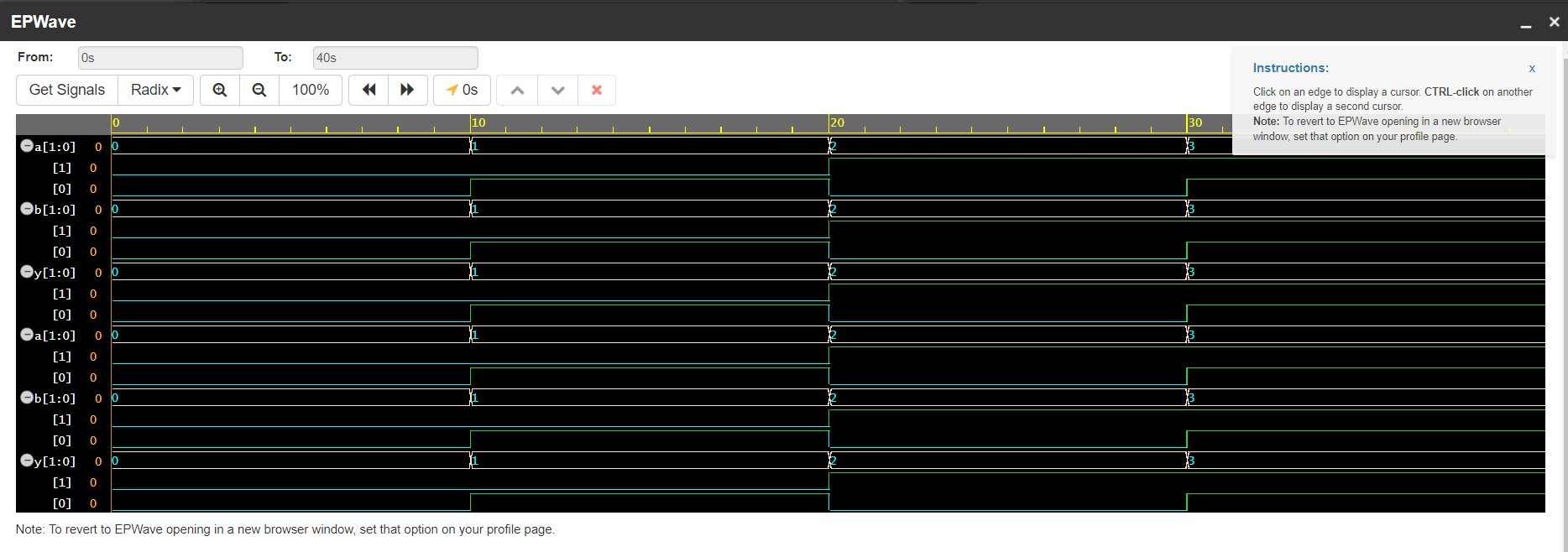
a = 2'b10; b = 2'b10; #10;

a = 2'b11; b = 2'b11; #10;

$finish;

end

endmodule



(4)

1. bits NAND gate

module nand\_gate\_2bit (

input [1:0] a, b,

output [1:0] y

);

assign y = ~(a & b);

endmodule

module test\_nand\_gate\_2bit;

reg [1:0] a, b;

wire [1:0] y;

nand\_gate\_2bit uut (

.a(a),

.b(b),

.y(y)

);

initial begin

$dumpfile("nand\_gate\_2bit.vcd");

$dumpvars(0, test\_nand\_gate\_2bit);

// Test cases

a = 2'b00; b = 2'b00; #10;

a = 2'b01; b = 2'b01; #10;

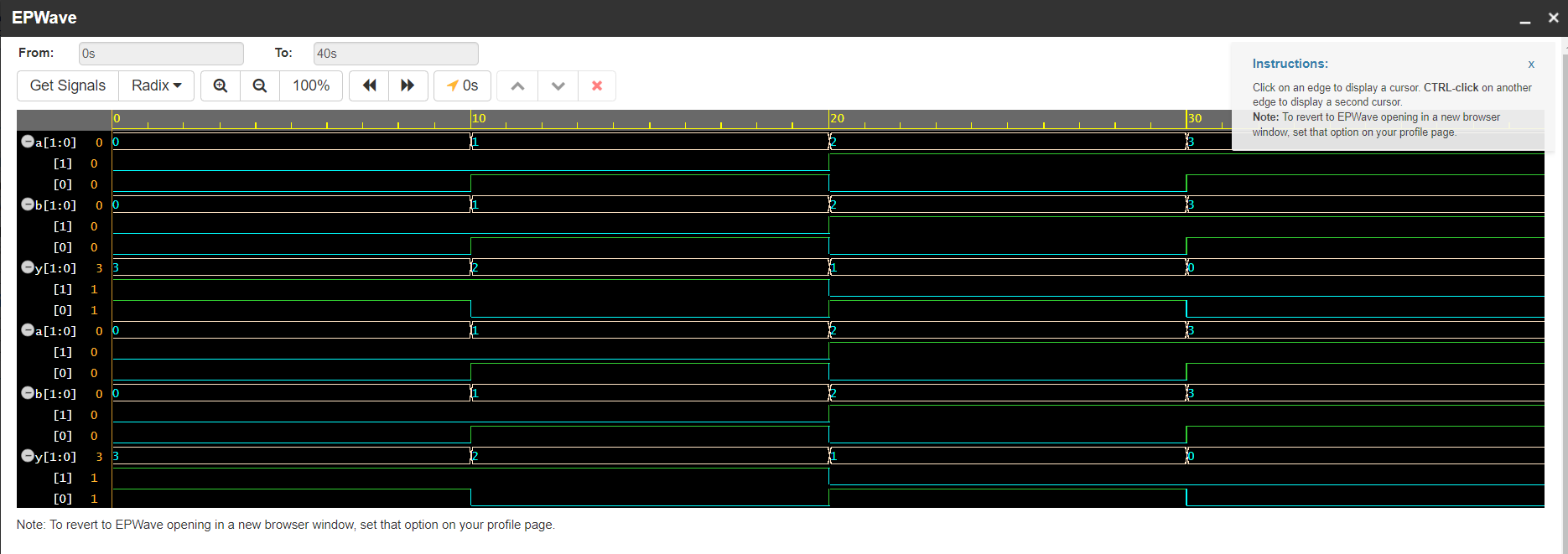
a = 2'b10; b = 2'b10; #10;

a = 2'b11; b = 2'b11; #10;

$finish;

end

endmodule



(5)

1. bits NOR gate

module nor\_gate\_2bit (

input [1:0] a, b,

output [1:0] y

);

assign y = ~(a | b);

endmodule

module test\_nor\_gate\_2bit;

reg [1:0] a, b;

wire [1:0] y;

// Instantiate the NOR gate module

nor\_gate\_2bit uut (

.a(a),

.b(b),

.y(y)

);

initial begin

$dumpfile("nor\_gate\_2bit.vcd"); // For VCD waveform generation

$dumpvars(0, test\_nor\_gate\_2bit);

// Apply test cases

a = 2'b00; b = 2'b00; #10; // Expected output: y = 2'b11

a = 2'b01; b = 2'b00; #10; // Expected output: y = 2'b10

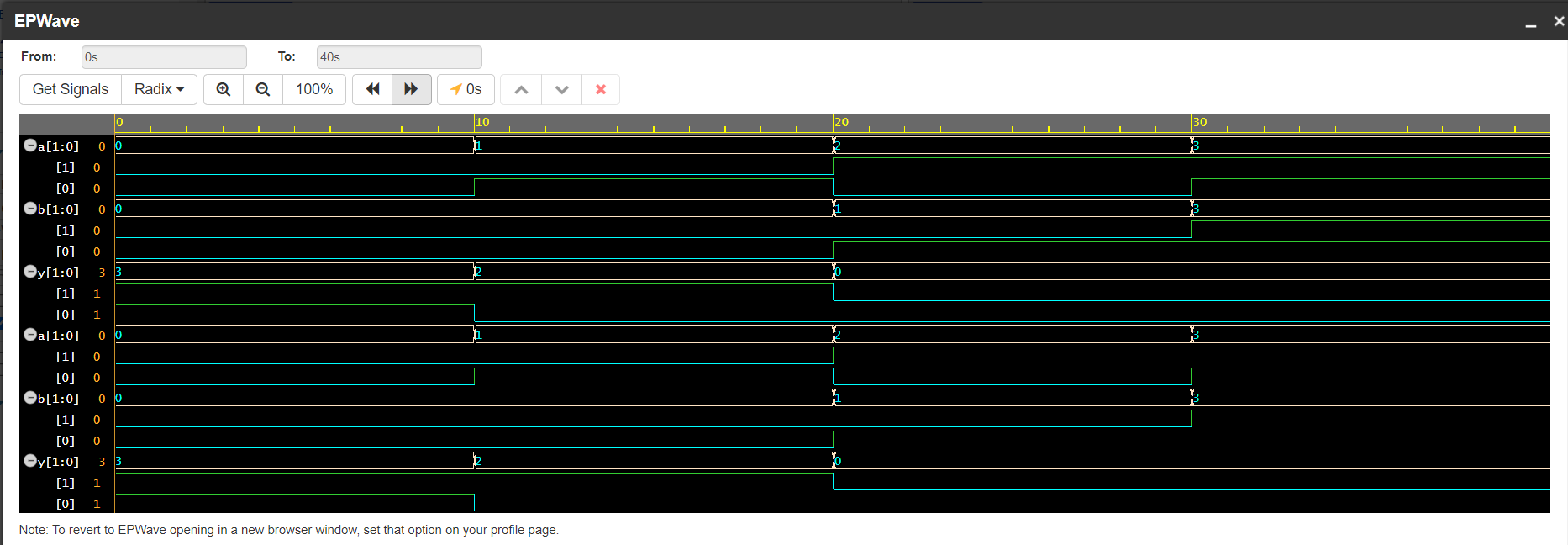
a = 2'b10; b = 2'b01; #10; // Expected output: y = 2'b00

a = 2'b11; b = 2'b11; #10; // Expected output: y = 2'b00

$finish;

end

endmodule



(6)

2-bits XOR gate

module xor\_gate\_2bit (

input [1:0] a, b,

output [1:0] y

);

assign y = a ^ b;

endmodule

module test\_xor\_gate\_2bit;

reg [1:0] a, b;

wire [1:0] y;

// Instantiate the XOR gate module

xor\_gate\_2bit uut (

.a(a),

.b(b),

.y(y)

);

initial begin

$dumpfile("xor\_gate\_2bit.vcd"); // For VCD waveform generation

$dumpvars(0, test\_xor\_gate\_2bit);

// Apply test cases

a = 2'b00; b = 2'b00; #10; // Expected output: y = 2'b00

a = 2'b01; b = 2'b01; #10; // Expected output: y = 2'b00

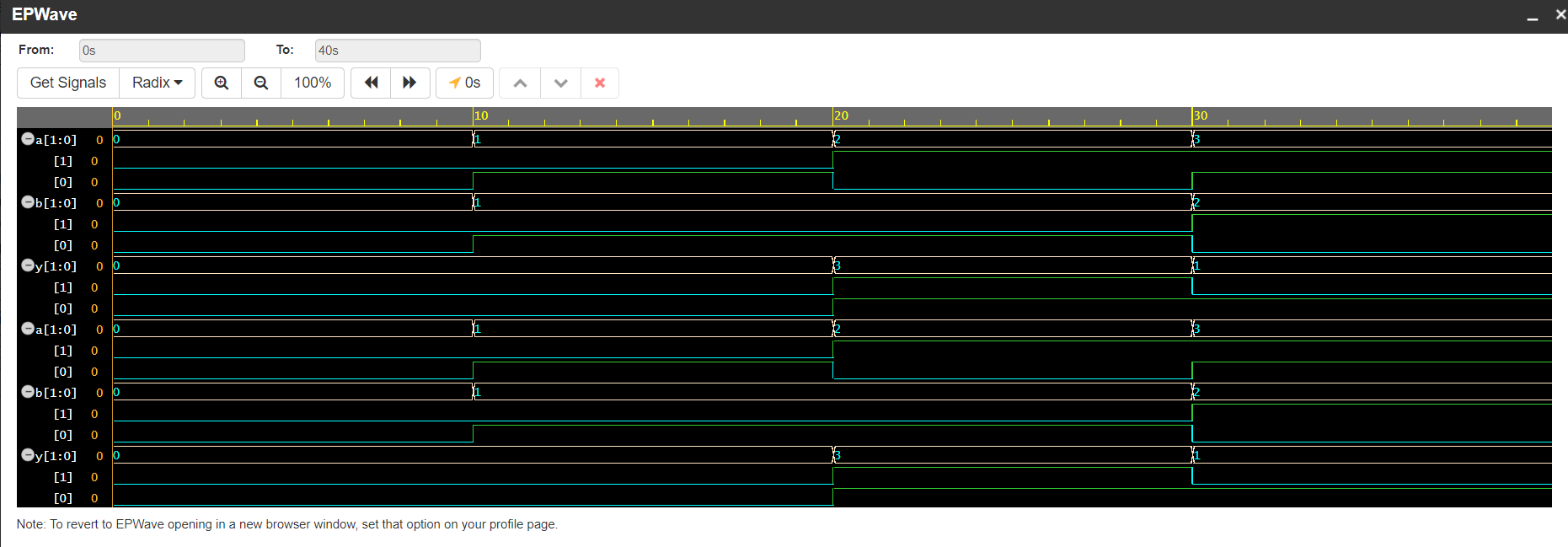
a = 2'b10; b = 2'b01; #10; // Expected output: y = 2'b11

a = 2'b11; b = 2'b10; #10; // Expected output: y = 2'b01

$finish;

end

endmodule



(7)

2-bits XNOR gate

module xnor\_gate\_2bit (

input [1:0] a, b,

output [1:0] y

);

assign y = ~(a ^ b);

endmodule

module test\_xnor\_gate\_2bit;

reg [1:0] a, b;

wire [1:0] y;

// Instantiate the XNOR gate module

xnor\_gate\_2bit uut (

.a(a),

.b(b),

.y(y)

);

initial begin

$dumpfile("xnor\_gate\_2bit.vcd"); // For VCD waveform generation

$dumpvars(0, test\_xnor\_gate\_2bit);

// Apply test cases

a = 2'b00; b = 2'b00; #10; // Expected output: y = 2'b11

a = 2'b01; b = 2'b01; #10; // Expected output: y = 2'b11

a = 2'b10; b = 2'b01; #10; // Expected output: y = 2'b00

a = 2'b11; b = 2'b10; #10; // Expected output: y = 2'b10

$finish;

end

endmodule

