**Full substractor:**

module full\_sub(a,b,bin,d,b0);

input a,b,bin;

output d,b0;

assign d = a ^ b ^ bin;

assign b0 = (~a & b) | ((~(a ^ b)) & bin);

Endmodule

**Test-bench:**

`timescale 1ns / 1ps

module full\_sub\_tb;

// Testbench signals

reg a, b, bin;

wire d, b0;

// Instantiate the full subtractor module

full\_sub uut (

.a(a),

.b(b),

.bin(bin),

.d(d),

.b0(b0)

);

initial begin

$display("A B Bin | D B0");

$display("--------|------");

// Apply all input combinations

a = 0; b = 0; bin = 0; #10 $display("%b %b %b | %b %b", a, b, bin, d, b0);

a = 0; b = 0; bin = 1; #10 $display("%b %b %b | %b %b", a, b, bin, d, b0);

a = 0; b = 1; bin = 0; #10 $display("%b %b %b | %b %b", a, b, bin, d, b0);

a = 0; b = 1; bin = 1; #10 $display("%b %b %b | %b %b", a, b, bin, d, b0);

a = 1; b = 0; bin = 0; #10 $display("%b %b %b | %b %b", a, b, bin, d, b0);

a = 1; b = 0; bin = 1; #10 $display("%b %b %b | %b %b", a, b, bin, d, b0);

a = 1; b = 1; bin = 0; #10 $display("%b %b %b | %b %b", a, b, bin, d, b0);

a = 1; b = 1; bin = 1; #10 $display("%b %b %b | %b %b", a, b, bin, d, b0);

$finish;

end

endmodule

**Half\_substractor:**

module half\_subs(a,b,d,b0);

input a,b;

output d,b0;

xor x1(d,a,b);

wire nota;

not x2(nota,a);

and x3(b0,nota,b);

endmodule

**Test-bench:**

`timescale 1ns / 1ps

module half\_subs\_tb;

// Inputs

reg a;

reg b;

// Outputs

wire d;

wire b0;

// Instantiate the Unit Under Test (UUT)

half\_subs uut (

.a(a),

.b(b),

.d(d),

.b0(b0)

);

initial begin

// Monitor the changes

$monitor("Time=%0t | a=%b b=%b => d=%b b0=%b", $time, a, b, d, b0);

// Test all combinations

a = 0; b = 0; #10;

a = 0; b = 1; #10;

a = 1; b = 0; #10;

a = 1; b = 1; #10;

$finish;

end

endmodule

**Multiplexer:**

module mux\_2to1 (

input a,

input b,

input sel,

output y

);

assign y = sel ? b : a;

endmodule

**Test-bench:**

`timescale 1ns / 1ps

module mux\_2to1\_tb;

// Inputs

reg a;

reg b;

reg sel;

// Output

wire y;

// Instantiate the Unit Under Test (UUT)

mux\_2to1 uut (

.a(a),

.b(b),

.sel(sel),

.y(y)

);

initial begin

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

$display("-------------");

// Test all combinations

a = 0; b = 0; sel = 0; #10;

$display("%b %b %b | %b", a, b, sel, y);

a = 0; b = 1; sel = 0; #10;

$display("%b %b %b | %b", a, b, sel, y);

a = 1; b = 0; sel = 0; #10;

$display("%b %b %b | %b", a, b, sel, y);

a = 1; b = 1; sel = 0; #10;

$display("%b %b %b | %b", a, b, sel, y);

a = 0; b = 0; sel = 1; #10;

$display("%b %b %b | %b", a, b, sel, y);

a = 0; b = 1; sel = 1; #10;

$display("%b %b %b | %b", a, b, sel, y);

a = 1; b = 0; sel = 1; #10;

$display("%b %b %b | %b", a, b, sel, y);

a = 1; b = 1; sel = 1; #10;

$display("%b %b %b | %b", a, b, sel, y);

$finish;

end

endmodule

**Mux 4:1 :**

module mux4to1 (

input [3:0] a, // 4-bit input data lines: a[3], a[2], a[1], a[0]

input [1:0] sel, // 2-bit select lines

output y // Output

);

assign y = (sel == 2'b00) ? a[0] :

(sel == 2'b01) ? a[1] :

(sel == 2'b10) ? a[2] :

a[3];

endmodule

**Test-bench:**

`timescale 1ns / 1ps

module tb\_mux4to1;

reg [3:0] a; // Inputs to the MUX

reg [1:0] sel; // Select lines

wire y; // Output from the MUX

// Instantiate the MUX module

mux4to1 uut (

.a(a),

.sel(sel),

.y(y)

);

initial begin

// Display header

$display("Time\t a\t sel\t y");

$monitor("%0t\t %b\t %b\t %b", $time, a, sel, y);

// Test case 1: sel = 00

a = 4'b1010; sel = 2'b00; #10;

// Test case 2: sel = 01

sel = 2'b01; #10;

// Test case 3: sel = 10

sel = 2'b10; #10;

// Test case 4: sel = 11

sel = 2'b11; #10;

// Change input and repeat

a = 4'b1100; sel = 2'b00; #10;

sel = 2'b01; #10;

sel = 2'b10; #10;

sel = 2'b11; #10;

// End simulation

$finish;

end

endmodule