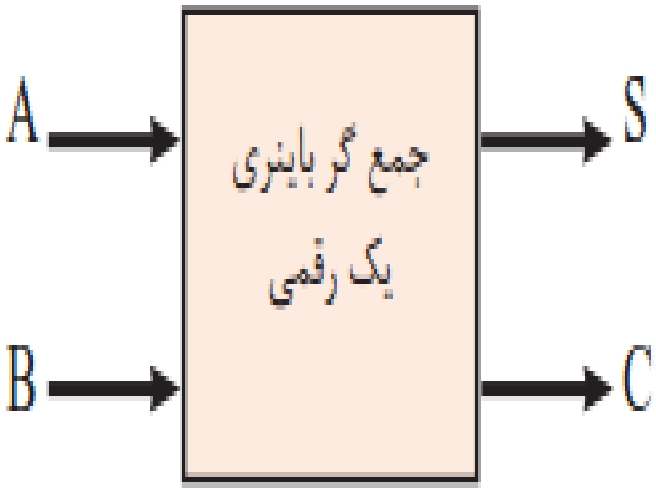
آزمایشگاه سیستم های دیجیتال2

آزمایش 2

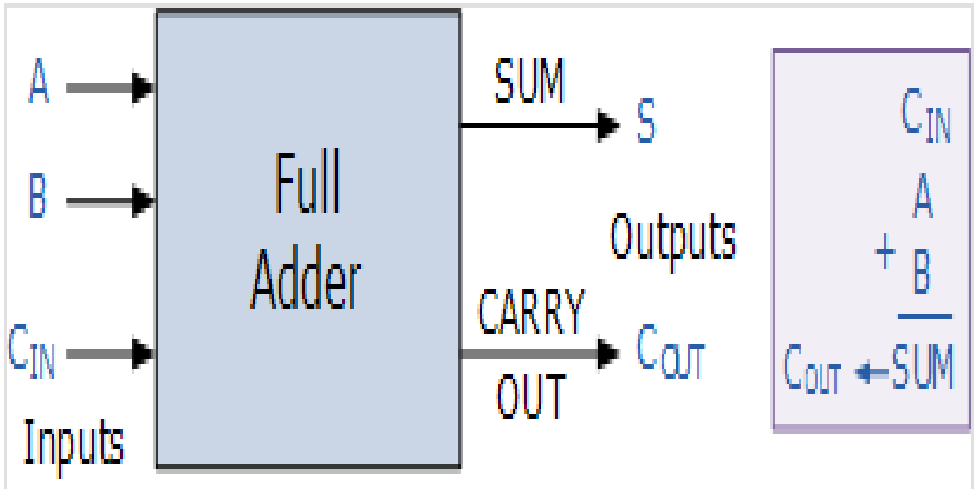
جمع کننده ها

نیمه جمع کننده(Half adder)

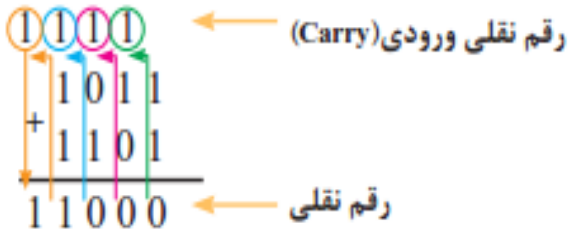


A	B	C	S	
0	0	0	0	0 + 0 = 0
0	1	0	1	0 + 1 = 1
1	0	0	1	1 + 0 = 1
1	1	1	0	1 + 1 =10

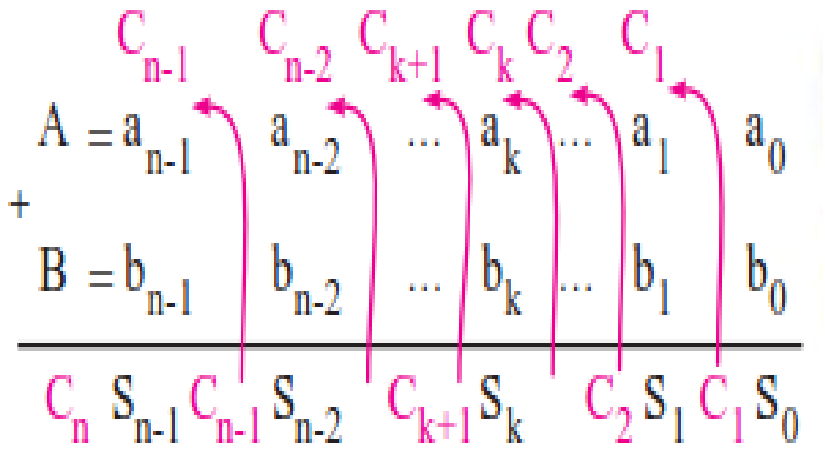
تمام جمع کننده(Full adder)

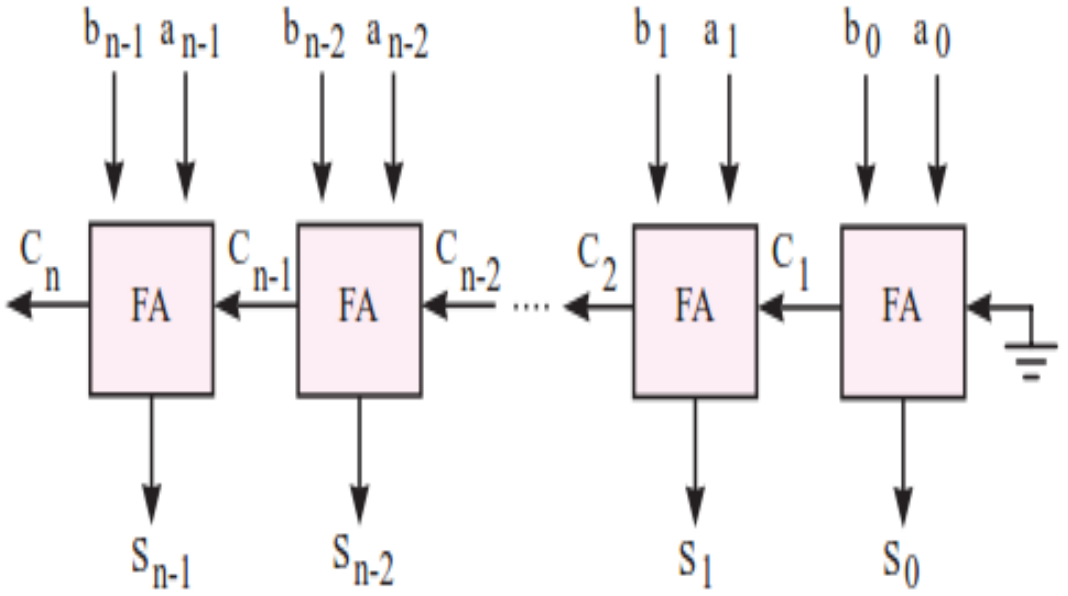


C-in	B	A	Sum	C-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



ادامه

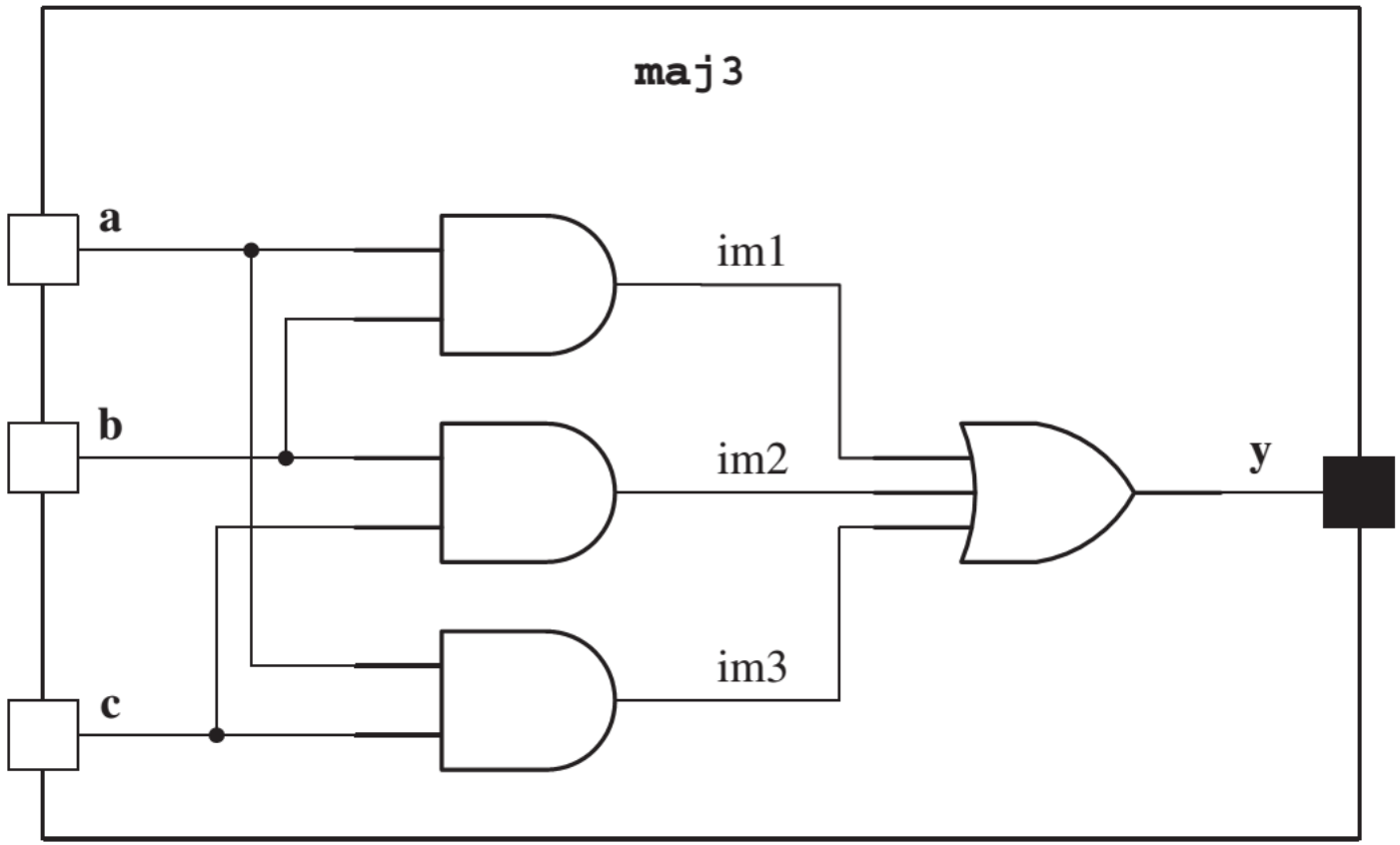

C_{n-1} C_{n-2} C_{k+1} C_k C_2 C_1
$A = a_{n-1}$ a_{n-2} ... a_k ... a_1 a_0
+
$B = b_{n-1}$ b_{n-2} ... b_k ... b_1 b_0
C_n S_{n-1} C_{n-1} S_{n-2} C_{k+1} S_k C_2 S_1 C_1 S_0


b_{n-1} a_{n-1} b_{n-2} a_{n-2} b_1 a_1 b_0 a_0
C_n FA C_{n-1} FA C_{n-2} .... C_2 FA C_1 FA
S_{n-1} S_{n-2} S_1 S_0

Gate Level Logic

Gate Primitives

Majority Circuit



Gate Level Logic

Gate Primitives

Majority Circuit

```
module maj3 (a, b, c, y);
    input a, b, c;
    output y;
    wire im1, im2, im3;

    and #(2, 4)
        ( im1, a, b ),
        ( im2, b, c ),
        ( im3, c, a );
    or #(3, 5) (y, im1, im2, im3);

endmodule
```

Assign Statements

Bitwise operators

maj3 with assign Statement

```
`timescale 1ns/100ps

module maj3 (input a, b, c, output y);
   assign #(4) y = ( a & b ) | ( b & c ) | ( a & c );
endmodule
```

Gate Level Logic

Gate Primitives

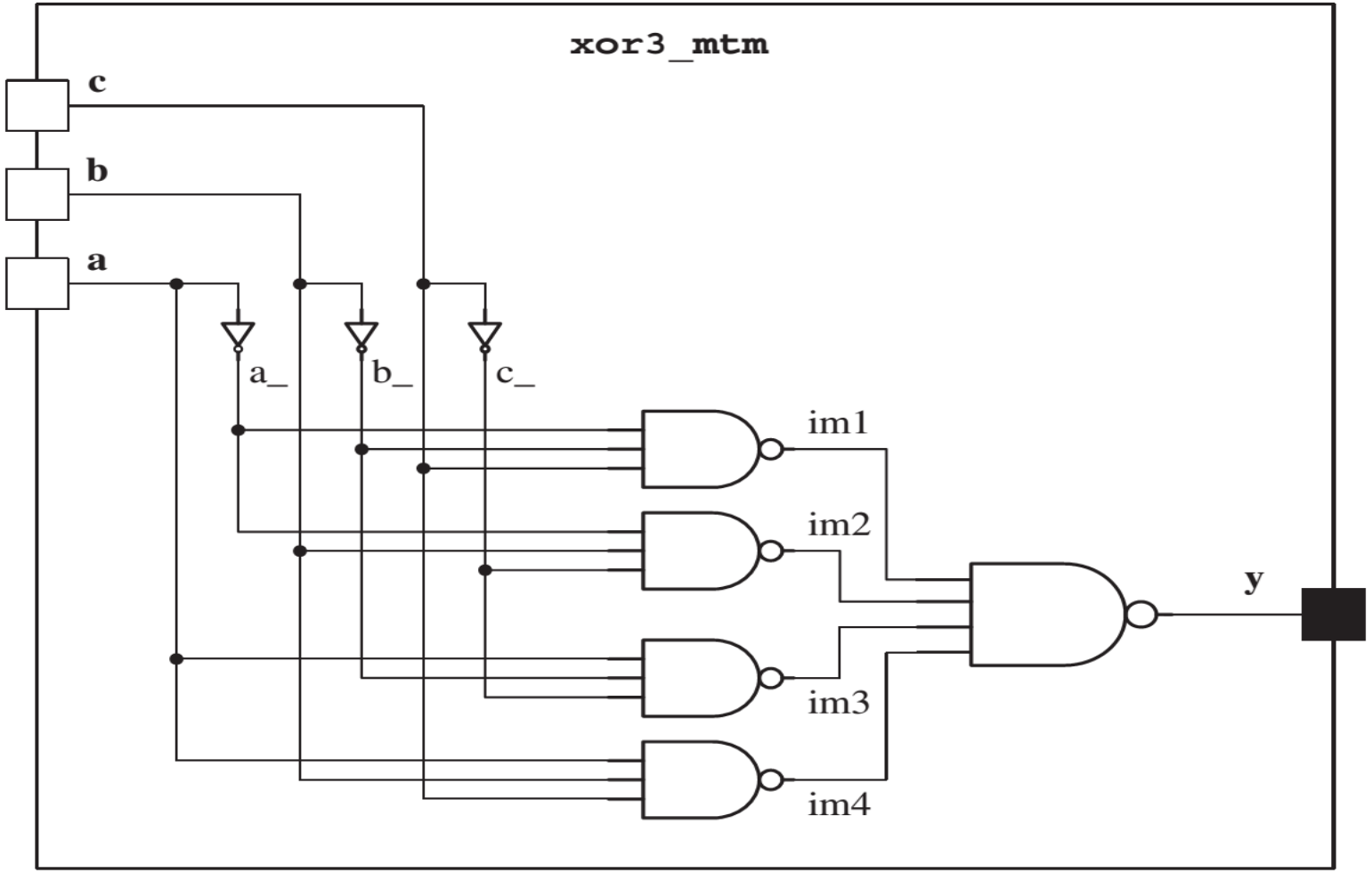
Test for Majority Circuit

```
module test_maj;
    // Inputs
    reg a;
    reg b;
    reg c;
    // Outputs
    wire y;
    // Instantiate the Unit Under Test (UUT)
    maj3 uut (
        .a(a),
        .b(b),
        .c(c),
        .y(y));
    initial begin
        // Initialize Inputs
        a = 0; b = 0; c = 0;
        #20 a = 1;
        #5  b = 1;
        #20 a = 0; b = 0; c = 0;
        #20 a = 1;
        #5  c = 1;
        #20 a = 0; b = 0; c = 0;
        #20 b = 1;
        #5  c = 1;
        #20 a = 0; b = 0; c = 0;
    end

endmodule
```

Gate Level Logic

Three input XOR

XOR Circuit



Gate Level Logic

Three input XOR

XOR Circuit

```
module xor3_mtm (input a, b, c, output y);
    wire a_, b_, c_;
    wire im1, im2, im3, im4;

    not #(1:3:5, 2:4:6)
        ( a_, a ),
        ( b_, b ),
        ( c_, c );
    nand #(2:4:6, 3:5:7)
        ( im1, a_, b_, c ),
        ( im2, a_, b, c_ ),
        ( im3, a, b_, c_ ),
        ( im4, a, b, c );
    nand #(2:4:6, 3:5:7) (y, im1, im2, im3, im4);

endmodule
```

Min : type : Max Delay

Assign Statements

Bitwise operators

xor3 with assign Statement

```
`timescale 1ns/100ps

module xor3 (input a, b, c, output y);
    assign y = a ^ b ^ c;
endmodule
```

Gate Level Logic

Three input XOR

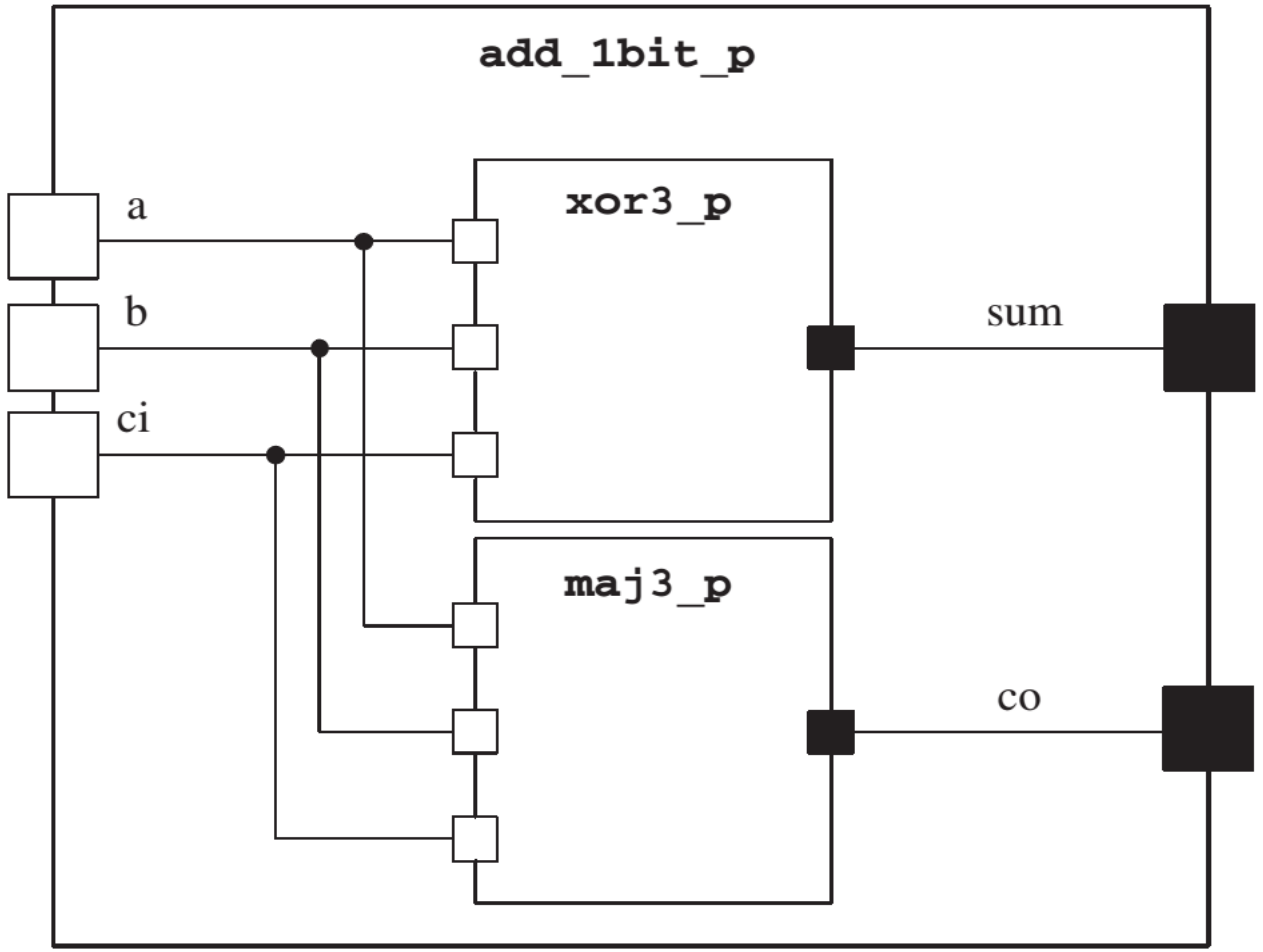
Test for XOR Circuit

```
module test_xor3;

    // Inputs
    reg a;
    reg b;
    reg c;

    // Outputs
    wire y;

    // Instantiate the Unit Under Test (UUT)
    xor3_mtm uut (
        .a(a),
        .b(b),
        .c(c),
        .y(y)
    );

initial repeat (9) #10 a = ~a;
initial repeat (5) #20 b = ~b;
initial repeat (5) #40 c = ~c;
initial begin
    // Initialize Inputs
    a = 0;
    b = 0;
    c = 0;

    // Wait 100 ns for global reset to finish
    #100;

    // Add stimulus here

end
```

Hierarchical Structures

Simple hierarchies

Full Adder Using xor3_p and maj3_p



Assign Statements

Bitwise operators

Add 1 bit with assign Statement

```
`timescale 1ns/100ps

module add_1bit (input a, b, ci, output s, co);

   assign #(10) s = a ^ b ^ ci;
   assign #(8) co = ( a & b ) | ( b & ci ) | ( a & ci );

endmodule
```

Assign Statements

Bitwise operators

Full Adder Using Concatenation

```
`timescale 1ns/100ps

module add_1bit (input a, b, ci, output s, co);
  assign #(3, 4) {co, s} = {(a & b)|(b & ci)|(a & ci), a^b^ci};
endmodule
```

Hierarchical Structures

Simple hierarchies

Full Adder Using xor3_p and maj3_p

```
`timescale 1ns/100ps

module add_1bit_p (input a, b, ci, output sum, co);

    xor3_p xr1 (a, b, ci, sum);
    maj3_p mj1 (a, b, ci, co);

endmodule
```

Hierarchical Structures

Simple hierarchies

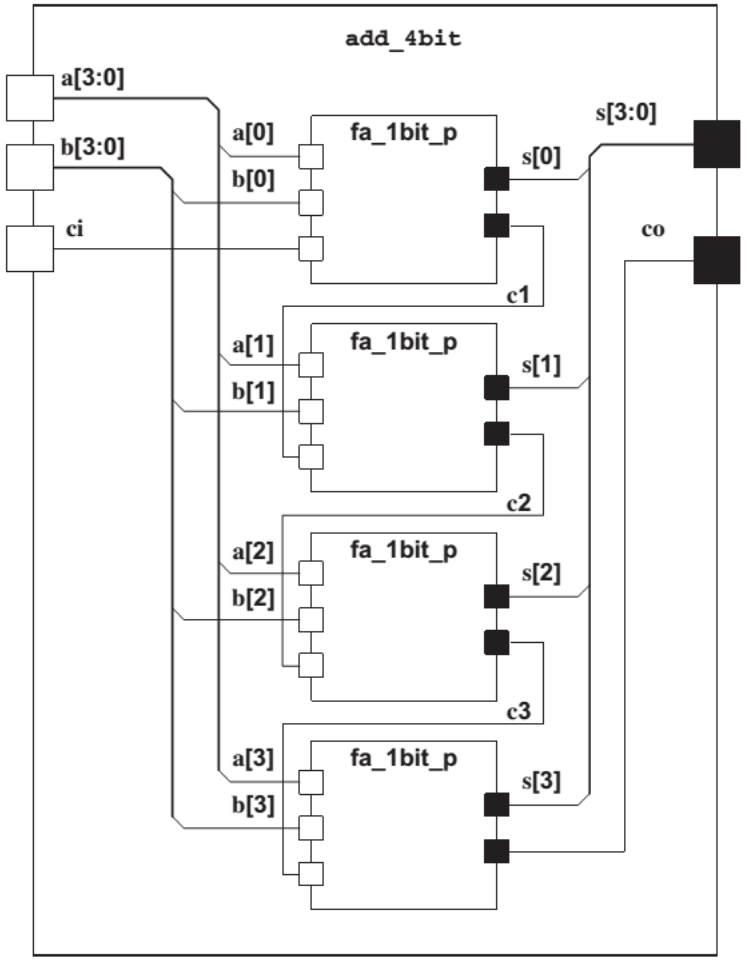
Test for Full Adder Using xor3_p and maj3_p

```
module test_1bit_p;
    // Inputs
    reg a;
    reg b;
    reg ci;
    // Outputs
    wire s;
    wire co;
    // Instantiate the Unit Under Test (UUT)
    add_1bit_p uut (
        .a(a),
        .b(b),
        .ci(ci),
        .sum(s),
        .co(co));
    initial begin
        // Initialize Inputs
        a = 0;
        b = 0;
        ci = 0;
        #50 a = 1;
        #50 b = 1;
        #50 ci= 1;
        #50 a = 0;
        #50 b = 0;
        // Wait 100 ns for global reset to finish
        #100;
        // Add stimulus here
    end
endmodule
```

Hierarchical Structures

Vector declarations

A 4-bit Adder by Wiring Four Full Adders



Hierarchical Structures

Vector declarations

A 4-bit Adder by Wiring Four Full Adders

```
`timescale 1ns/100ps

module add_4bit (input [3:0] a, b, input ci,
                 output [3:0] s, output co);
   wire c1, c2, c3;

   add_1bit_p fa0 (a[0], b[0], ci, s[0], c1);
   add_1bit_p fa1 (a[1], b[1], c1, s[1], c2);
   add_1bit_p fa2 (a[2], b[2], c2, s[2], c3);
   add_1bit_p fa3 (a[3], b[3], c3, s[3], co);

endmodule
```

OR

```
add_1bit_p fa1 (.a(a[1]),.b(b[1],.ci(c1),.sum(s[1]),.co(c2));
```

Hierarchical Structures

Iterative structures

4-bit Adder Using Array of Instances

```
`timescale 1ns/100ps

module add_4bit_vec (input [3:0] a, b, input ci,
                     output [3:0] s, output co);
   wire c1, c2, c3;
   add_1bit_p fa[3:0] (a, b, {c3, c2, c1, ci}, s,
                             {co, c3, c2, c1});

endmodule
```

Hierarchical Structures

Vector declarations

A 4-bit Adder by Wiring Four Full Adders

```
module test_add4;
    // Inputs
    reg [3:0] a;
    reg [3:0] b;
    reg ci;
    // Outputs
    wire [3:0] s;
    wire co;
    // Instantiate the Unit Under Test (UUT)
    add_4bit uut (
        .a(a),
        .b(b),
        .ci(ci),
        .s(s),
        .co(co));
    initial begin
        // Initialize Inputs
        a = 0;
        b = 0;
        ci = 0;
        #50 a = 4'b1010; b = 4'b1001;
        #50 a = 4'b0010; b = 4'b1001;
        #50 a = 4'b1011; b = 4'b1101; ci = 1;
        // Wait 100 ns for global reset to finish
        #100;
        // Add stimulus here
    end
endmodule
```