**// 4-bit by 4-bit array like unsigned Multiplier Structural Design**

module half\_adder(a,b,s,cout);

input a,b;

output s,cout;

assign s = a ^ b;

assign cout = (a & b);

endmodule

module full\_adder(a,b,cin,s,cout);

input a,b, cin;

output s,cout;

assign s = a ^ b ^ cin;

assign cout =(a & b) | (b & cin) | (cin & a);

endmodule

module mult\_4\_bit(x,y,p);

input [3:0] x,y;

output [7:0] p;

assign p[0] = x[0]&y[0];

wire r1c1,r1c2,r1c3,r1c4,r1c5;

wire r2c1,r2c2,r2c3,r2c4,r2c5;

wire r3c1,r3c2,r3c3,r3c4,r3c5;

wire r4c1,r4c2;

half\_adder ha1(x[0]&y[1], x[1]&y[0],p[1],r1c1);

half\_adder ha2(x[0]&y[2],x[1]&y[1],r1c2,r1c3);

half\_adder ha3(x[0]&y[3],x[1]&y[2],r1c4,r1c5);

full\_adder fa1(x[2]&y[0],r1c2,r1c3,p[2],r2c1);

full\_adder fa2(x[2]&y[1],r1c3,r1c4,r2c2,r2c3);

full\_adder fa3(x[2]&y[2],r1c5,x[1]&y[3],r2c4,r2c5);

full\_adder fa4(x[3]&y[0],r2c1,r2c2,p[3],r3c1);

full\_adder fa5(x[3]&y[1],r2c3,r2c4,r3c2,r3c3);

full\_adder fa6(x[3]&y[2],r2c5,x[2]&y[3],r3c4,r3c5);

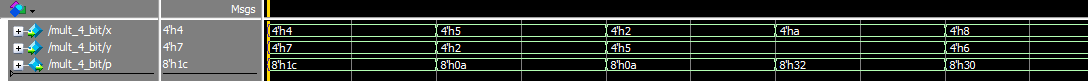
half\_adder ha4(r3c1,r3c2,p[4],r4c1);

full\_adder fa7(r3c3,r4c1,r3c4,p[5],r4c2);

full\_adder fa8(x[3]&y[3],r3c5,r4c2,p[6],p[7]);

endmodule

**Simulation done successfully in MODELSIM:**



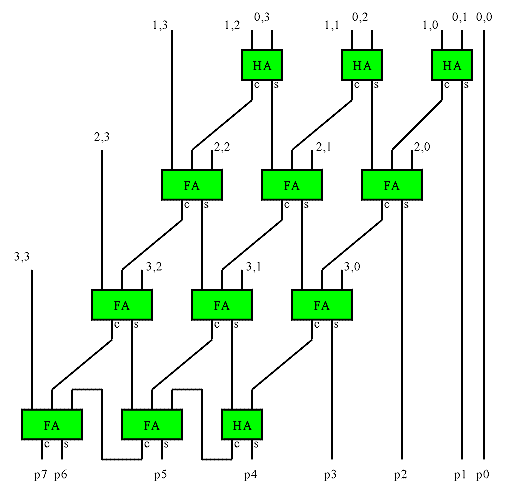
REFERENCE:

1.Look at [this excellent paper by Ray Andraka](http://www.andraka.com/multipli.php) for an overview.

In your case, [Booth's algorithm](https://en.m.wikipedia.org/wiki/Booth%27s_multiplication_algorithm) would be a good approach.

[share](https://electronics.stackexchange.com/a/268894)[improve this answer](https://electronics.stackexchange.com/posts/268894/edit)

<http://www.andraka.com/multipli.php>

[](http://www.andraka.com/multipli.php#home)

// 4-bit by 4-bit Buagh Wooley like Multiplier Structural Design

module half\_adder(a,b,s,cout);

input a,b;

output s,cout;

assign s = a ^ b;

assign cout = (a & b);

endmodule

module full\_adder(a,b,cin,s,cout);

input a,b, cin;

output s,cout;

assign s = a ^ b ^ cin;

assign cout =(a & b) | (b & cin) | (cin & a);

endmodule

module AND(a,b,c);

input a,b;

output c;

assign c = a & b;

endmodule

module mult\_4\_bit(x,y,p);

input [3:0] x,y;

output [7:0] p;

wire o1,o2,o3,o4,o5,o6,o7,o8,o9,o10,o11,o12,o13,o14,o15;

wire r1c1,r1c2,r1c3,r1c4,r1c5;

wire r2c1,r2c2,r2c3,r2c4,r2c5;

wire r3c1,r3c2,r3c3,r3c4,r3c5;

wire r4c1,r4c2;

assign p[0] = x[0]&y[0];

AND and1(.a(x[0]),.b(y[1]),.c(o1));

AND and2(.a(x[1]),.b(y[0]),.c(o2));

AND and3(.a(x[0]),.b(y[2]),.c(o3));

AND and4(.a(x[1]),.b(y[1]),.c(o4));

AND and5(.a(x[0]),.b(y[3]),.c(o5));

AND and6(.a(x[1]),.b(y[2]),.c(o6));

AND and7(.a(x[2]),.b(y[0]),.c(o7));

AND and8(.a(x[2]),.b(y[1]),.c(o8));

AND and9(.a(x[2]),.b(y[2]),.c(o9));

AND and10(.a(x[1]),.b(y[3]),.c(o10));

AND and11(.a(x[3]),.b(y[0]),.c(o11));

AND and12(.a(x[3]),.b(y[1]),.c(o12));

AND and13(.a(x[3]),.b(y[2]),.c(o13));

AND and14(.a(x[2]),.b(y[3]),.c(o14));

AND and15(.a(x[3]),.b(y[3]),.c(o15));

half\_adder ha1(.a(o1), .b(o2),.s(p[1]),.cout(r1c1));

half\_adder ha2(.a(o3),.b(o4),.s(r1c2),.cout(r1c3));

half\_adder ha3(.a(o5),.b(o6),.s(r1c4),.cout(r1c5));

full\_adder fa1(.a(o7),.b(r1c2),.cin(r1c3),.s(p[2]),.cout(r2c1));

full\_adder fa2(.a(o8),.b(r1c3),.cin(r1c4),.s(r2c2),.cout(r2c3));

full\_adder fa3(.a(o9),.b(r1c5),.cin(o10),.s(r2c4),.cout(r2c5));

full\_adder fa4(.a(o11),.b(r2c1),.cin(r2c2),.s(p[3]),.cout(r3c1));

full\_adder fa5(.a(o12),.b(r2c3),.cin(r2c4),.s(r3c2),.cout(r3c3));

full\_adder fa6(.a(o13),.b(r2c5),.cin(o14),.s(r3c4),.cout(r3c5));

half\_adder ha4(.a(r3c1),.b(r3c2),.s(p[4]),.cout(r4c1));

full\_adder fa7(.a(r3c3),.b(r4c1),.cin(r3c4),.s(p[5]),.cout(r4c2));

full\_adder fa8(.a(o15),.b(r3c5),.cin(r4c2),.s(p[6]),.cout(p[7]));

endmodule

**SIMULATION RESULTS:**

