California State University, Fullerton

Computer Engineering

**EGCP 446 – Advanced Digital Design using Verilog HDL**

**(Fall 2018)**

**Practice Lab No 2: Structural Design and Conditional Statement in HDL**

1. **Lab Description**

**Part A: Full Adder**

Students are asked to design a 1-bit Full adder using HDL. Use the port names based on the picture shown below. Write a testbench code to verify your design.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Cin** | **A** | **B** | **Cout** | **Sum** | | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 1 | 0 | 1 | | 0 | 1 | 0 | 0 | 1 | | 0 | 1 | 1 | 1 | 0 | | 1 | 0 | 0 | 0 | 1 | | 1 | 0 | 1 | 1 | 0 | | 1 | 1 | 0 | 1 | 0 | | 1 | 1 | 1 | 1 | 1 |   **Truth Table** |

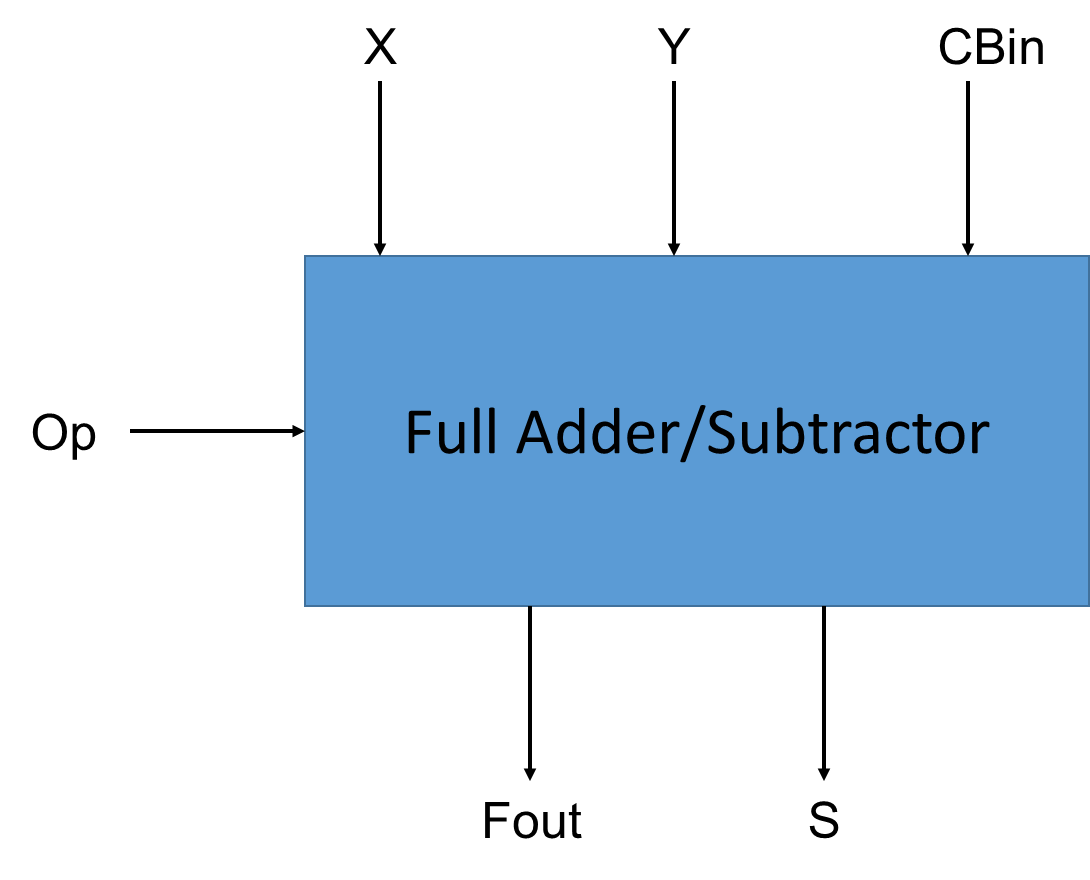
**Part B: Full Subtractor**

Now create a 1-bit full subtractor using HDL. Use the port names based on the picture shown below. Write a testbench code to verify your design

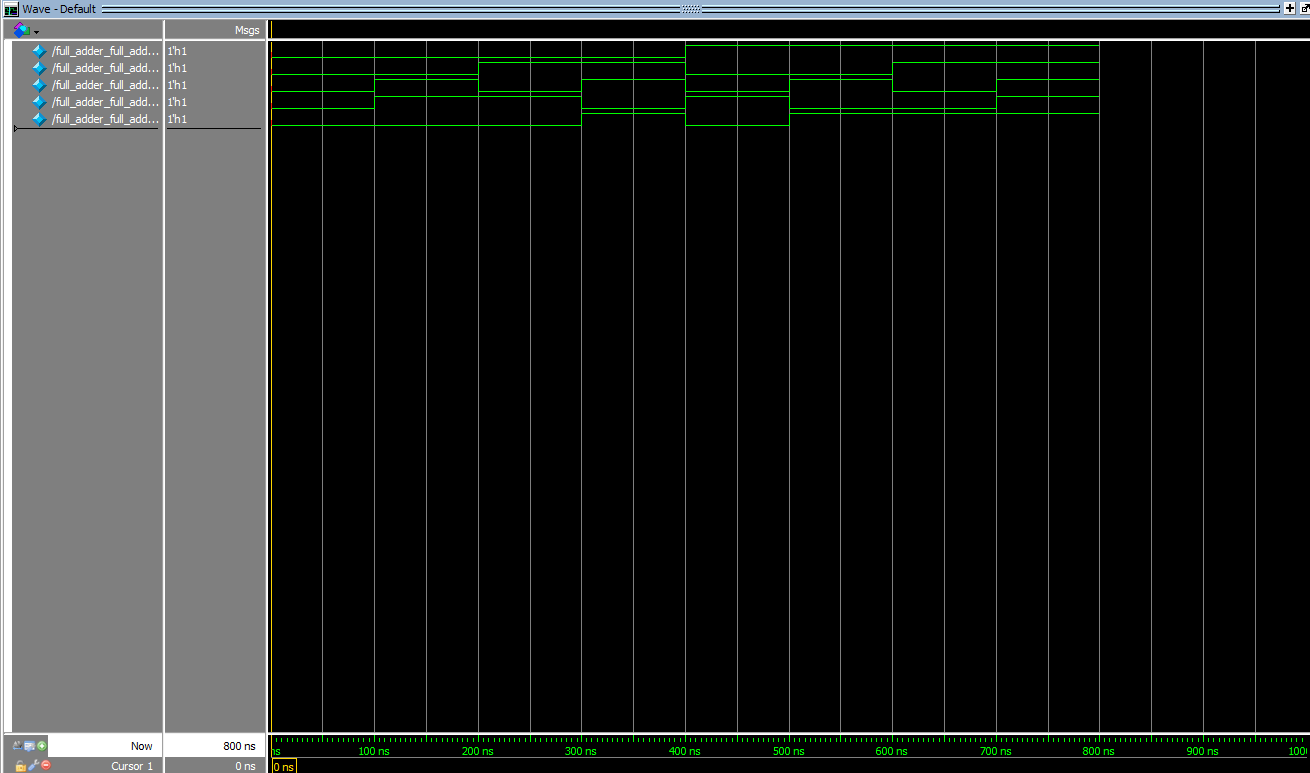
|  |
| --- |
| **Truth Table** |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **A** | **B** | **Bin** | **Bout** | **D** | | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 1 | 1 | 1 | | 0 | 1 | 0 | 1 | 1 | | 0 | 1 | 1 | 1 | 0 | | 1 | 0 | 0 | 0 | 1 | | 1 | 0 | 1 | 0 | 0 | | 1 | 1 | 0 | 0 | 0 | | 1 | 1 | 1 | 1 | 1 | |

**Part C: 1-Bit Adder/Subtractor Circuit**

Create a block diagram by using a selector switch (Multiplexer) to create a 1-bit adder/subtractor by using HDL code used in Part A and Part B. Write a testbench code to verify your design. Use the port name shown below.



Write your answers from here.



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

input a,b,cin;

output sum,cout;

//wire xor1, and1,and2,and3;

//xor (xor1, a, b);

//xor (sum, xor1, cin);

// and (and1, a, b);

//and (and2, a, cin);

//and (and3, cin, b);

// or (cout, and1, and2, and3);

assign sum = (cin ^ (a ^ b));

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

endmodule

module full\_adder\_full\_adder\_tb\_v\_tf();

reg a;

reg b;

reg cin;

wire sum;

wire cout;

full\_adder uut (

.a(a),

.b(b),

.cin(cin),

.sum(sum),

.cout(cout)

);

initial begin

a = 0;

b = 0;

cin = 0;

#100;

a = 0;

b = 0;

cin = 1;

#100;

a = 0;

b = 1;

cin = 0;

#100;

a = 0;

b = 1;

cin = 1;

#100;

a = 1;

b = 0;

cin = 0;

#100;

a = 1;

b = 0;

cin = 1;

#100;

a = 1;

b = 1;

cin = 0;

#100;

a = 1;

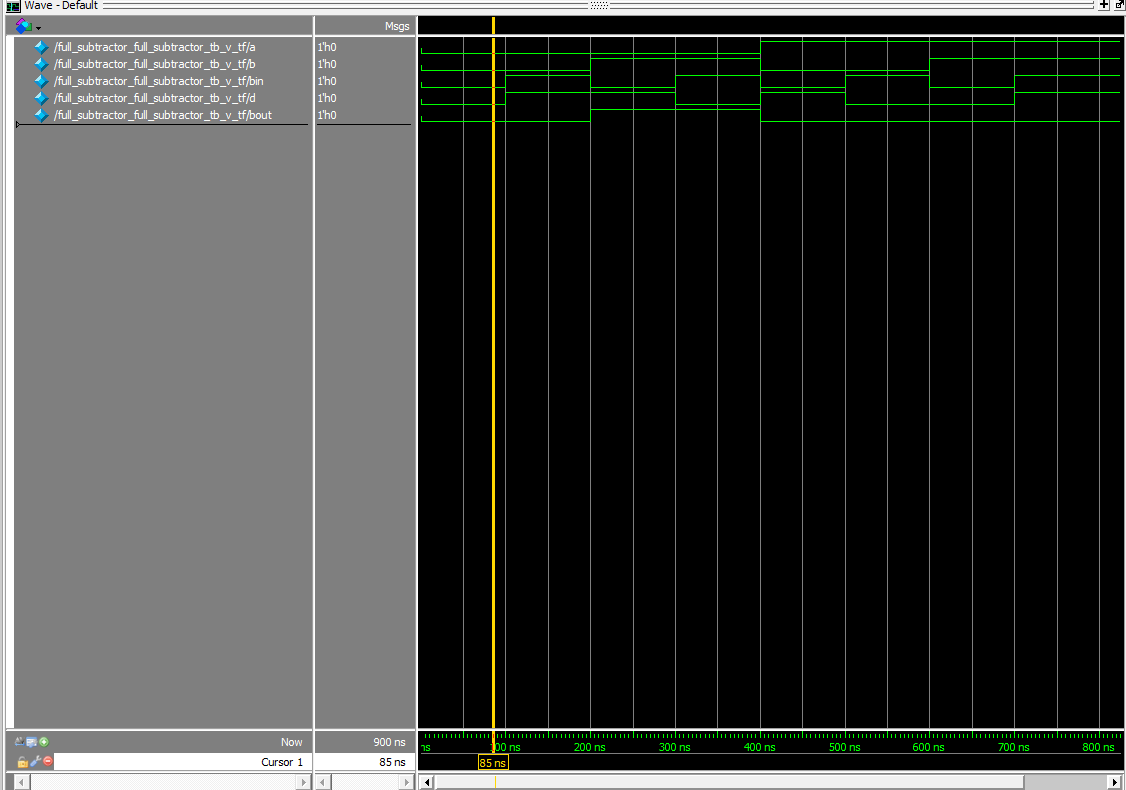
b = 1;

cin = 1;

end

endmodule

---------------------------------------------------------------------------------------



module full\_subtractor(a,b,bin,d,bout);

input a,b,bin;

output d,bout;

// wire xor1, and1, and2, not1, not2;

//not (not1, a);

//xor (xor1, a, b);

//and (and1, not1, b);

//not (not2, xor1);

//xor (d, bin, xor1);

//and (and2, not2, and1);

//or (bout, and2, and1);

assign d = (bin ^ (a ^ b));

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

endmodule

module full\_subtractor\_full\_subtractor\_tb\_v\_tf();

reg a;

reg b;

reg bin;

wire d;

wire bout;

full\_subtractor uut (

.a(a),

.b(b),

.bin(bin),

.d(d),

.bout(bout)

);

initial begin

a = 0;

b = 0;

bin = 0;

#100;

a = 0;

b = 0;

bin = 1;

#100;

a = 0;

b = 1;

bin = 0;

#100;

a = 0;

b = 1;

bin = 1;

#100;

a = 1;

b = 0;

bin = 0;

#100;

a = 1;

b = 0;

bin = 1;

#100;

a = 1;

b = 1;

bin = 0;

#100;

a = 1;

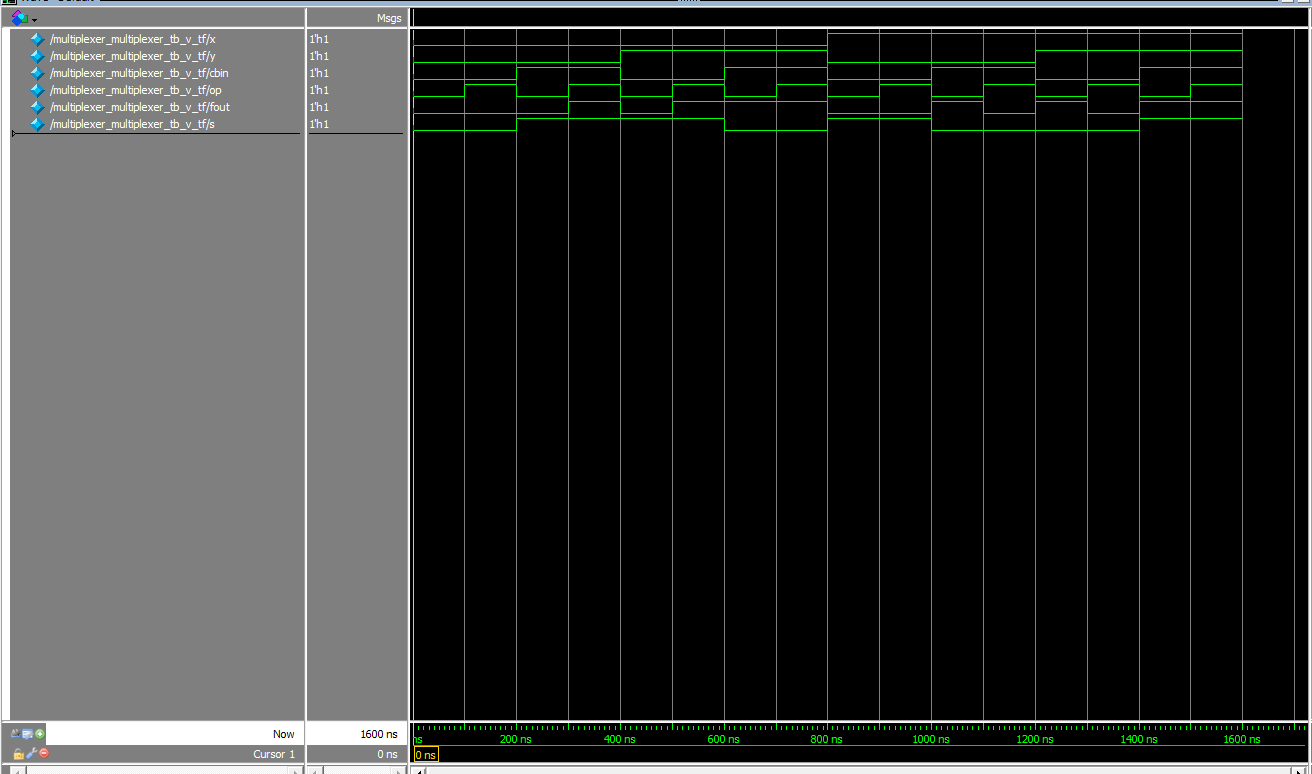
b = 1;

bin = 1;

end

endmodule

-------------------------------------------------------



module multiplexer(x, y, cbin,op,fout,s);

input x, y, cbin,op;

output reg fout,s;

always @\*

if (op == 1'b0)

begin

s = (cbin ^ (x ^ y));

fout = (x & y) | (x & cbin) | (y & cbin);

end

else

begin

s = cbin ^ (x ^ y);

fout = ((~x) & y) | ((~(x ^ y)) & cbin);

end

endmodule

module multiplexer\_multiplexer\_tb\_v\_tf();

reg x;

reg y;

reg cbin;

reg op;

wire fout;

wire s;

multiplexer uut (

.x(x),

.y(y),

.cbin(cbin),

.op(op),

.fout(fout),

.s(s)

);

initial begin

x = 0;

y = 0;

cbin = 0;

op = 0;

#100;

x = 0;

y = 0;

cbin = 0;

op = 1;

#100;

x = 0;

y = 0;

cbin = 1;

op = 0;

#100;

x = 0;

y = 0;

cbin = 1;

op = 1;

#100;

x = 0;

y = 1;

cbin = 0;

op = 0;

#100;

x = 0;

y = 1;

cbin = 0;

op = 1;

#100;

x = 0;

y = 1;

cbin = 1;

op = 0;

#100;

x = 0;

y = 1;

cbin = 1;

op = 1;

#100;

x = 1;

y = 0;

cbin = 0;

op = 0;

#100;

x = 1;

y = 0;

cbin = 0;

op = 1;

#100;

x = 1;

y = 0;

cbin = 1;

op = 0;

#100;

x = 1;

y = 0;

cbin = 1;

op = 1;

#100;

x = 1;

y = 1;

cbin =0;

op = 0;

#100;

x = 1;

y = 1;

cbin = 0;

op = 1;

#100;

x = 1;

y = 1;

cbin = 1;

op = 0;

#100;

x = 1;

y = 1;

cbin = 1;

op = 1;

end

endmodule