Project 1

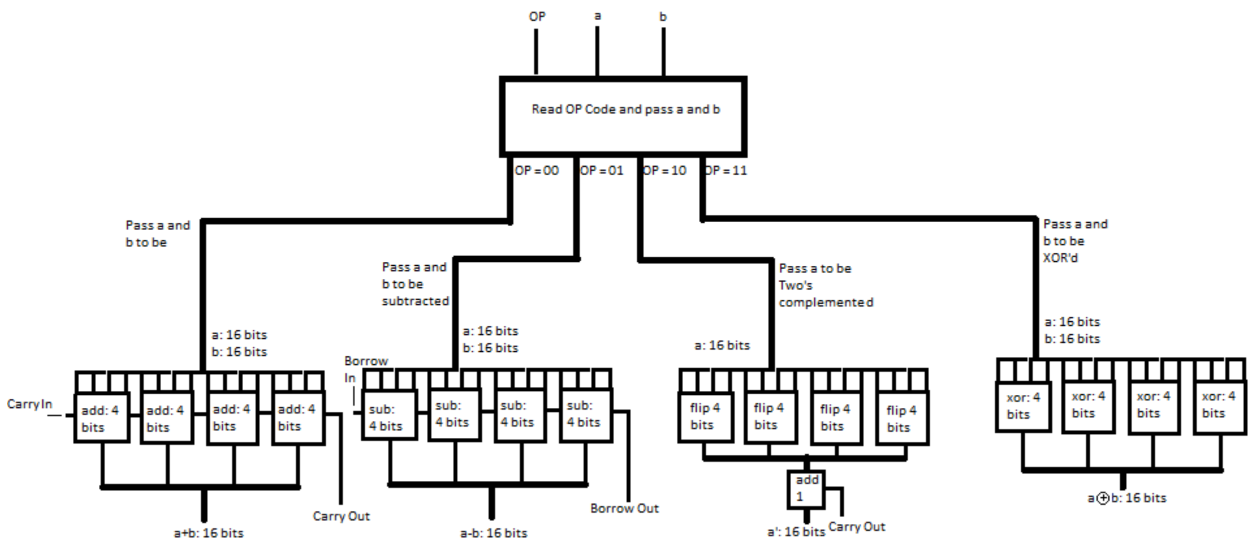
Professor Umer Cheema

Design of a 16-bit Arithmetic Logic Unit

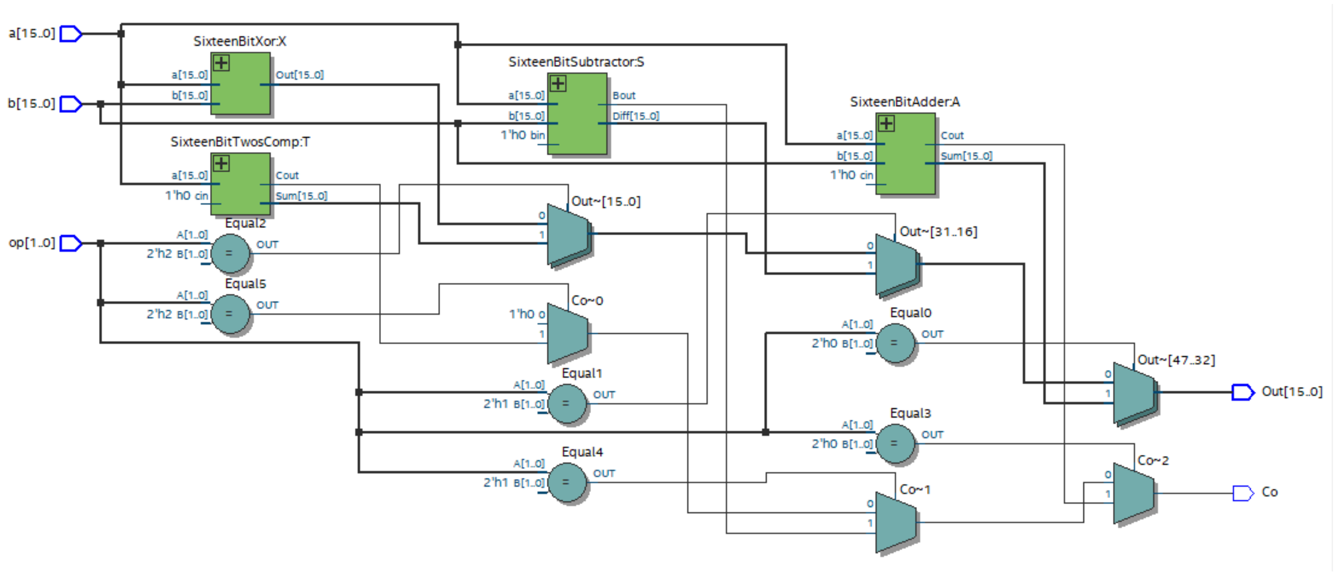
Sean Atangan 655409356

10/17/2021

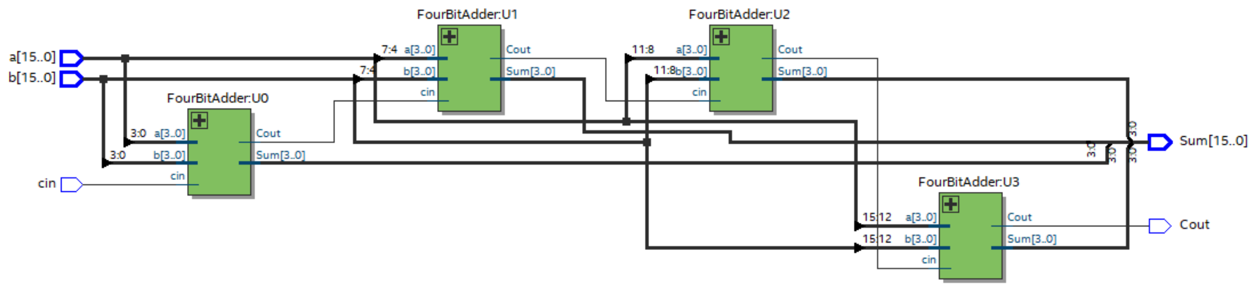
Block Diagram



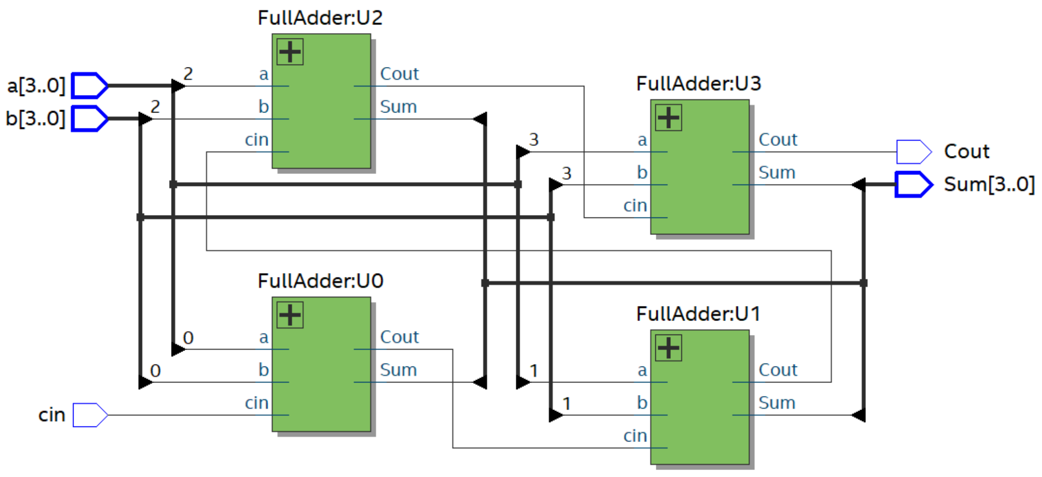
Sixteen Bit ALU



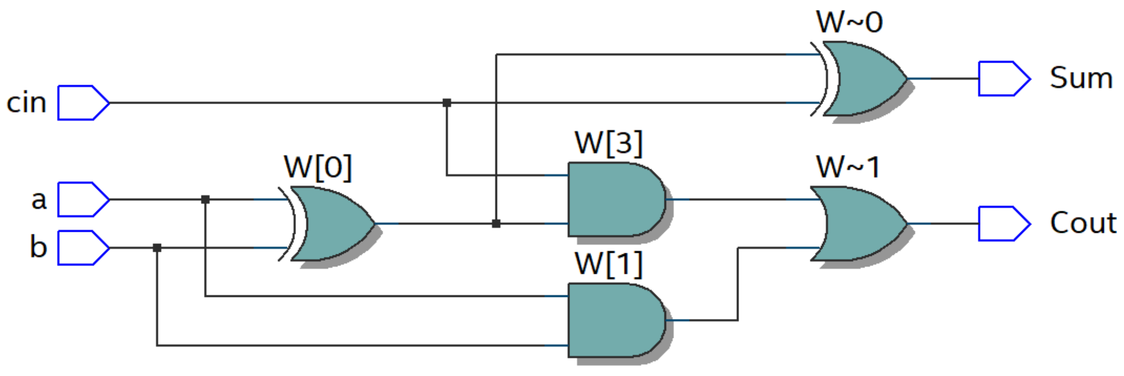
Sixteen Bit Adder



Four Bit Adder



Full Adder

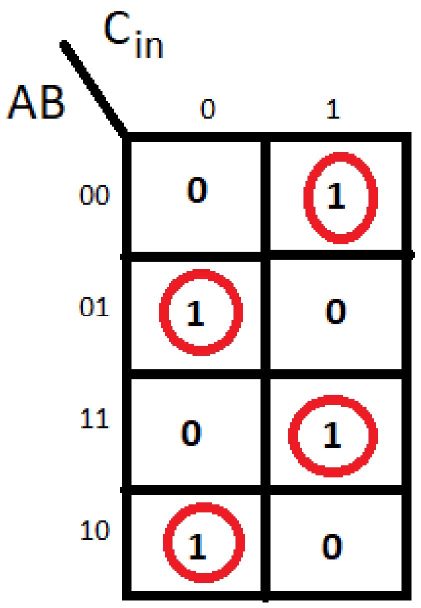


Minimization Logic for Full Adder:

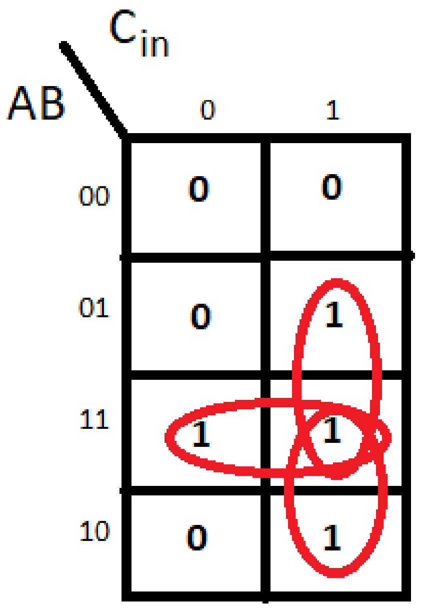
Truth Table

A	B	Cin	S	Cout
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

K-Maps

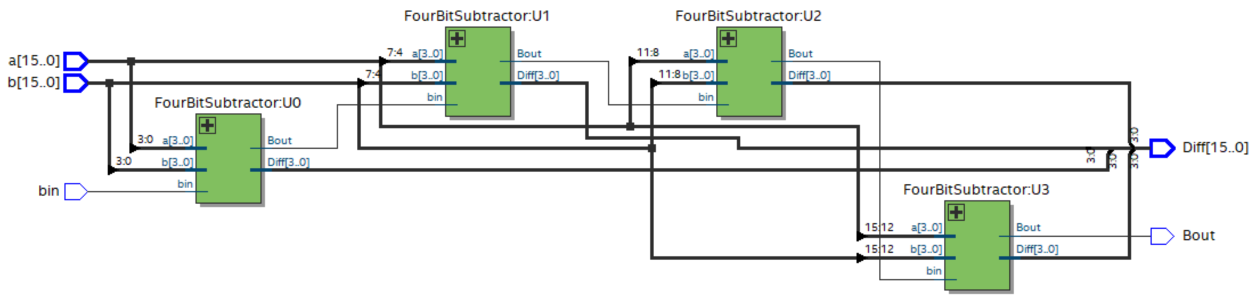


S = A or B or C

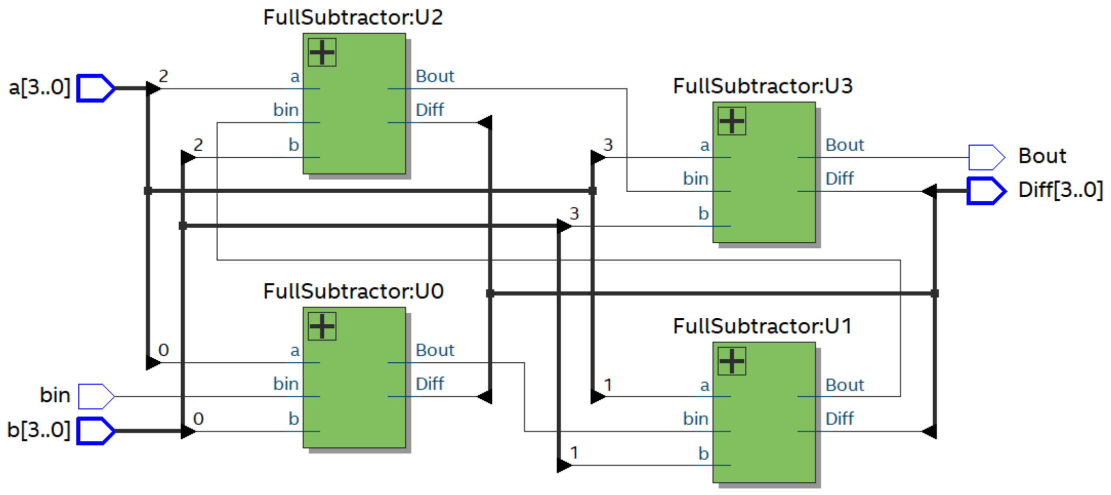


Cout = AB + BC + AC
= (A or B)C + AB

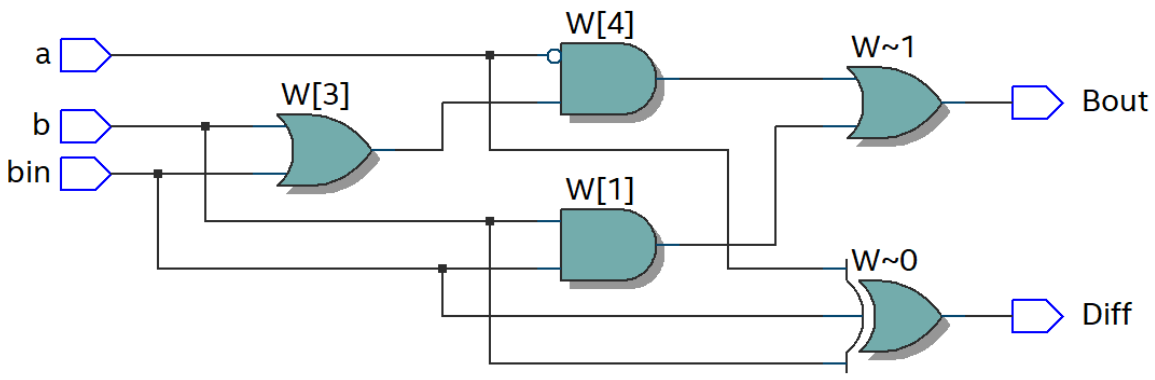
Sixteen Bit Subtractor



Four Bit Subtractor



Full Subtractor

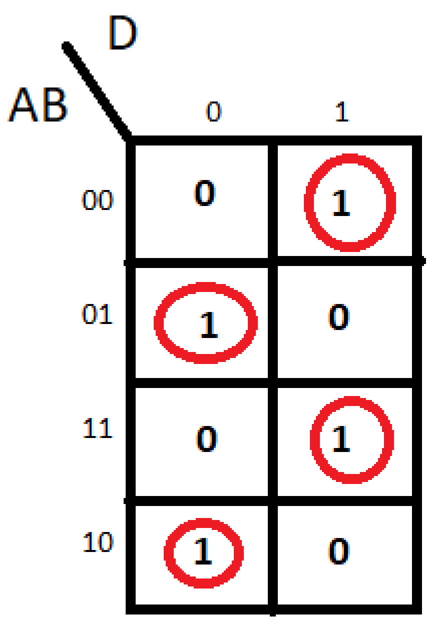


Minimization Logic for Full Subtractor:

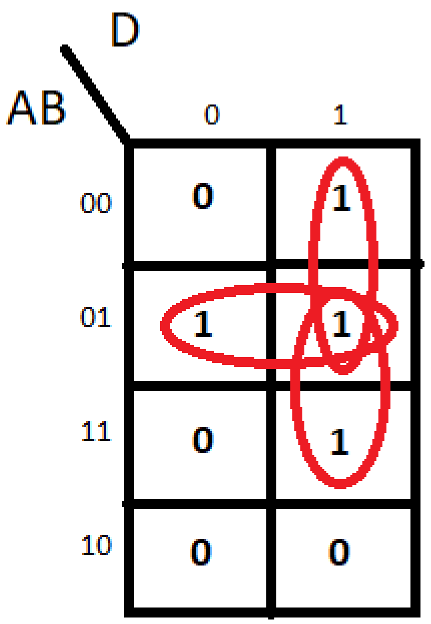
Truth Table

A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-Maps

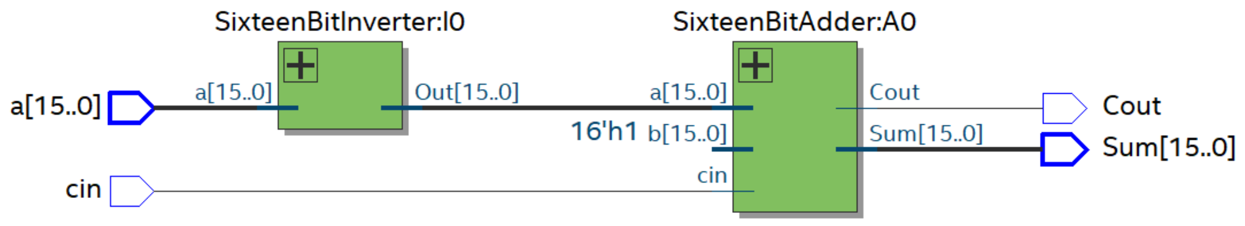


Diff = A or B or D

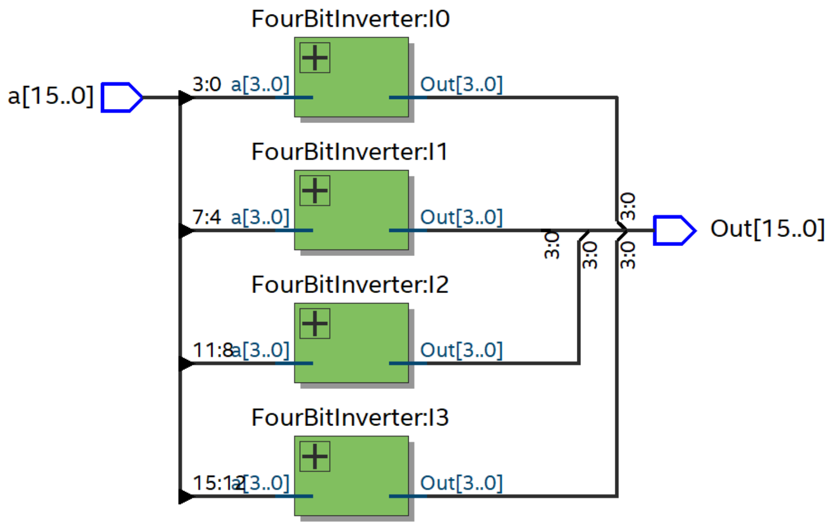


Bout = A'D + BD + A'B
= A'(B+D) + BD

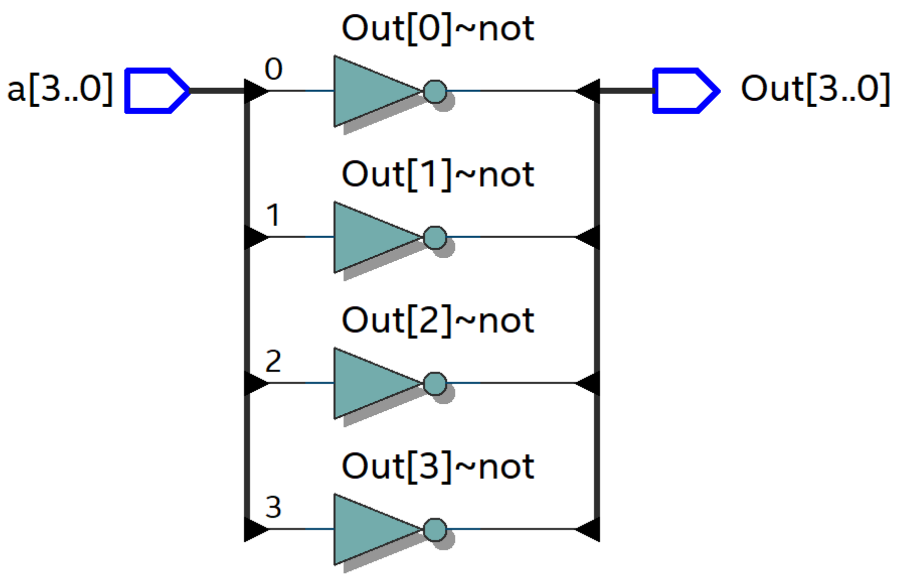
Sixteen Bit Two's Complement



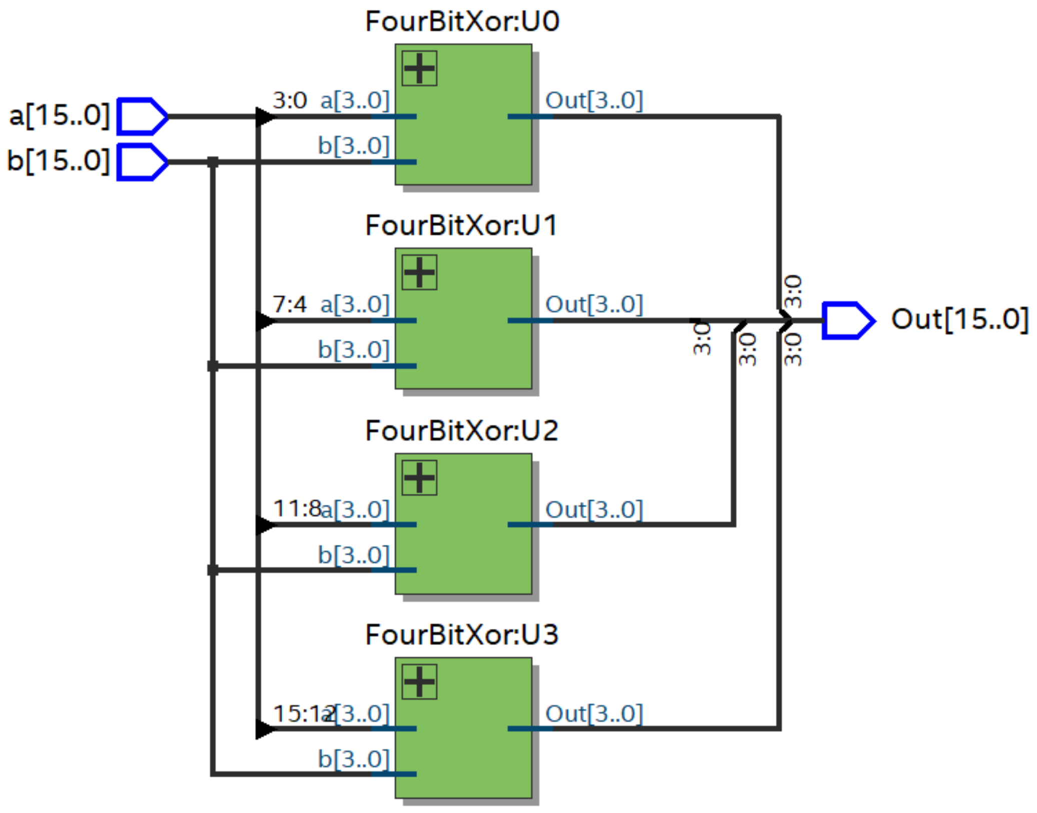
Sixteen Bit Inverter



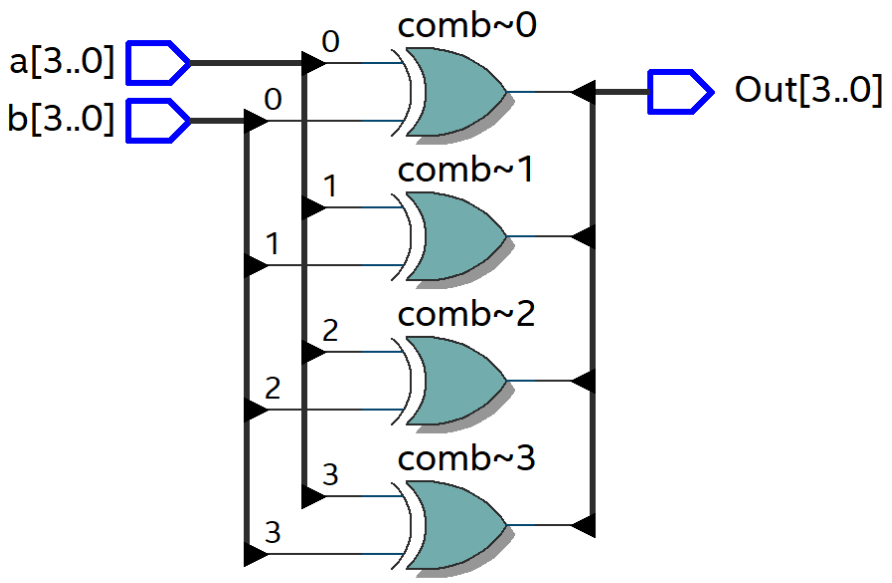
Four Bit Inverter



Sixteen Bit XOR

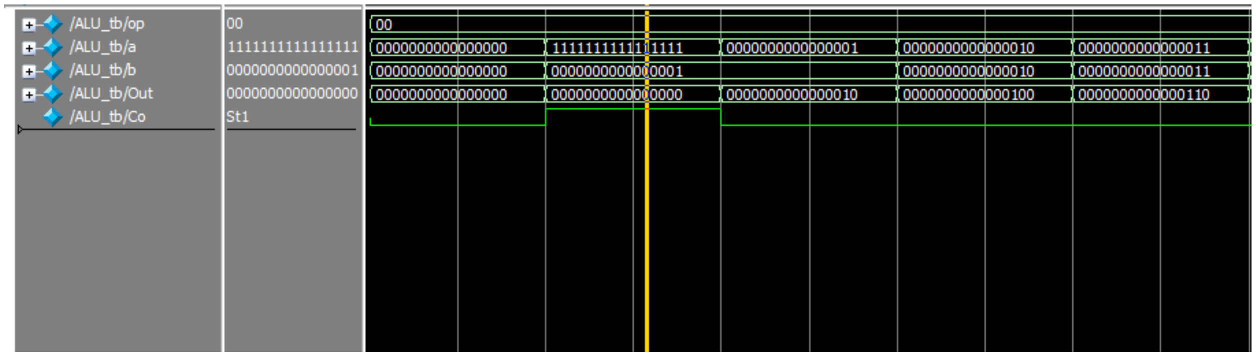


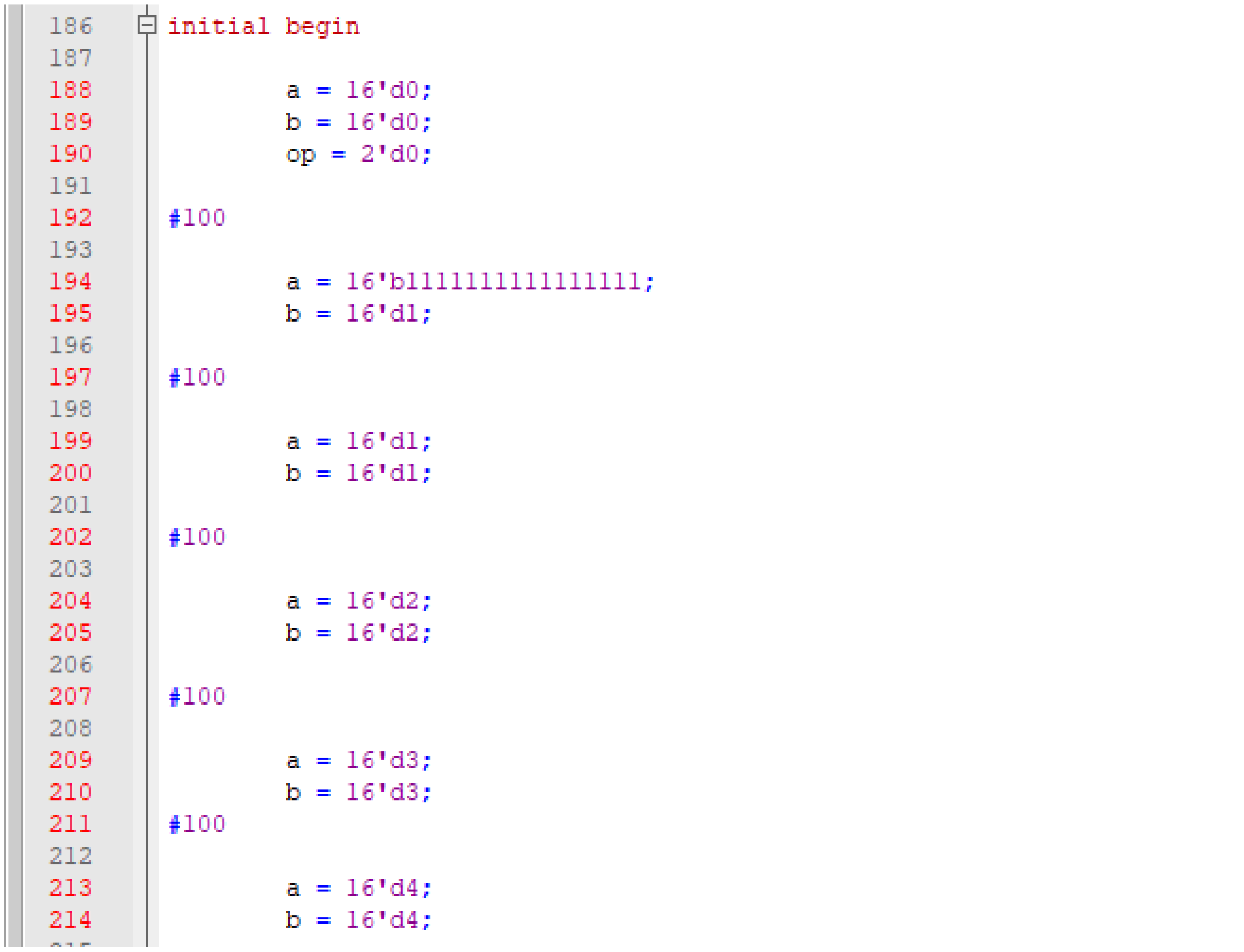
Four Bit XOR



<u>**Testing:**</u>

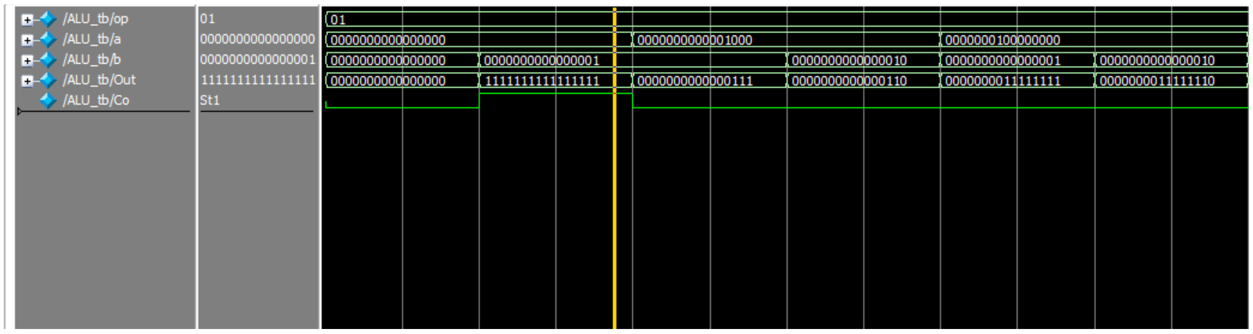
Testing 16-bit adder for general output and carry bit: Passed



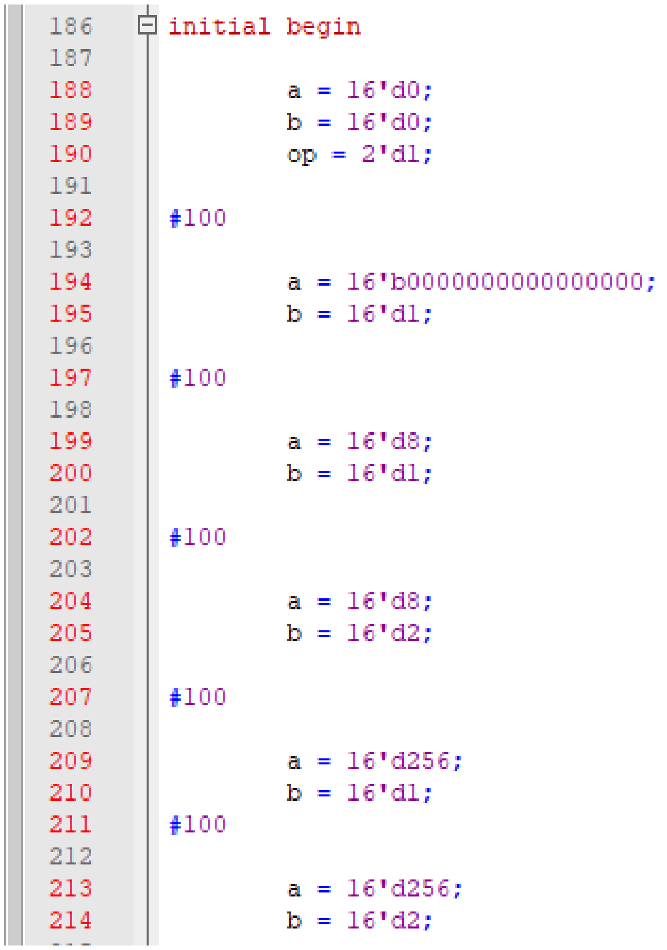


```
initial begin

        a = 16'd0;
        b = 16'd0;
        op = 2'd0;

#100

        a = 16'b1111111111111111;
        b = 16'd1;

#100

        a = 16'd1;
        b = 16'd1;

#100

        a = 16'd2;
        b = 16'd2;

#100

        a = 16'd3;
        b = 16'd3;
#100

        a = 16'd4;
        b = 16'd4;
```

Testing 16-bit subtracter for general output and borrow bit: Passed

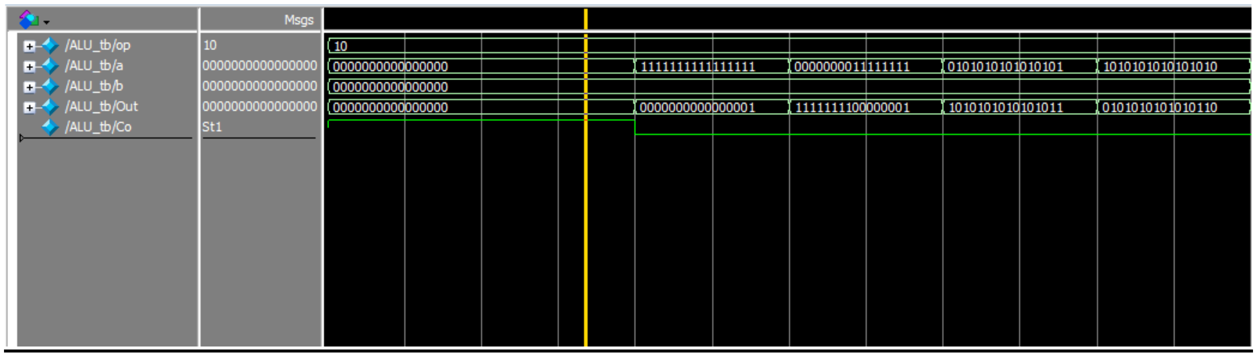


```
/ALU_tb/op    01                 01
/ALU_tb/a     0000000000000000   0000000000000000                                    0000000000001000                                    0000000100000000
/ALU_tb/b     0000000000000001   0000000000000000   0000000000000001                                    0000000000000010   0000000000000001   0000000000000010
/ALU_tb/Out   1111111111111111   0000000000000000   1111111111111111   0000000000000111   0000000000000110   0000000011111111   0000000011111110
/ALU_tb/Co    St1
```

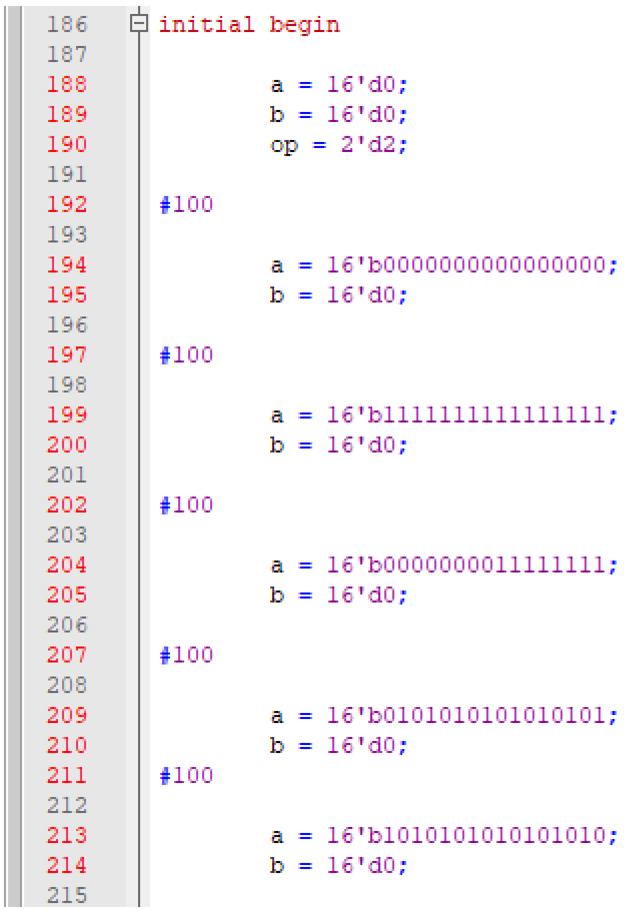


```
initial begin

        a = 16'd0;
        b = 16'd0;
        op = 2'd1;

#100

        a = 16'b0000000000000000;
        b = 16'd1;

#100

        a = 16'd8;
        b = 16'd1;

#100

        a = 16'd8;
        b = 16'd2;

#100

        a = 16'd256;
        b = 16'd1;
#100

        a = 16'd256;
        b = 16'd2;
```

Testing 16-bit Two’s Complement for general output and carry bit: Passed

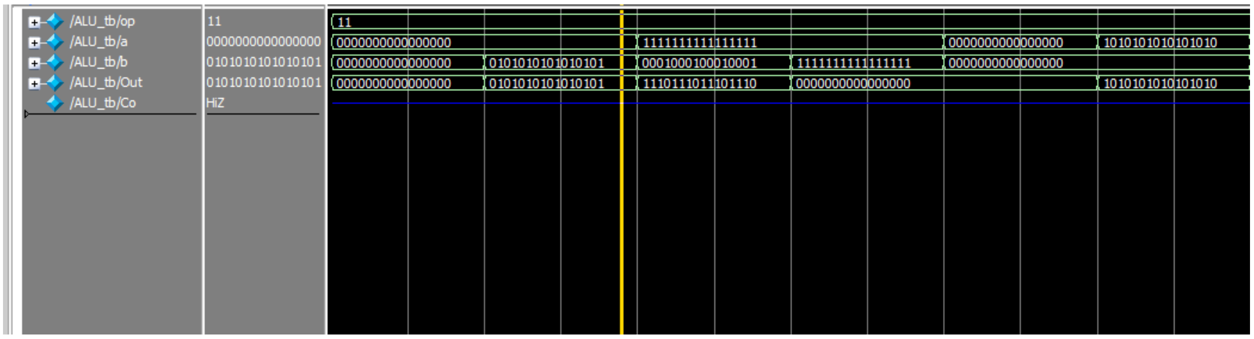


```
                 Msgs
/ALU_tb/op       10                 10
/ALU_tb/a        0000000000000000   0000000000000000  1111111111111111  0000000011111111  0101010101010101  1010101010101010
/ALU_tb/b        0000000000000000   0000000000000000
/ALU_tb/Out      0000000000000000   0000000000000000  0000000000000001  1111111100000001  1010101010101011  0101010101010110
/ALU_tb/Co       St1
```

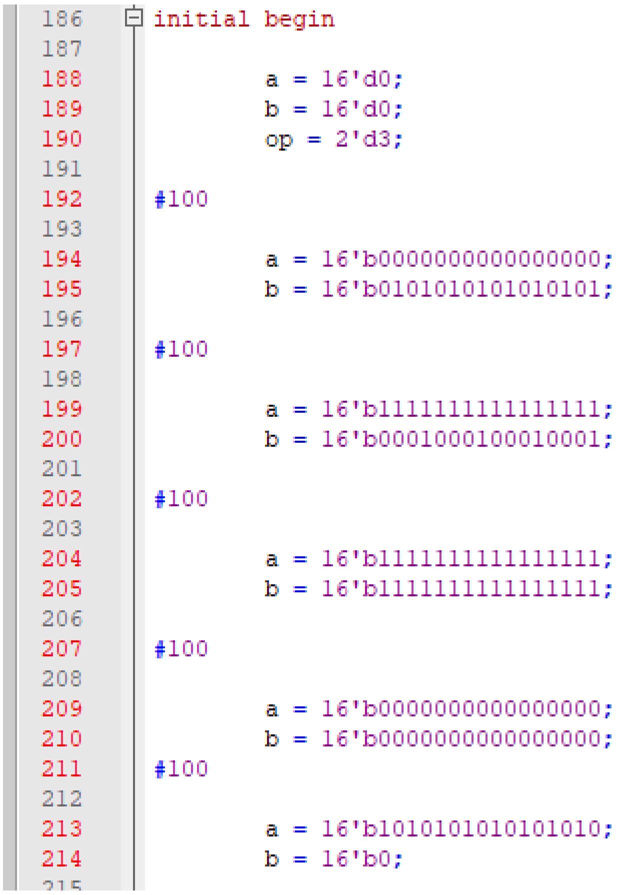


```
initial begin

        a = 16'd0;
        b = 16'd0;
        op = 2'd2;

#100

        a = 16'b0000000000000000;
        b = 16'd0;

#100

        a = 16'b1111111111111111;
        b = 16'd0;

#100

        a = 16'b0000000011111111;
        b = 16'd0;

#100

        a = 16'b0101010101010101;
        b = 16'd0;
#100

        a = 16'b1010101010101010;
        b = 16'd0;

```

Testing 16-bit XOR for general output: Passed



```
/ALU_tb/op    11                 11
/ALU_tb/a     0000000000000000   0000000000000000 | 1111111111111111 | 0000000000000000 | 1010101010101010
/ALU_tb/b     0101010101010101   0000000000000000 | 0101010101010101 | 0001000100010001 | 1111111111111111 | 0000000000000000
/ALU_tb/Out   0101010101010101   0000000000000000 | 0101010101010101 | 1110111011101110 | 0000000000000000 | 1010101010101010
/ALU_tb/Co    HiZ
```



```
initial begin

        a = 16'd0;
        b = 16'd0;
        op = 2'd3;

#100

        a = 16'b0000000000000000;
        b = 16'b0101010101010101;

#100

        a = 16'b1111111111111111;
        b = 16'b0001000100010001;

#100

        a = 16'b1111111111111111;
        b = 16'b1111111111111111;

#100

        a = 16'b0000000000000000;
        b = 16'b0000000000000000;
#100

        a = 16'b1010101010101010;
        b = 16'b0;
```

Conclusion:

This 16-bit ALU was designed to perform four different operations on two 16-bit operands *a* and *b*. Among receiving a 2-bit OP code of 00, a combination of 2x1 MUXs allow for the ALU to select the 16-bit Adder component, which comprises of four, 4-bit Adders. Each 4-bit Adder also contains four Full-Adder components. The 16-bit Adder was designed with abstraction in mind to further minimize hardware complexity, as well as improve the efficiency of the circuit by method of divide-and-conquer. Similarly, abstraction was used for the 16-bit Subtractor, 16-bit Two's Complement, and 16-Bit XOR for the same reasons. Each sub-component of the 16-bit ALU perform carry in and carry out functionality- save for the 16-Bit XOR that of which the output carry bit is don't care. All sub-components were tested for correct outputs to ensure the integrity of the 16-bit ALU's hardware logic.

Verilog Code:

```
// Author: Sean Atangan
// Professor Umer Cheema
// University Of Illinois at Chicago
// ECE 465
// This program is a design for a 16-bit ALU with
// Addition, Subtraction, Two’s Complement, and XOR functionality


// 16 Bit ALU
module ALU (op, a, b, Out, Co);
        input [15:0] a;
        input [15:0] b;
     input[1:0] op;
        output [15: 0]Out;
        output Co;
        wire [15:0]Add;
        wire [15:0]Sub;
        wire [15:0]Two;
        wire [15:0]Xor;
        wire [15:0]W;
        wire [3:0]C;
        wire cin = 1'd0;
        SixteenBitAdder A(.a(a[15:0]), .b(b[15:0]), .cin(cin) , .Cout(C[0]), .Sum(Add[15:0]));
        SixteenBitSubtractor S(.a(a[15:0]), .b(b[15:0]), .bin(cin), .Bout(C[1]), .Diff(Sub[15:0]));
        SixteenBitTwosComp T(.a(a[15:0]), .cin(cin),.Cout(C[2]), .Sum(Two[15:0]));
        SixteenBitXor X(.a(a[15:0]), .b(b[15:0]), .Out(Xor[15:0]));
                                                                          // Output Bits
        assign Out[15:0] = ({op[1],op[0]} == 2'b00) ? Add[15:0] :       // If OP code is 00, MUX
chooses the 16bitAdder
                ({op[1],op[0]} == 2'b01) ? Sub[15:0] :                  // If OP code is 01, MUX
chooses the 16bitSubtractor
                ({op[1],op[0]} == 2'b10) ? Two[15:0] :                  // If OP code is 10, MUX
chooses the 16bitTwosComp
```

```
                    ({op[1],op[0]} == 2'b11) ? Xor[15:0]: 1'bx ;                    // If OP code is 11, MUX
chooses the 16bitXOR



                                                                                        // Carry Out

          assign Co = ({op[1],op[0]} == 2'b00) ? C[0] :                 // If OP code is 00, MUX chooses the
16bitAdder

                    ({op[1],op[0]} == 2'b01) ? C[1] :                   // If OP code is 01, MUX chooses the
16bitSubtractor

                    ({op[1],op[0]} == 2'b10) ? C[2] :                                               // If OP
code is 10, MUX chooses the 16bitTwosComp

                    ({op[1],op[0]} == 2'b11) ? C[3]: 1'bx ;                       // If OP code is 11, MUX
chooses the 16bitXOR

endmodule


// Add
*********************************************************************************

module SixteenBitAdder (a, b, cin, Cout, Sum);

          input [15:0] a;

          input [15:0] b;

          input cin;

          output [15:0] Sum;

          output Cout;

          wire [15:0] Z;

          wire [3:0] W;

          FourBitAdder FA0(.a(a[3:0]), .b(b[3:0]), .cin(cin), .Cout(W[0]), .Sum(Z[3:0]));

          FourBitAdder FA1(.a(a[7:4]), .b(b[7:4]), .cin(W[0]), .Cout(W[1]), .Sum(Z[7:4]));

          FourBitAdder FA2(.a(a[11:8]), .b(b[11:8]), .cin(W[1]), .Cout(W[2]), .Sum(Z[11:8]));

          FourBitAdder FA3(.a(a[15:12]), .b(b[15:12]), .cin(W[2]), .Cout(W[3]), .Sum(Z[15:12]));

          assign Sum[15:0] = Z[15:0];

          assign Cout = W[3];

endmodule

module FourBitAdder (a, b, cin, Cout, Sum);

          input [3:0] a;
```

```
	input [3:0] b;
	input cin;
	output [3:0] Sum;
	output Cout;
	wire [3:0] Z;
	wire [3:0] W;
	FullAdder Fv0(.a(a[0]), .b(b[0]), .cin(cin), .Cout(W[0]), .Sum(Z[0]));
	FullAdder Fv1(.a(a[1]), .b(b[1]), .cin(W[0]), .Cout(W[1]), .Sum(Z[1]));
	FullAdder Fv2(.a(a[2]), .b(b[2]), .cin(W[1]), .Cout(W[2]), .Sum(Z[2]));
	FullAdder Fv3(.a(a[3]), .b(b[3]), .cin(W[2]), .Cout(W[3]), .Sum(Z[3]));
	assign Sum[3:0] = Z[3:0];
	assign Cout = W[3];
endmodule
module FullAdder(a, b, cin, Cout, Sum);
	input a;
	input b;
	input cin;
	output Sum;
	output Cout;
	assign {Cout, Sum} = a + b + cin;
endmodule

// Subtract
******************************************************************************
module SixteenBitSubtractor (a, b, bin, Bout, Diff);
	input [15:0] a;
	input [15:0] b;
	input bin;
	output [15:0] Diff;
	output Bout;
	wire [15:0] Z;
	wire [3:0] W;
```

```
        FourBitSubtractor FS0(.a(a[3:0]), .b(b[3:0]), .bin(bin), .Bout(W[0]), .Diff(Z[3:0]));
        FourBitSubtractor FS1(.a(a[7:4]), .b(b[7:4]), .bin(W[0]), .Bout(W[1]), .Diff(Z[7:4]));
        FourBitSubtractor FS2(.a(a[11:8]), .b(b[11:8]), .bin(W[1]), .Bout(W[2]), .Diff(Z[11:8]));
        FourBitSubtractor FS3(.a(a[15:12]), .b(b[15:12]), .bin(W[2]), .Bout(W[3]), .Diff(Z[15:12]));
        assign Diff[15:0] = Z[15:0];
        assign Bout = W[3];
endmodule
module FourBitSubtractor (a, b, bin, Bout, Diff);
        input [3:0] a;
        input [3:0] b;
        input bin;
        output [3:0] Diff;
        output Bout;
        wire [3:0] Z;
        wire [3:0] W;
        FullSubtractor Fs0(.a(a[0]), .b(b[0]), .bin(bin), .Bout(W[0]), .Diff(Z[0]));
        FullSubtractor Fs1(.a(a[1]), .b(b[1]), .bin(W[0]), .Bout(W[1]), .Diff(Z[1]));
        FullSubtractor Fs2(.a(a[2]), .b(b[2]), .bin(W[1]), .Bout(W[2]), .Diff(Z[2]));
        FullSubtractor Fs3(.a(a[3]), .b(b[3]), .bin(W[2]), .Bout(W[3]), .Diff(Z[3]));
        assign Diff[3:0] = Z[3:0];
        assign Bout = W[3];
endmodule
module FullSubtractor(a, b, bin, Bout, Diff);
        input  a;
        input  b;
        input bin;
        output Diff;
        output Bout;
        wire[5:0]W;
        assign W[0] = a^b;
        assign W[1] = b&bin;
        assign W[2] = bin^W[0]; // Diff
        assign W[3] = b|bin;
        assign W[4] = !a&W[3];
        assign W[5] = W[1]|W[4]; //Bout
        assign Diff = W[2];
        assign Bout = W[5];
endmodule


// Two's Complement
*****************************************************************************
module SixteenBitTwosComp (a, cin, Cout, Sum);
        input [15:0] a;
        input cin;
        output [15:0] Sum;
        output Cout;
        wire [15:0] Z;
        wire [15:0] S;
        wire C;
        SixteenBitInverter I0(.a(a[15:0]), .Out(Z[15:0]));
        SixteenBitAdder A0(.a(Z[15:0]), .b(16'b1), .cin(cin), .Cout(C), .Sum(S[15:0]));
        assign Sum[15:0] = S[15:0];
        assign Cout = C;
endmodule
module SixteenBitInverter (a, Out);
        input [15:0] a;
        output [15:0] Out;
        wire [15:0]W;
        FourBitInverter I0 (.a(a[3:0]), .Out(W[3:0]));
        FourBitInverter I1 (.a(a[7:4]), .Out(W[7:4]));
        FourBitInverter I2 (.a(a[11:8]), .Out(W[11:8]));
        FourBitInverter I3 (.a(a[15:12]), .Out(W[15:12]));
        assign Out[15:0] = W[15:0];
```

```
endmodule
module FourBitInverter(a, Out);
        input [3:0] a;
        output [3:0] Out;
        assign Out[0] = !a[0];
        assign Out[1] = !a[1];
        assign Out[2] = !a[2];
        assign Out[3] = !a[3];
endmodule


// Bitwise XOR
****************************************************************************
module SixteenBitXor (a, b, Out);
        input [15:0] a;
        input [15:0] b;
        output [15:0] Out;
        wire [15:0] Z;
        wire [3:0] W;
        FourBitXor FX0(.a(a[3:0]), .b(b[3:0]), .Out(Z[3:0]));
        FourBitXor FX1(.a(a[7:4]), .b(b[3:0]), .Out(Z[7:4]));
        FourBitXor FX2(.a(a[11:8]), .b(b[3:0]), .Out(Z[11:8]));
        FourBitXor FX3(.a(a[15:12]), .b(b[3:0]), .Out(Z[15:12]));
        assign Out[15:0] = Z[15:0];
endmodule
module FourBitXor (a, b, Out);
        input [3:0] a;
        input [3:0] b;
        output [3:0] Out;
        wire [3:0] Z;
        wire [3:0] W;
        wire N = 1;
        wire [3:0] A;
```

```
          xor(Out[0], a[0], b[0]);
          xor(Out[1], a[1], b[1]);
          xor(Out[2], a[2], b[2]);
          xor(Out[3], a[3], b[3]);
endmodule
```