University of Science and Technology
Communications & Information Engineering Program
CIE 239: Digital Design and Computer Architecture
Summer 2024

3-bit Arithmetic and Logic Unit (ALU)

Name	ID
Aya Oraby	201900967
Nada Ghafagy	202001944
Menna Khaled	202301665

Designing and implement a 4-bit Arithmetic and Logic Unit (ALU) that has inputs A & B to do the following:

Operation	Output (E)
OR	E = A V B
AND	E = A Λ B
CMP	E = B’
ADD	E = A + B
SUB	E = A – B
XOR	E = A ⊕ B
Two’s CMP	A’+1
INC	E = B + 1

To make that
All the operation will be connected to 8:1 Multiplexer and the selector choose what operation should be done

Truth table of Multiplexer 8:1

S1	S2	S3	Out
0	0	0	AB (AND)
0	0	1	A|B(OR)
0	1	0	A^B(XOR)
0	1	1	~B(CMP
1	0	0	A+B (ADD)
1	0	1	SUB(A-B)
1	1	0	ASR (A3A3A2A1)
1	1	1	INC (B+1)

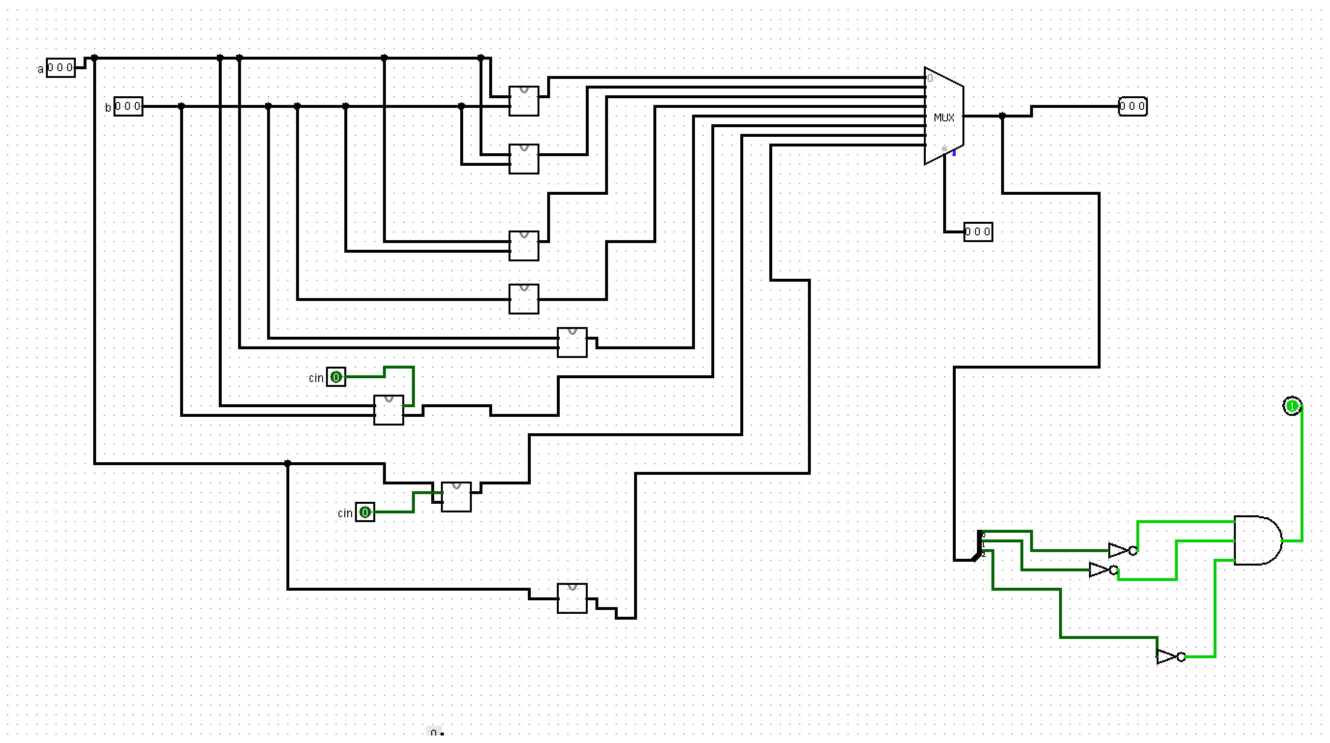
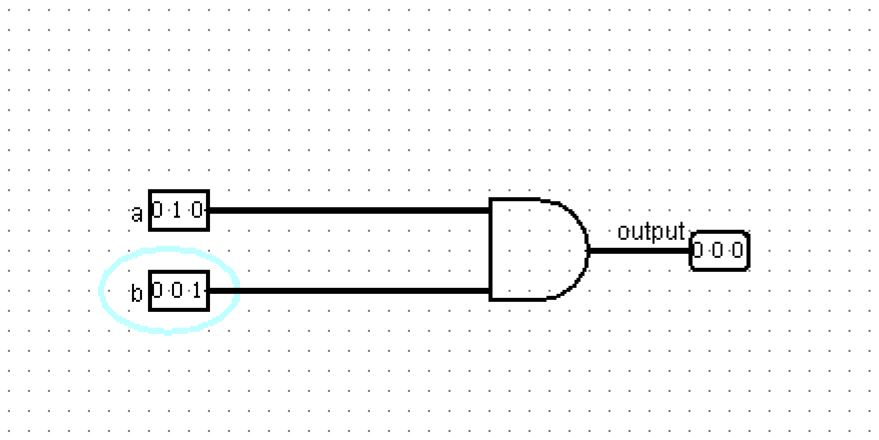


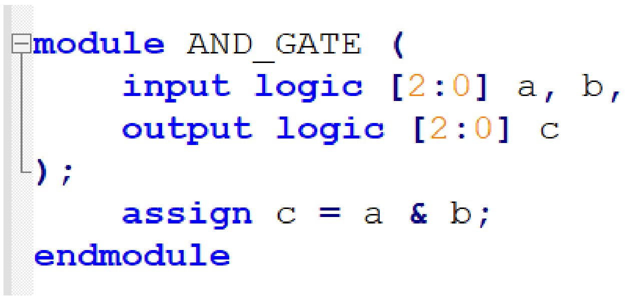
Figure 1Multiplexer 8:1

AND GATE

Both A and B are 4 bits so all the output of AND GATE is

A	B	OUT
0	0	0
0	1	0
1	0	0
1	1	1





```
module AND_GATE (
    input logic [2:0] a, b,
    output logic [2:0] c
);
    assign c = a & b;
endmodule
```

Figure 3AND Gate using System Verilog

OR GATE

Both A and B are 4 bits so all the output of OR GATE is

A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	1

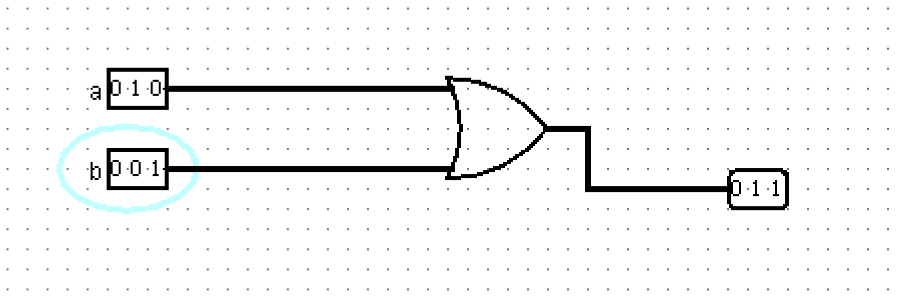
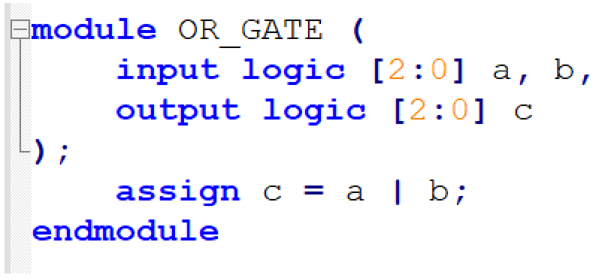


Figure 4ORGate



```
module OR_GATE (
	input logic [2:0] a, b,
	output logic [2:0] c
);
	assign c = a | b;
endmodule
```

Figure 5ORGate Using SyestemVerilog

XOR Gate

Both A and B are 4 bits so all the output of XOR GATE is

A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	1

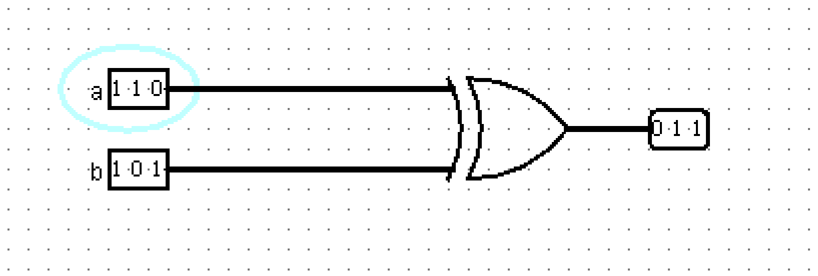
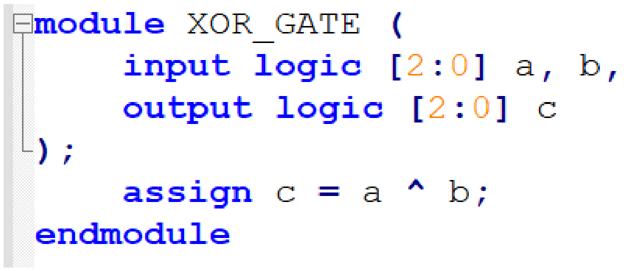


Figure 6XOR GATE



```
module XOR_GATE (
     input logic [2:0] a, b,
     output logic [2:0] c
);
     assign c = a ^ b;
endmodule
```

Figure 7XOR GATE Using System Verlig

CMP Gate

Complement Truth Table is

B	~B
1	0

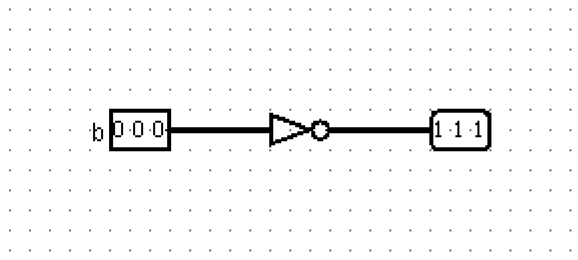
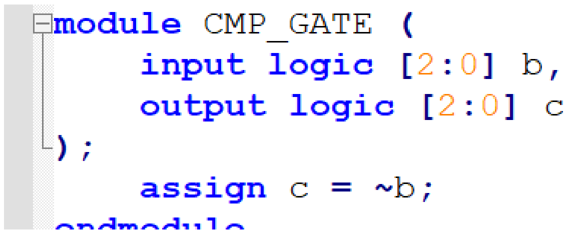


Figure 8NOT Gate



```
module CMP_GATE (
    input logic [2:0] b,
    output logic [2:0] c
);
    assign c = ~b;
```

Figure 9CMP USING System Verilog

To be able to Add four bit A,B we must have 4-bit full adder

The design I choose is two half adder that makes Full Adder
and when connect 4 Full Adder then we make 4 bit Full Adder

Half Adder

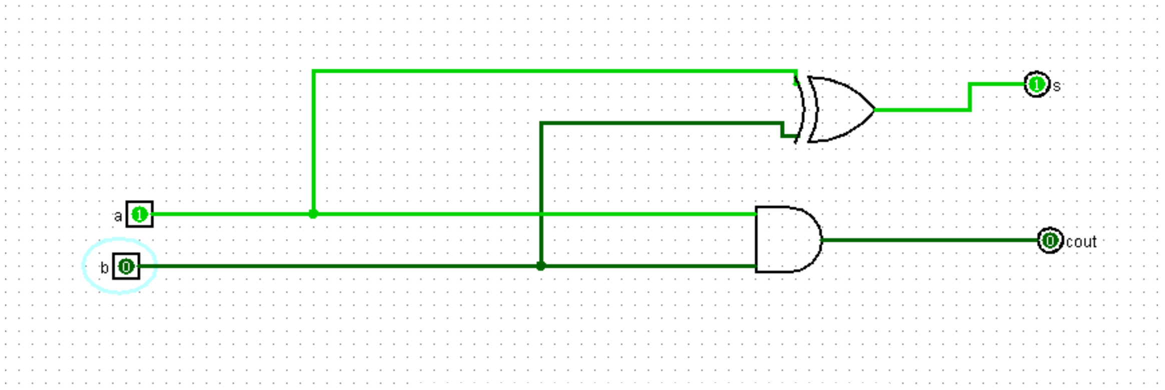
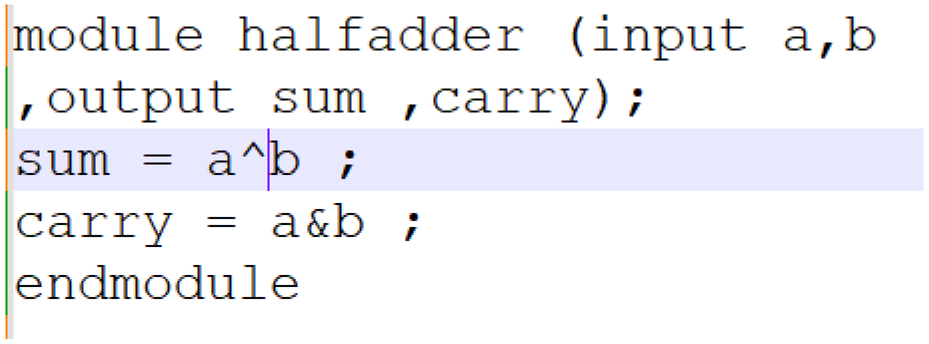
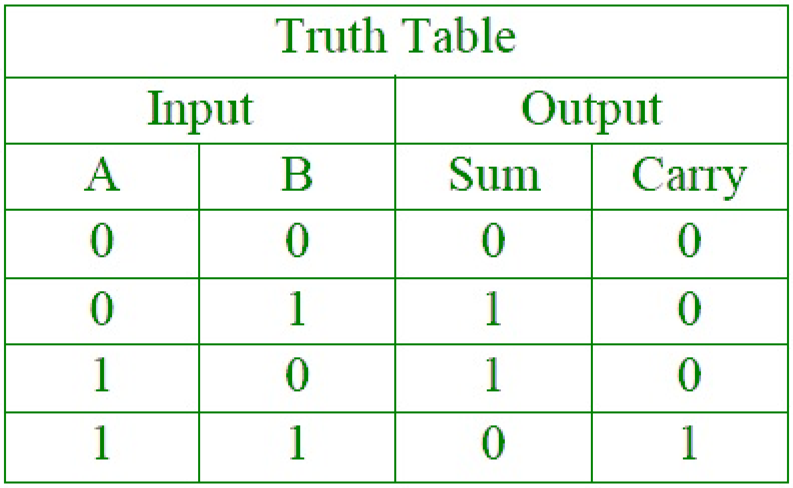


Figure 10HalfAdder



```
module halfadder (input a,b
,output sum ,carry);
sum = a^b ;
carry = a&b ;
endmodule
```

Figure 11HalfAdder Using SystemVerilog



Truth Table			
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

WE Connect two half adder to make fulladder

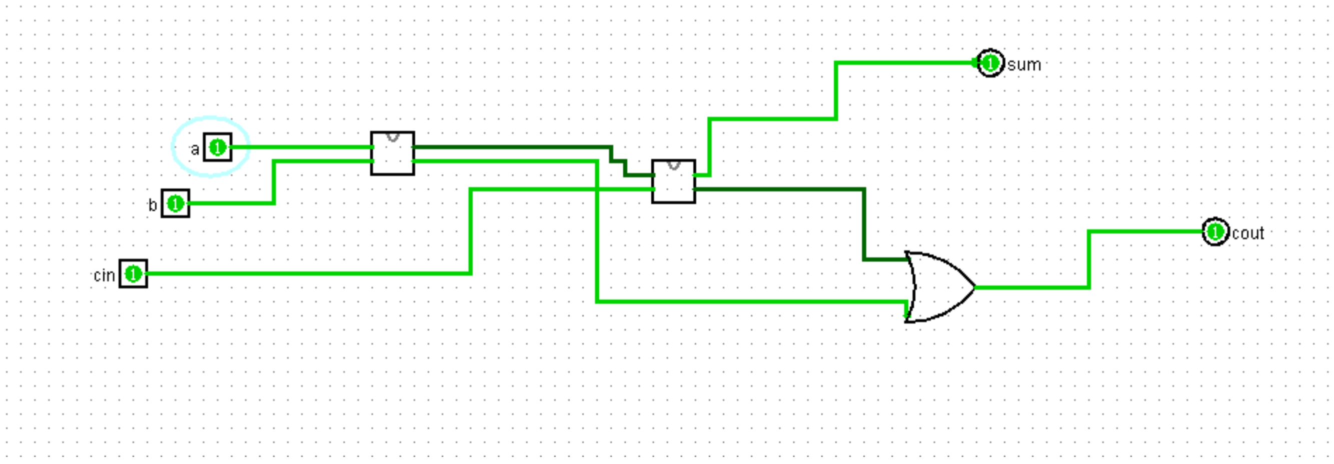
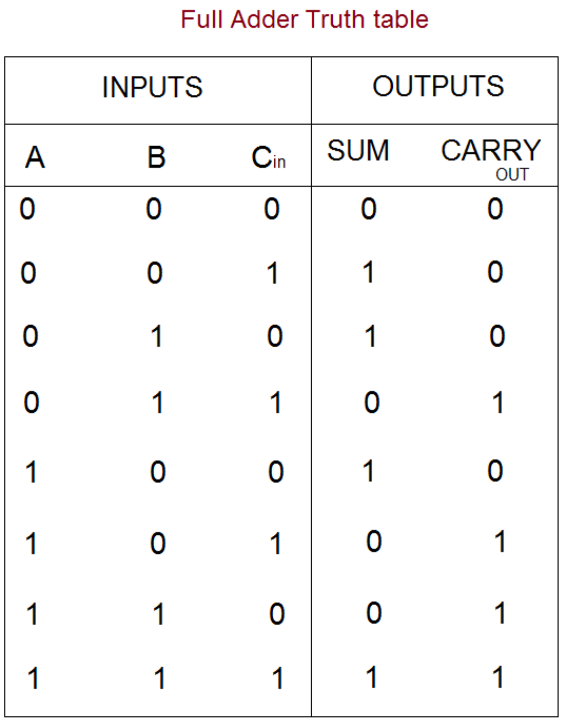
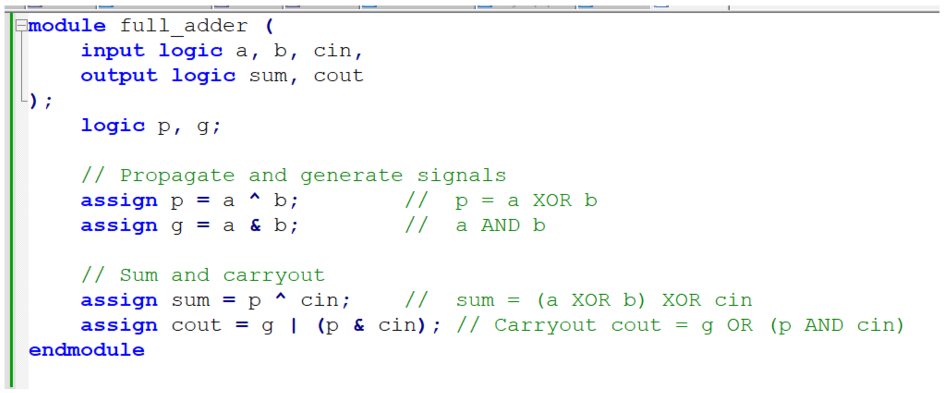


Figure 12FullAddrCircuit



Full Adder Truth table

INPUTS			OUTPUTS	
A	B	C_{in}	SUM	$CARRY_{OUT}$
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



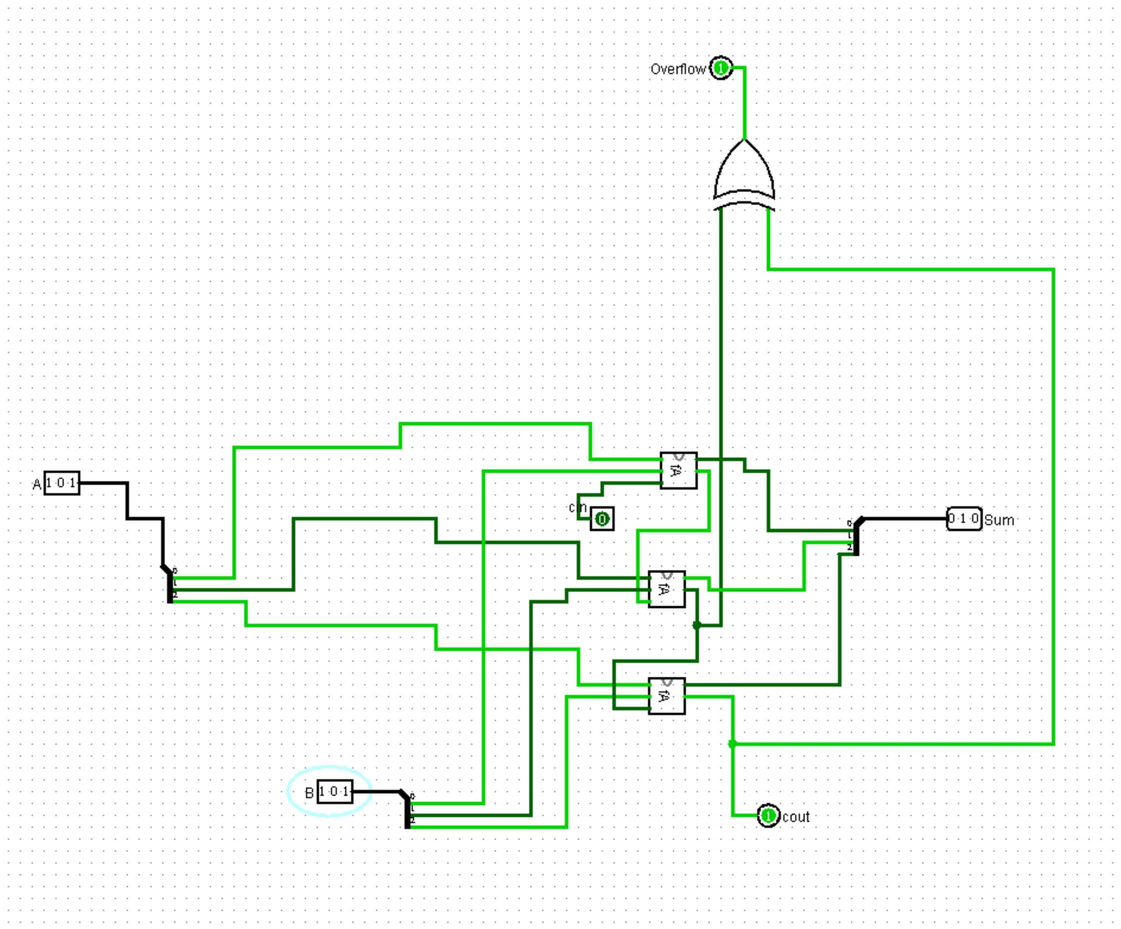
```
module full_adder (
    input logic a, b, cin,
    output logic sum, cout
);
    logic p, g;

    // Propagate and generate signals
    assign p = a ^ b;        //  p = a XOR b
    assign g = a & b;        //  a AND b

    // Sum and carryout
    assign sum = p ^ cin;    //  sum = (a XOR b) XOR cin
    assign cout = g | (p & cin); // Carryout cout = g OR (p AND cin)
endmodule
```

Figure 14 4-bitFullAdder

Weconnecct 4 full adder to make 4-bit full Adder



ADD Gate

Based on full Adder

output = A+B

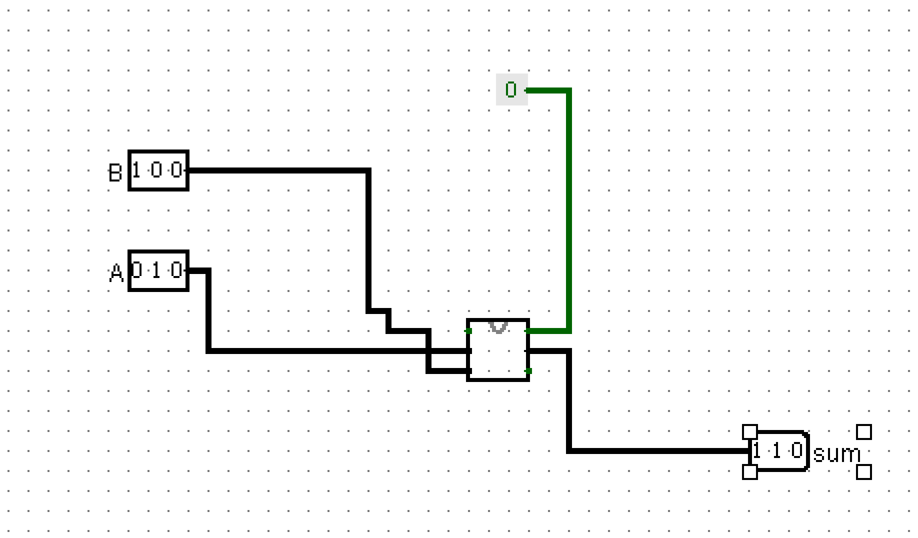
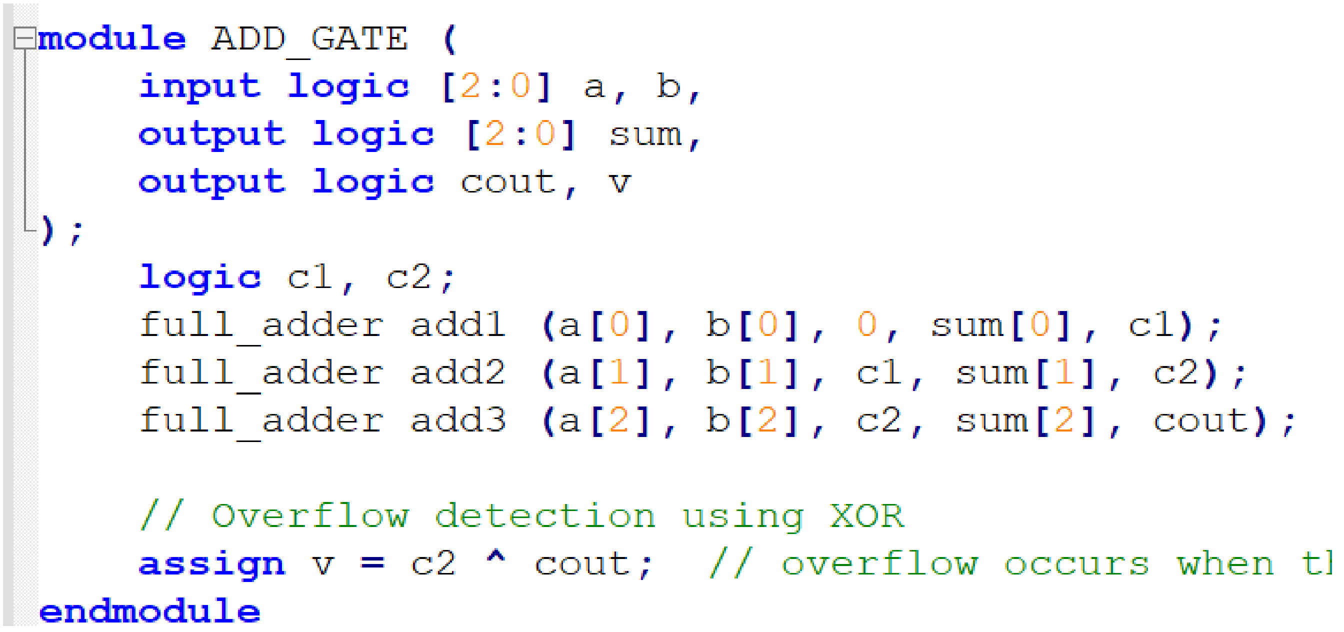


Figure 15ADD Gate

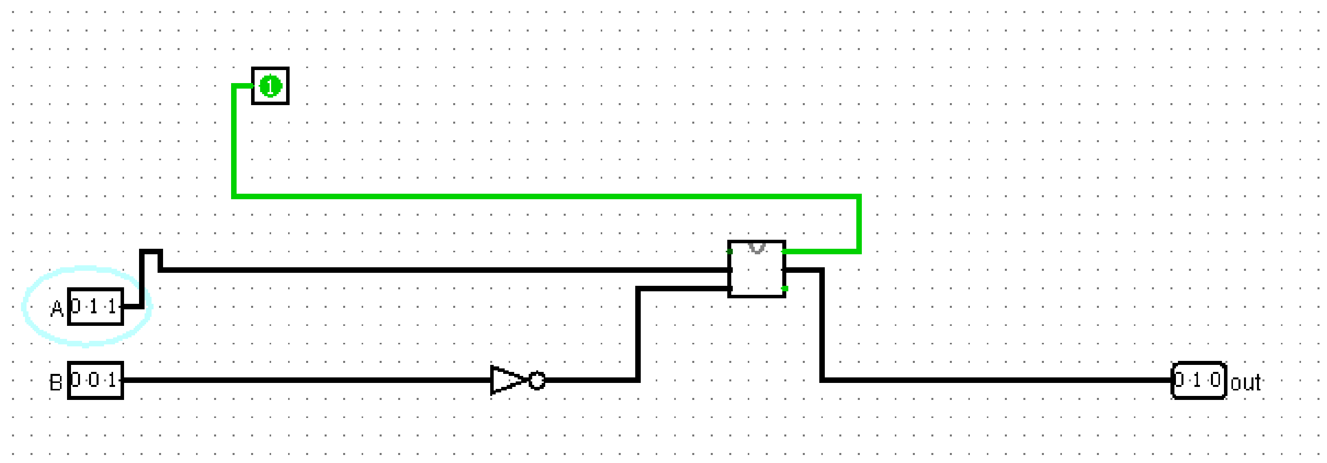


```
module ADD_GATE (
    input logic [2:0] a, b,
    output logic [2:0] sum,
    output logic cout, v
);
    logic c1, c2;
    full_adder add1 (a[0], b[0], 0, sum[0], c1);
    full_adder add2 (a[1], b[1], c1, sum[1], c2);
    full_adder add3 (a[2], b[2], c2, sum[2], cout);

    // Overflow detection using XOR
    assign v = c2 ^ cout;  // overflow occurs when tl
endmodule
```

Figure 16ADD Using System Verilog

SUB Gate



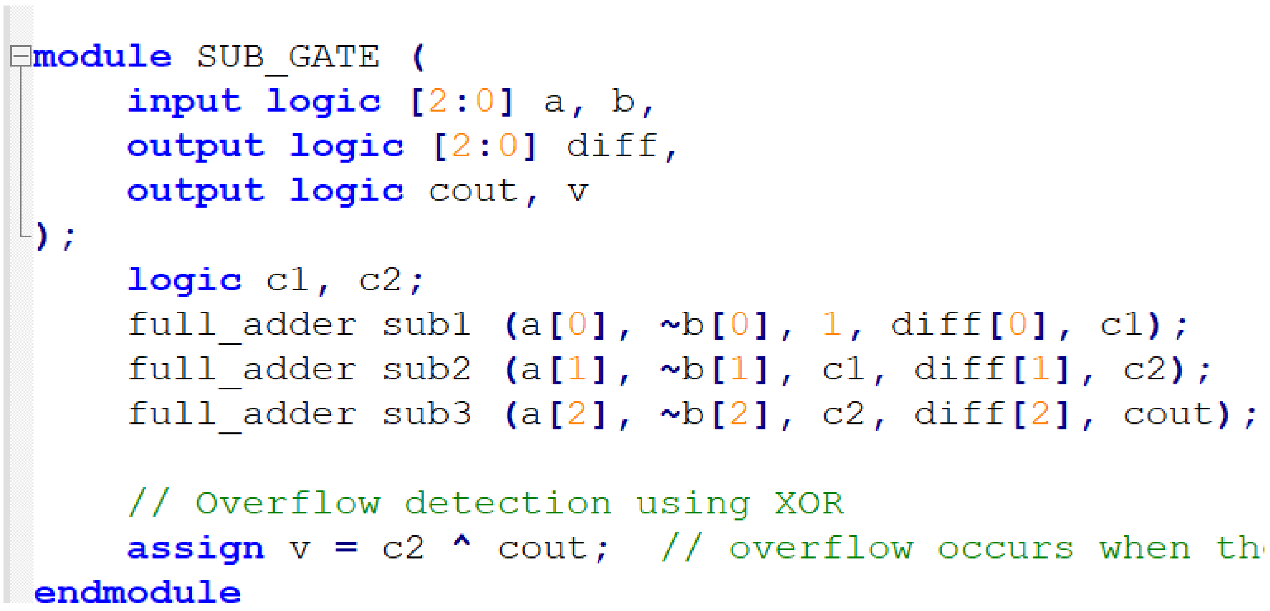
The Operation Of Subtraction is A-B

So it is can be done by   A+~B+1

The reason this works is because in two's complement representation, negating a number (finding its two's complement) involves inverting all the bits and then adding 1. This is a convenient property that allows subtraction to be performed using addition.

So

A-B =A+~B+1



```
module SUB_GATE (
    input logic [2:0] a, b,
    output logic [2:0] diff,
    output logic cout, v
);
    logic c1, c2;
    full_adder sub1 (a[0], ~b[0], 1, diff[0], c1);
    full_adder sub2 (a[1], ~b[1], c1, diff[1], c2);
    full_adder sub3 (a[2], ~b[2], c2, diff[2], cout);

    // Overflow detection using XOR
    assign v = c2 ^ cout;  // overflow occurs when th
endmodule
```

Figure 17SUB using System Verilog

INCREMENT OPERTION

B+1 so A is constant with zero value using 4-bitfull-adder

0+B+1

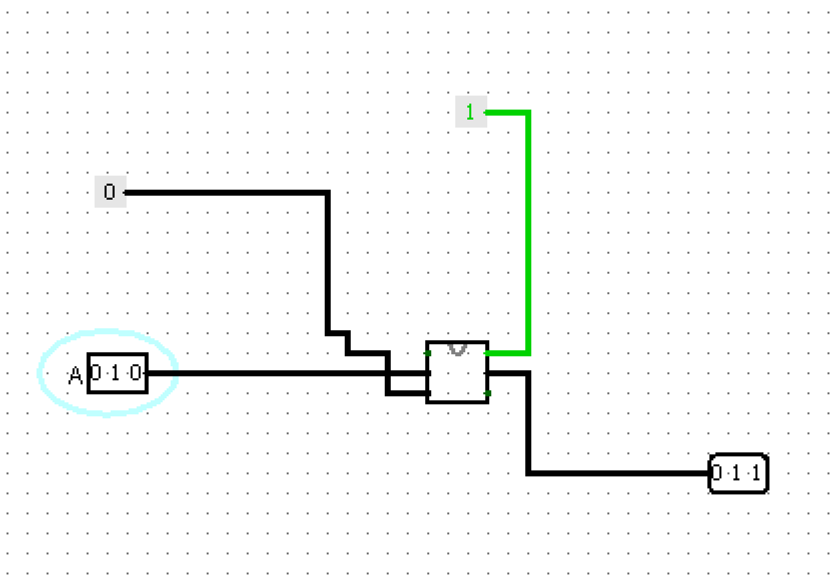
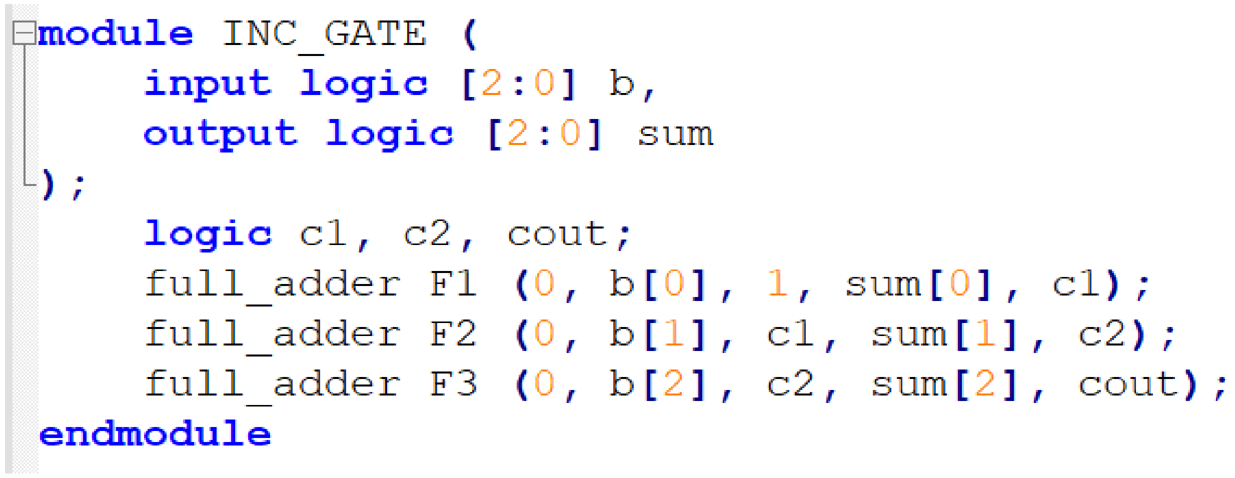


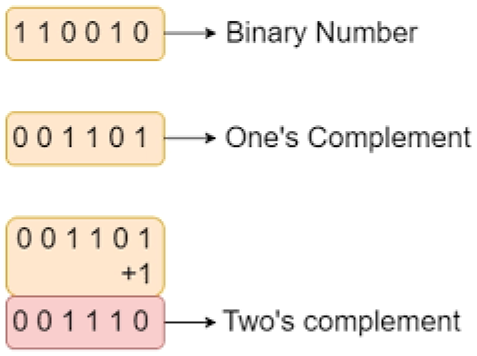
Figure 18Increment Operation



```
module INC_GATE (
     input logic [2:0] b,
     output logic [2:0] sum
);
     logic c1, c2, cout;
     full_adder F1 (0, b[0], 1, sum[0], c1);
     full_adder F2 (0, b[1], c1, sum[1], c2);
     full_adder F3 (0, b[2], c2, sum[2], cout);
endmodule
```

Figure 19Increment Uisng System Verrlo

Two's complement
Invert the number then adding one



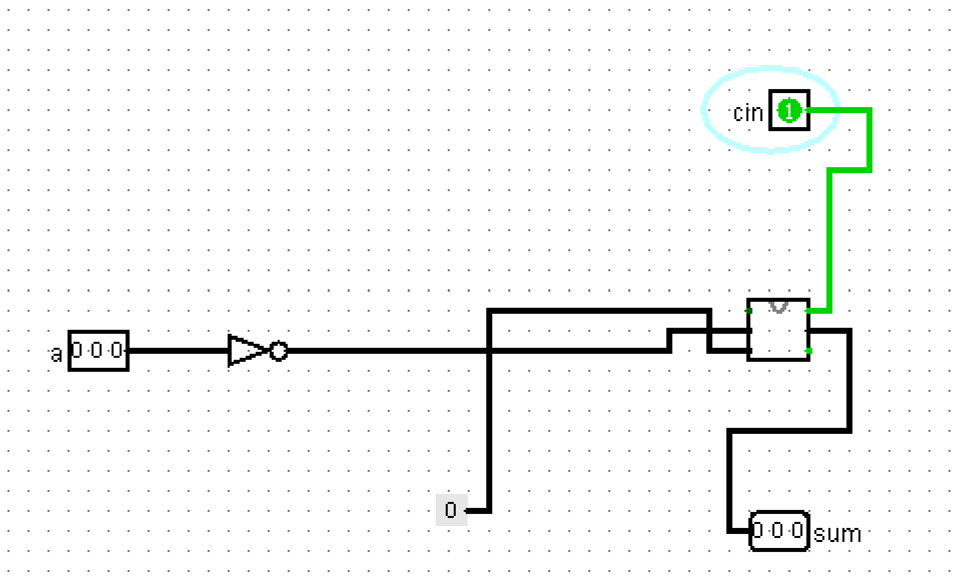
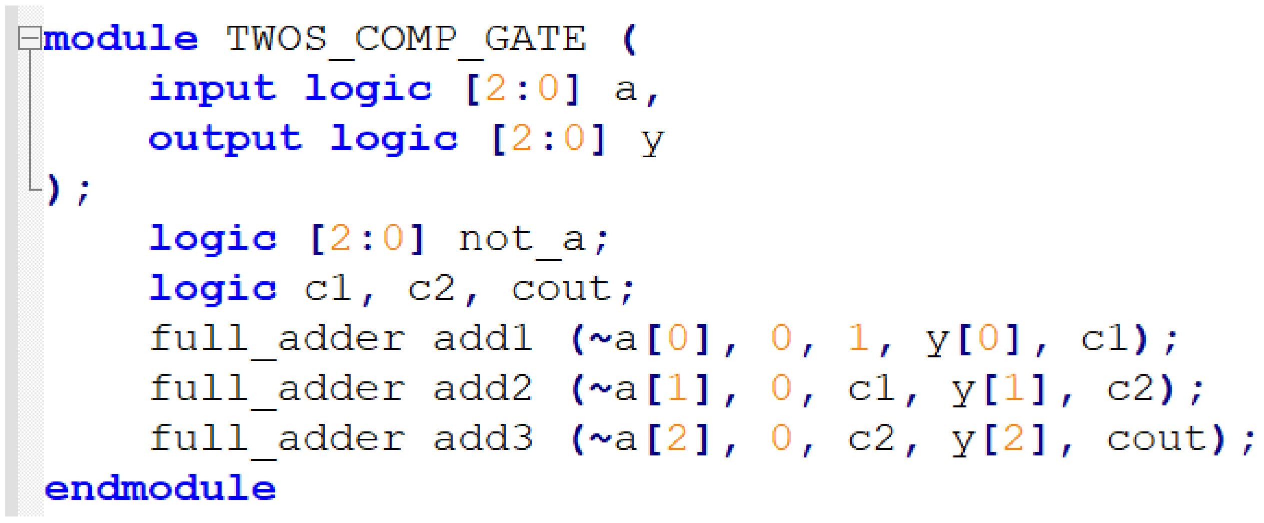


Figure 20Two's compelement Operation

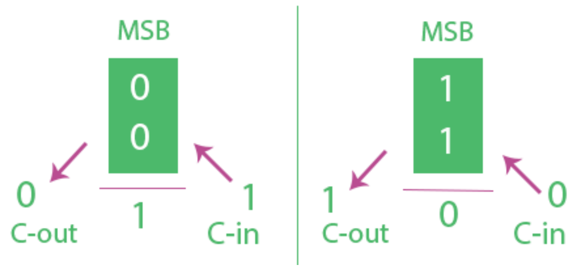


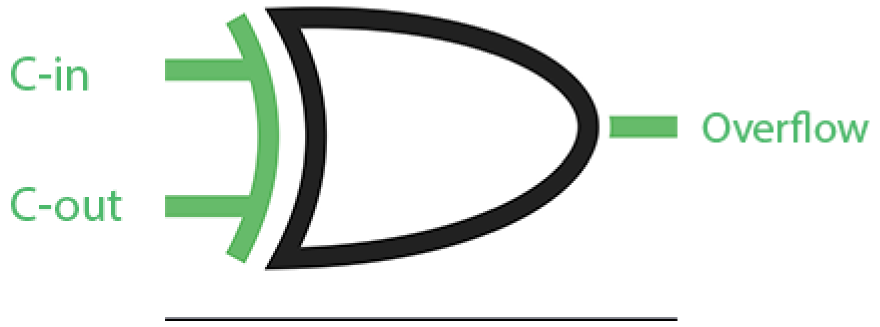
```
module TWOS_COMP_GATE (
     input logic [2:0] a,
     output logic [2:0] y
);
     logic [2:0] not_a;
     logic c1, c2, cout;
     full_adder add1 (~a[0], 0, 1, y[0], c1);
     full_adder add2 (~a[1], 0, c1, y[1], c2);
     full_adder add3 (~a[2], 0, c2, y[2], cout);
endmodule
```

Figure 21Two's compement using systemverilog

OverFlow

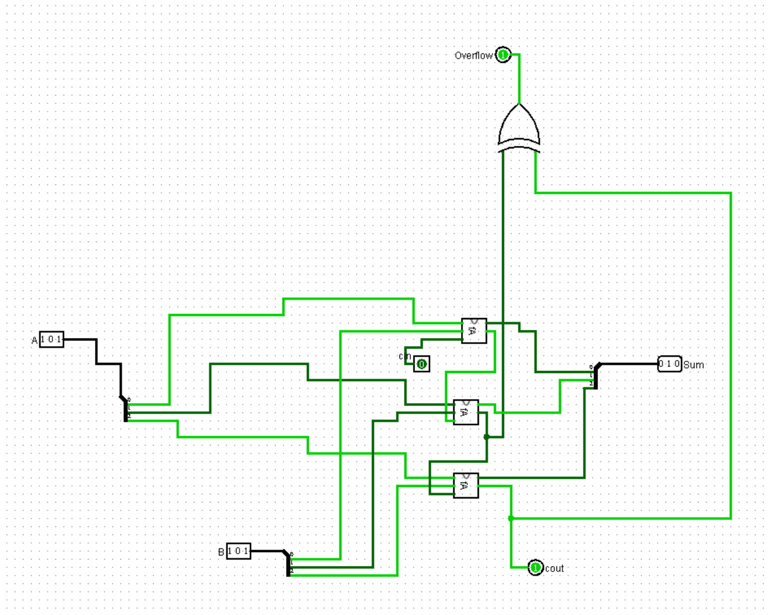
Overflow Occurs when C-in ≠C-out. The above expression for overflow can be explained below Analysis.



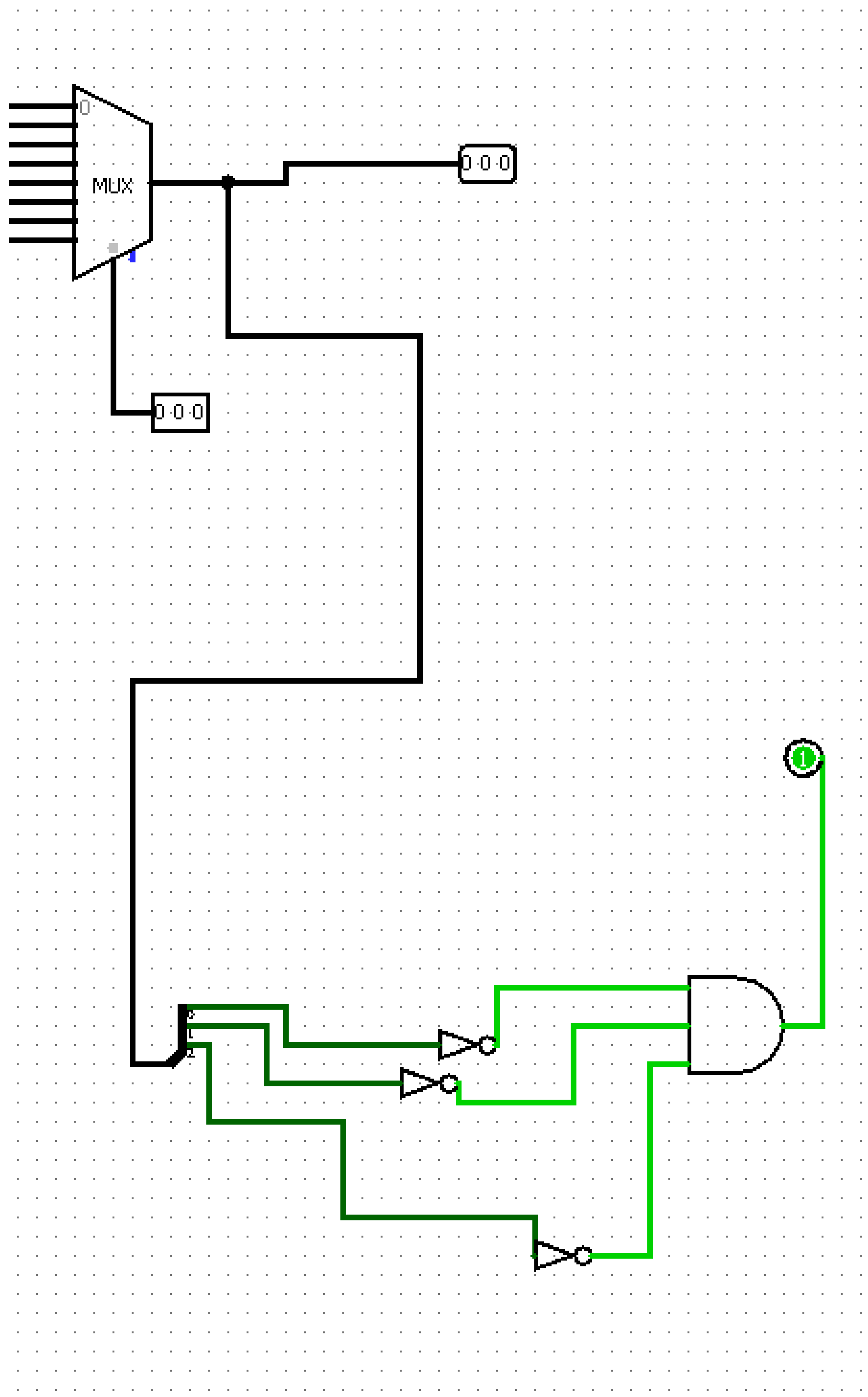


To check the overflow

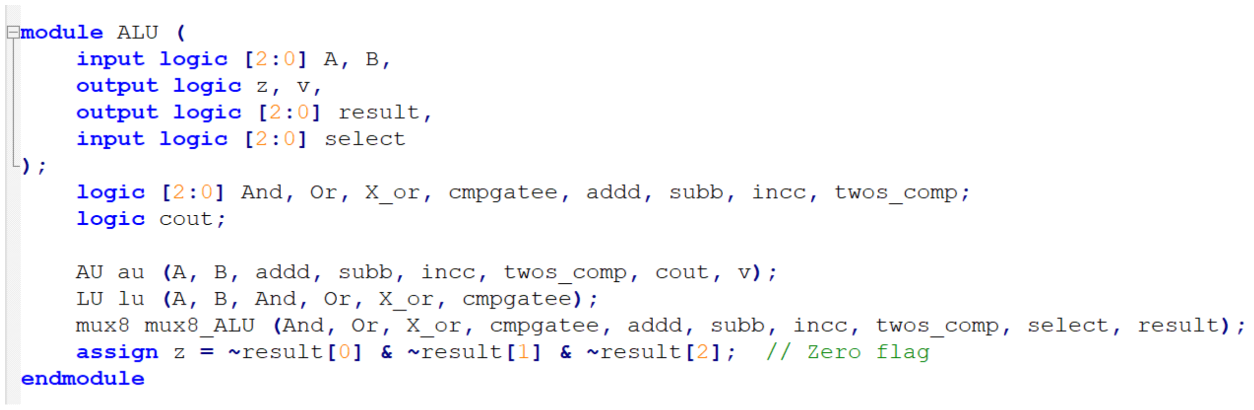
overflow occurs when the carry into the MSB and the carry out differ



Output bit Z, where Z = 1 if E = 0.

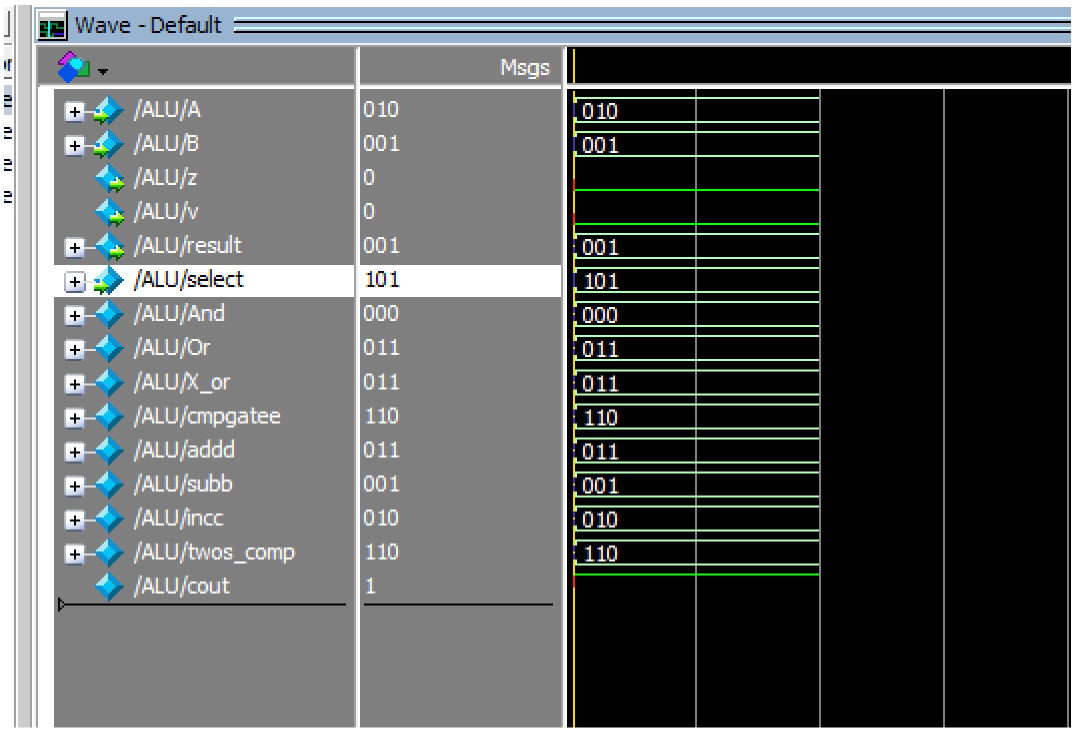

MUX
0
0·0·0
0·0·0
1

We take the output of multiplexer and connect it with inverter gate
so when the output is 000 so the output of And gate after not gate is zero so
z =



```
module ALU (
    input logic [2:0] A, B,
    output logic z, v,
    output logic [2:0] result,
    input logic [2:0] select
);
    logic [2:0] And, Or, X_or, cmpgatee, addd, subb, incc, twos_comp;
    logic cout;

    AU au (A, B, addd, subb, incc, twos_comp, cout, v);
    LU lu (A, B, And, Or, X_or, cmpgatee);
    mux8 mux8_ALU (And, Or, X_or, cmpgatee, addd, subb, incc, twos_comp, select, result);
    assign z = ~result[0] & ~result[1] & ~result[2];  // Zero flag
endmodule
```



Wave - Default

	Msgs
/ALU/A	010
/ALU/B	001
/ALU/z	0
/ALU/v	0
/ALU/result	001
/ALU/select	101
/ALU/And	000
/ALU/Or	011
/ALU/X_or	011
/ALU/cmpgatee	110
/ALU/addd	011
/ALU/subb	001
/ALU/incc	010
/ALU/twos_comp	110
/ALU/cout	1

Figure 22 suimulation of ALU

The result is 001 as the gate is Subtraction 101
so
010 – 001 = 001