

Amrita Vishwa Vidyapeetham
Amrita School of Engineering, Coimbatore
Department of Computer Science and Engineering
Topic: Basic ALU Implementation using iVerilog

Sub Code: 19CS211

Sub Title: COA

Roll No: CB.EN.U4CSE19453

Name: R.ABHINAV

Lab Evaluation No: 3

Date: 01-03-2021

1. Design of 1 bit ALU with the following operations

iVerilog Code:

```

module ALU(A,B,Operation,out);
input [3:0]A, B;
input [2:0] Operation;
output reg [3:0] out;
always@(*)
begin
    case(Operation)
        3'b000: out= A+B;
        3'b001: out= A-B;
        3'b010: out= A & B;
        3'b011: out= A | B;
        3'b100: out= ~A;
        3'b101: out= ~(A & B);
        3'b110: out= ~(A | B);
        3'b111: out= 0;
        default: out=0;
    endcase
end
endmodule

```

TestBench :

```

module ALU_tb;
wire t_out;
reg t_x, t_y, t_z, t_x, t_y;
ALU_m my_gate( .a(t_x), .b(t_y), .x(t_x), .y(t_y), .z(t_z), .out(t_out) );

```

initial

begin

```
$monitor("X:%b Y:%b Z:%b | A:%b B:%b | Output:%b \n", t_x, t_y, t_z, t_x,  
t_y, t_out );
```

```
t_a = 1'b0;
```

```
t_b = 1'b0;
```

```
t_x = 1'b0;
```

```
t_y = 1'b0;
```

```
t_z = 1'b0;
```

```
#5
```

```
t_a = 1'b0;
```

```
t_b = 1'b0;
```

```
t_x = 1'b0;
```

```
t_y = 1'b0;
```

```
t_z = 1'b1;
```

```
#5
```

```
t_a = 1'b0;
```

```
t_b = 1'b0;
```

```
t_x = 1'b0;
```

```
t_y = 1'b1;
```

```
t_z = 1'b0;
```

```
#5
```

```
t_a = 1'b0;
```

t_b = 1'b0;

t_x = 1'b0;

t_y = 1'b1;

t_z = 1'b1;

#5

t_a = 1'b0;

t_b = 1'b0;

t_x = 1'b1;

t_y = 1'b0;

t_z = 1'b0;

#5

t_a = 1'b0;

t_b = 1'b0;

t_x = 1'b1;

t_y = 1'b0;

t_z = 1'b1;

#5

t_a = 1'b1;

t_b = 1'b1;

t_x = 1'b1;

t_y = 1'b1;

t_z = 1'b0;

#5

t_a = 1'b1;

t_b = 1'b1;

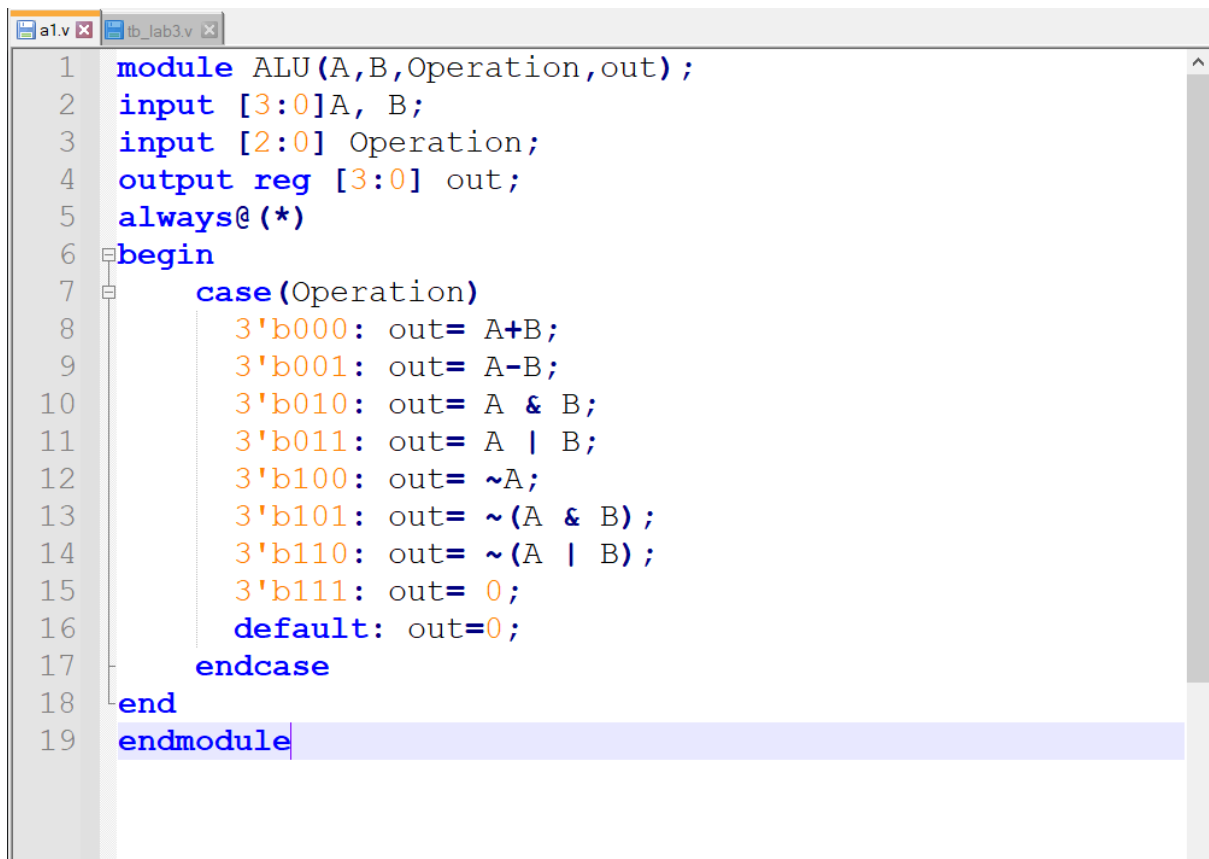
t_x = 1'b1;

t_y = 1'b1;

t_z = 1'b1;

end

endmodule



The screenshot shows a Verilog code editor with two tabs: 'a1.v' and 'tb_lab3.v'. The code in 'a1.v' defines an ALU module. It has three inputs: a 4-bit register 'A', a 4-bit register 'B', and a 3-bit 'Operation' signal. The output is a 4-bit register 'out'. The module is implemented with an 'always' block triggered by any change in inputs. Inside the 'always' block, a 'case' statement handles eight different operations based on the 'Operation' input. The operations are: addition (000), subtraction (001), bitwise AND (010), bitwise OR (011), bitwise NOT of A (100), bitwise NOT of (A AND B) (101), bitwise NOT of (A OR B) (110), and setting the output to 0 (111). A default case sets the output to 0. The code is numbered from 1 to 19.

```
1 module ALU(A,B,Operation,out);
2   input [3:0]A, B;
3   input [2:0] Operation;
4   output reg [3:0] out;
5   always@(*)
6   begin
7     case(Operation)
8       3'b000: out= A+B;
9       3'b001: out= A-B;
10      3'b010: out= A & B;
11      3'b011: out= A | B;
12      3'b100: out= ~A;
13      3'b101: out= ~(A & B);
14      3'b110: out= ~(A | B);
15      3'b111: out= 0;
16      default: out=0;
17    endcase
18  end
19 endmodule
```