LAB 2 – Raja Aadhithan

Design – ALU:

Code:

```
module alu(input [7:0]a,b,
           input [3:0]command_in,
       input oe,
       output [15:0]d_out);

   parameter     ADD  = 4'b0000, // Add two 8 bit numbers a and b.
         INC  = 4'b0001, // Increment a by 1.
         SUB  = 4'b0010, // Subtracts b from a.
         DEC  = 4'b0011, // Decrement a by 1.
         MUL  = 4'b0100, // Multiply 4 bit numbers a and b.( note : 4 bits ? )
         DIV  = 4'b0101, // Divide a by b.
         SHL  = 4'b0110, // Shift a to left side by 1 bit.
         SHR  = 4'b0111, // Shift a to right by 1 bit.
         AND  = 4'b1000, // Logical AND operation
         OR   = 4'b1001, // Logical OR operation
         INV  = 4'b1010, // Logical Negation
         NAND = 4'b1011, // Bitwise NAND
         NOR  = 4'b1100, // Bitwise NOR
         XOR  = 4'b1101, // Bitwise XOR
         XNOR = 4'b1110, // Bitwise XNOR
         BUF  = 4'b1111; // BUF

   //Internal variable used during ALU operation
   reg  [15:0]out;


   /*Step1 : Write down the functionality of ALU based on the commands given above.
            *Use arithmetic and logical operators* */
   always@(command_in)
      begin
     case(command_in)
            ADD : out <= a+b;
            INC : out <= a+1;
            SUB : out <= b-a;
            DEC : out <= a-1;
            MUL : out <= a*b;
            DIV : out <= a/b;
            SHL : out <= a<<1;
            SHR : out <= a>>1;
            AND : out <= a&&b;
            OR  : out <= a||b;
            INV : out <= !a;
            NAND: out <= ~(a&b);
            NOR : out <= ~(a|b);
            XOR : out <= a^b;
            XNOR: out <= ~(a^b);
            BUF : out <= a;

     endcase
      end

   //Understand the tri-state logic for actual output
   assign d_out = (oe) ? out : 16'hzzzz;

endmodule
```

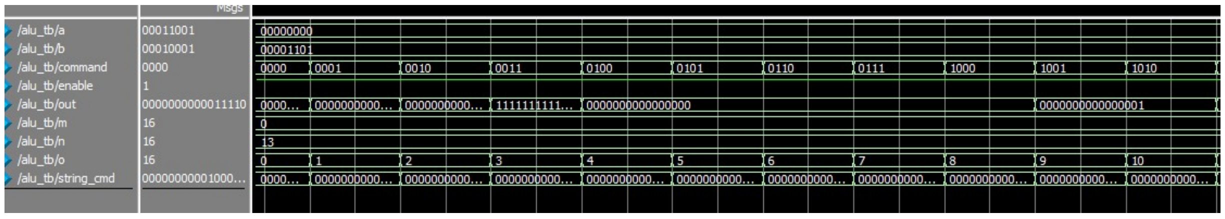
Test bench:

```
module alu_tb();

   //Testbench global variables
   reg [7:0]a,b;
   reg [3:0]command;
   reg enable;
   wire [15:0]out;

   //Variables for iteration of the loops
   integer m,n,o;

   //Parameter constants used for displaying the strings during operation
   parameter ADD  = 4'b0000, // Add two 8 bit numbers a and b.
        INC  = 4'b0001, // Increment a by 1.
        SUB  = 4'b0010, // Subtracts b from a.
        DEC  = 4'b0011, // Decrement a by 1.
        MUL  = 4'b0100, // Multiply 4 bit numbers a and b.
        DIV  = 4'b0101, // Divide a by b.
        SHL  = 4'b0110, // Shift a to left side by 1 bit.
        SHR  = 4'b0111, // Shift a to right by 1 bit.
        AND  = 4'b1000, // Logical AND operation
        OR   = 4'b1001, // Logical OR operation
        INV  = 4'b1010, // Logical Negation
        NAND = 4'b1011, // Bitwise NAND
        NOR  = 4'b1100, // Bitwise NOR
        XOR  = 4'b1101, // Bitwise XOR
        XNOR = 4'b1110, // Bitwise XNOR
        BUF  = 4'b1111; // BUF

   //Internal register for storing the string values
   reg [4*8:0]string_cmd;
```

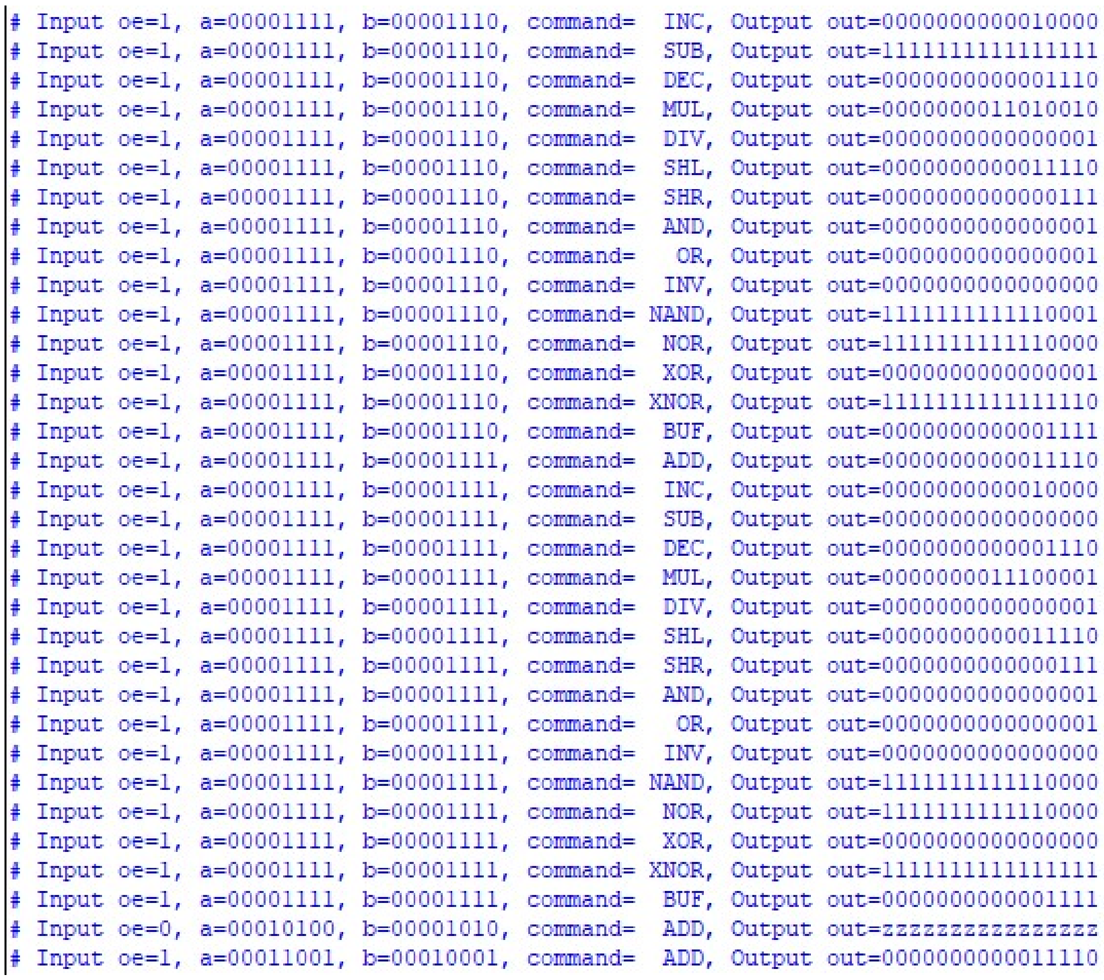
```
//Step1 : Instantiate the design ALU
alu dut(a,b,command,enable,out);

//Step2 : Write a task named "initialize" to initialize the inputs of DUT
task initialize();
   begin
      a = 8'b0;
      b = 8'b0;
      command = 1'b0;
      enable = 1'b0;
   end
endtask


//Tasks used for generating stimulus
task en_oe(input i);
   begin
      enable=i;
   end
endtask

task inputs(input [7:0]j,k);
   begin
 a=j;
 b=k;
   end
endtask

task cmd (input [3:0]l);
   begin
      command=l;
   end
endtask

task delay();
   begin
 #10;
   end
endtask

//Process to hold the string values as per the commands.
always@(command)
   begin
      case (command)
    ADD    :  string_cmd = "ADD";
    INC    :  string_cmd = "INC";
    SUB    :  string_cmd = "SUB";
    DEC    :  string_cmd = "DEC";
    MUL    :  string_cmd = "MUL";
```

```
       DIV    :  string_cmd = "DIV";
       SHL    :  string_cmd = "SHL";
       SHR    :  string_cmd = "SHR";
       INV    :  string_cmd = "INV";
       AND    :  string_cmd = "AND";
       OR     :  string_cmd = "OR";
       NAND   :  string_cmd = "NAND";
       NOR    :  string_cmd = "NOR";
       XOR    :  string_cmd = "XOR";
       XNOR   :  string_cmd = "XNOR";
       BUF    :  string_cmd = "BUF";
    endcase
      end

   //Process used for generating stimulus by calling tasks & passing values
   initial
      begin
    initialize;
    en_oe(1'b1);
    for(m=0;m<16;m=m+1)
       begin
          for(n=0;n<16;n=n+1)
             begin
                inputs(m,n);
           for(o=0;o<16;o=o+1)
         begin
            command=o;
            delay;
         end
             end
            end
         en_oe(0);
         inputs(8'd20,8'd10);
         cmd(ADD);
         delay;
         en_oe(1);
         inputs(8'd25,8'd17);
         cmd(ADD);
         delay;
         $finish;
      end

   //Process to monitor the changes in the variables
   initial
      $monitor("Input oe=%b, a=%b, b=%b, command=%s, Output out=%b",enable,a,b,string_cmd
,out);


endmodule
```

Wave:

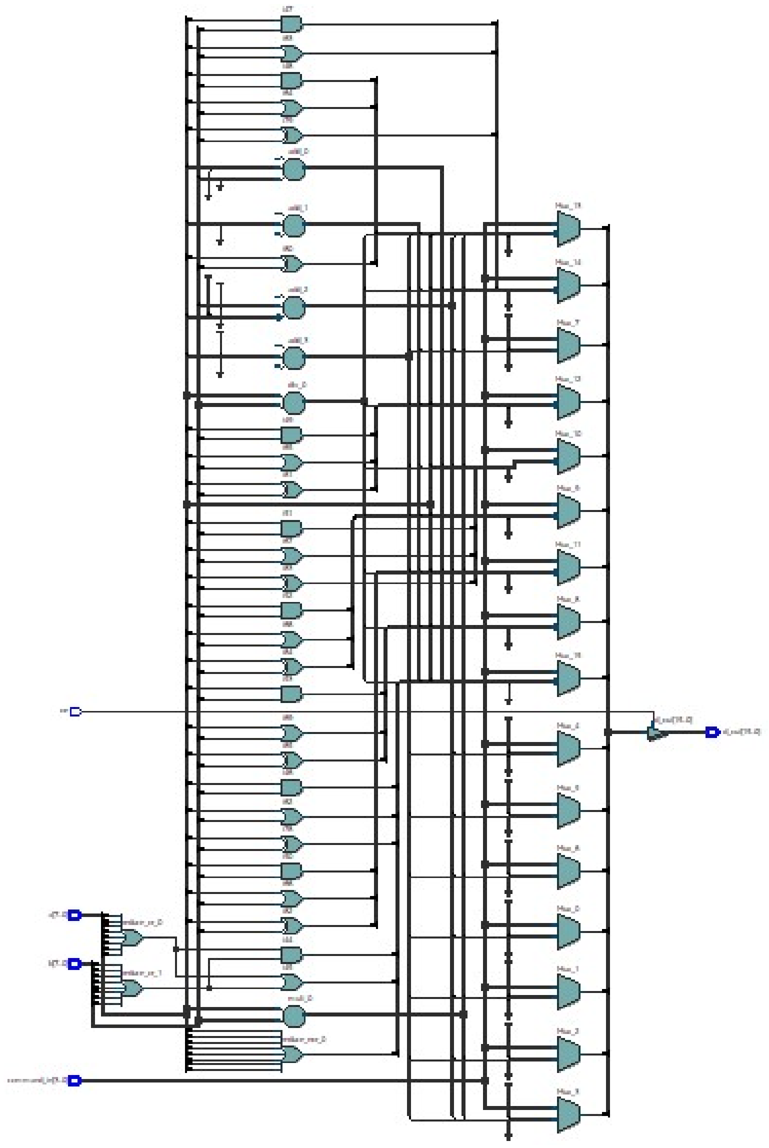


Output:



```
# Input oe=1, a=00001111, b=00001110, command=  INC, Output out=0000000000010000
# Input oe=1, a=00001111, b=00001110, command=  SUB, Output out=1111111111111111
# Input oe=1, a=00001111, b=00001110, command=  DEC, Output out=0000000000001110
# Input oe=1, a=00001111, b=00001110, command=  MUL, Output out=0000000011010010
# Input oe=1, a=00001111, b=00001110, command=  DIV, Output out=0000000000000001
# Input oe=1, a=00001111, b=00001110, command=  SHL, Output out=0000000000011110
# Input oe=1, a=00001111, b=00001110, command=  SHR, Output out=0000000000000111
# Input oe=1, a=00001111, b=00001110, command=  AND, Output out=0000000000000001
# Input oe=1, a=00001111, b=00001110, command=   OR, Output out=0000000000000001
# Input oe=1, a=00001111, b=00001110, command=  INV, Output out=0000000000000000
# Input oe=1, a=00001111, b=00001110, command= NAND, Output out=1111111111110001
# Input oe=1, a=00001111, b=00001110, command=  NOR, Output out=1111111111110000
# Input oe=1, a=00001111, b=00001110, command=  XOR, Output out=0000000000000001
# Input oe=1, a=00001111, b=00001110, command= XNOR, Output out=1111111111111110
# Input oe=1, a=00001111, b=00001110, command=  BUF, Output out=0000000000001111
# Input oe=1, a=00001111, b=00001111, command=  ADD, Output out=0000000000011110
# Input oe=1, a=00001111, b=00001111, command=  INC, Output out=0000000000010000
# Input oe=1, a=00001111, b=00001111, command=  SUB, Output out=0000000000000000
# Input oe=1, a=00001111, b=00001111, command=  DEC, Output out=0000000000001110
# Input oe=1, a=00001111, b=00001111, command=  MUL, Output out=0000000011100001
# Input oe=1, a=00001111, b=00001111, command=  DIV, Output out=0000000000000001
# Input oe=1, a=00001111, b=00001111, command=  SHL, Output out=0000000000011110
# Input oe=1, a=00001111, b=00001111, command=  SHR, Output out=0000000000000111
# Input oe=1, a=00001111, b=00001111, command=  AND, Output out=0000000000000001
# Input oe=1, a=00001111, b=00001111, command=   OR, Output out=0000000000000001
# Input oe=1, a=00001111, b=00001111, command=  INV, Output out=0000000000000000
# Input oe=1, a=00001111, b=00001111, command= NAND, Output out=1111111111110000
# Input oe=1, a=00001111, b=00001111, command=  NOR, Output out=1111111111110000
# Input oe=1, a=00001111, b=00001111, command=  XOR, Output out=0000000000000000
# Input oe=1, a=00001111, b=00001111, command= XNOR, Output out=1111111111111111
# Input oe=1, a=00001111, b=00001111, command=  BUF, Output out=0000000000001111
# Input oe=0, a=00010100, b=00001010, command=  ADD, Output out=zzzzzzzzzzzzzzzz
# Input oe=1, a=00011001, b=00010001, command=  ADD, Output out=0000000000011110
```

RTL:


add_0
add_1
add_2
add_3
mult_0
Mux_15
Mux_14
Mux_7
Mux_13
Mux_10
Mux_9
Mux_11
Mux_8
Mux_4
Mux_5
Mux_6
Mux_0
Mux_1
Mux_2
Mux_3
en
a[7:0]
b[7:0]
command_in[3:0]
d_out[15:0]