**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

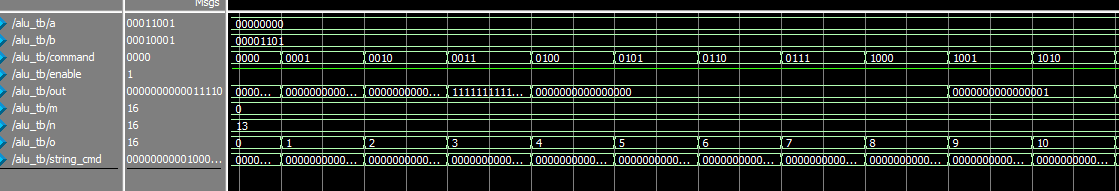
   //Process to monitor the changes in the variables

   initial

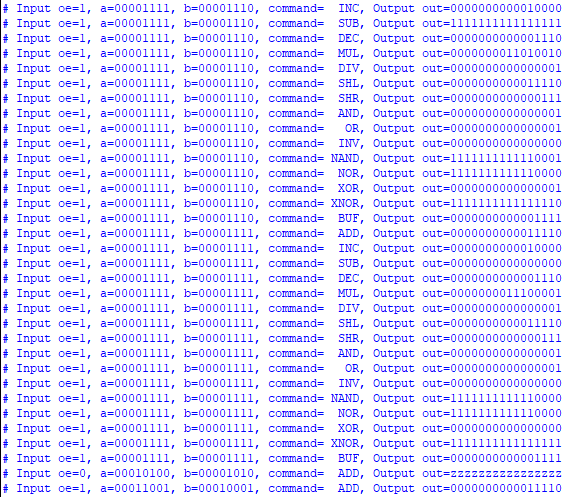
      $monitor("Input oe=%b, a=%b, b=%b, command=%s, Output out=%b",enable,a,b,string\_cmd,out);

endmodule

Wave:



Output:



RTL:

