GITAM

(Deemed to be University)

Hyderabad Campus

VLSI Design Practice

Laboratory Record

Contents

Name : Karthik M.B
Roll : 2210416132
Branch : ECE A1
Year/Sem : IV / VII

Experiment 1: Verilog code for an ALU

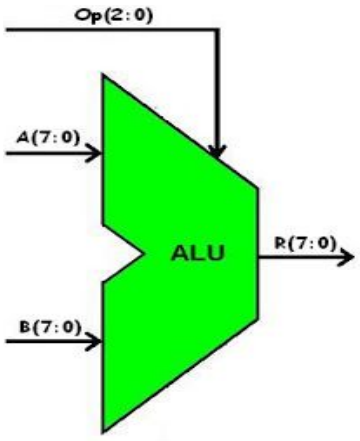
Aim: To design and simulate verilog 8-bit ALU using Xilinx software

Apparatus: Xilinx software, Computer

Theory: ALU(Arithmetic Logic Unit) is a digital circuit which does arithmetic and logical operations. Its a basic block in any processor.
ALU is capable of doing the following operations:

ALU Operation	Description
Add Signed	R = A + B : Treating A, B, and R as signed two's complement integers.
Subtract Signed	R = A - B : Treating A, B, and R as signed two's complement integers.
Bitwise AND	R(i) = A(i) AND B(i).
Bitwise NOR	R(i) = A(i) NOR B(i).
Bitwise OR	R(i) = A(i) OR B(i).
Bitwise NAND	R(i) = A(i) NAND B(i).
Bitwise XOR	R(i) = A(i) XOR B(i).
Biwise NOT	R(i) = NOT A(i).

Diagram:



Code:

```
//Verilog module for an ALU
module ALU(
    A,
    B,
    Op,
    R   );

    //inputs,outputs and internal variables declared here
```

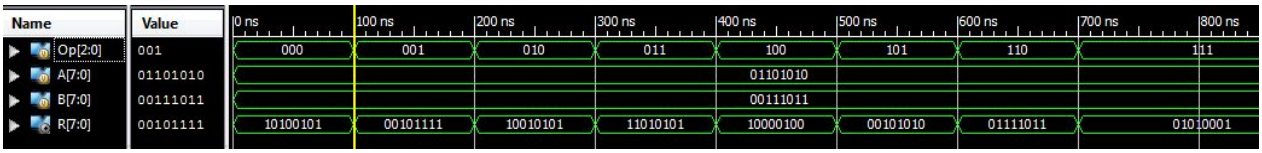
```
    input [7:0] A,B;
    input [2:0] Op;
    output [7:0] R;
    wire [7:0] Reg1,Reg2;
    reg [7:0] Reg3;

    //Assign A and B to internal variables for doing operations
    assign Reg1 = A;
    assign Reg2 = B;
    //Assign the output
    assign R = Reg3;

    //Always block with inputs in the sensitivity list.
    always @(Op or Reg1 or Reg2)
    begin
        case (Op)
            0 : Reg3 = Reg1 + Reg2;  //addition
         1 : Reg3 = Reg1 - Reg2; //subtraction
         2 : Reg3 = ~Reg1;  //NOT gate
         3 : Reg3 = ~(Reg1 & Reg2); //NAND gate
         4 : Reg3 = ~(Reg1 | Reg2); //NOR gate
         5 : Reg3 = Reg1 & Reg2;  //AND gate
         6 : Reg3 = Reg1 | Reg2;  //OR gate
         7 : Reg3 = Reg1 ^ Reg2; //XOR gate
        endcase
    end

endmodule
```

Testbench for ALU:

```
module tb_alu;

    // Inputs
    reg [7:0] A;
    reg [7:0] B;
    reg [2:0] Op;

    // Outputs
    wire [7:0] R;

    // Instantiate the Unit Under Test (UUT)
    ALU uut (
        .A(A),
        .B(B),
        .Op(Op),
        .R(R)
    );

    initial begin
        // Apply inputs.
        A = 8'b01101010;
        B = 8'b00111011;
        Op = 0; #100;
        Op = 1; #100;
        Op = 2; #100;
        Op = 3; #100;
        Op = 4; #100;
```

```
        Op = 5; #100;
        Op = 6; #100;
        Op = 7; #100;
    end

Endmodule
```

Waveforms:



Result: ALU is designed and waveforms are obtained using Xilinx software.

Experiment 2: Decoder

Aim: To design and simulate verilog Code for 3:8 Decoder using Case statement using Xilinx software

Apparatus: Xilinx software, Computer

Theory: Decoders are combinational circuits used for breaking down any combination of inputs to a set of output bits that are all set to '0' apart from one output bit. Therefore when one input changes, two output bits will change.

Let's say we have N input bits to a decoder, the number of output bits will be equal to 2^N.

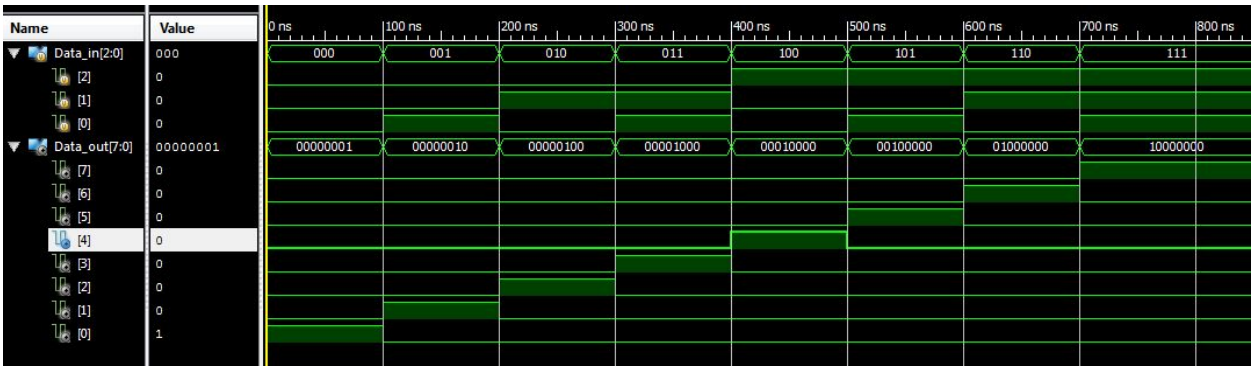
Code:

```
//declare the Verilog module - The inputs and output port names.
module decoder3to8(
    Data_in,
    Data_out
    );

    //what are the input ports and their sizes.
    input [2:0] Data_in;
    //what are the output ports and their sizes.
    output [7:0] Data_out;
    //Internal variables
    reg [7:0] Data_out;

    //Whenever there is a change in the Data_in, execute the always
block.
    always @(Data_in)
    case (Data_in)   //case statement. Check all the 8 combinations.
        3'b000 : Data_out = 8'b00000001;
        3'b001 : Data_out = 8'b00000010;
        3'b010 : Data_out = 8'b00000100;
        3'b011 : Data_out = 8'b00001000;
        3'b100 : Data_out = 8'b00010000;
        3'b101 : Data_out = 8'b00100000;
        3'b110 : Data_out = 8'b01000000;
        3'b111 : Data_out = 8'b10000000;
        //To make sure that latches are not created create a default
value for output.
        default : Data_out = 8'b00000000;
    endcase

endmodule
```

Testbench for 3:8 Decoder:

```
//This is a testbench code used for testing the 3:8 decoder module.
//Since its a testbench code we don't need to define any inputs or outputs
for the block.
module tb_decoder;

    // Declaring Inputs
    reg [2:0] Data_in;

    // Declaring Outputs
    wire [7:0] Data_out;

    // Instantiate the Unit Under Test (UUT)
    Decoder 3to8 uut (
        .Data_in(Data_in),
        .Data_out(Data_out)
    );

    initial begin
        //Apply Input and wait for 100 ns
        Data_in = 3'b000;     #100;
        Data_in = 3'b001;     #100;
        Data_in = 3'b010;     #100;
        Data_in = 3'b011;     #100;
        Data_in = 3'b100;     #100;
        Data_in = 3'b101;     #100;
        Data_in = 3'b110;     #100;
        Data_in = 3'b111;     #100;
    end

endmodule
```

Simulated Waveform:



Result: Decoder is designed and waveforms are obtained using Xilinx software.

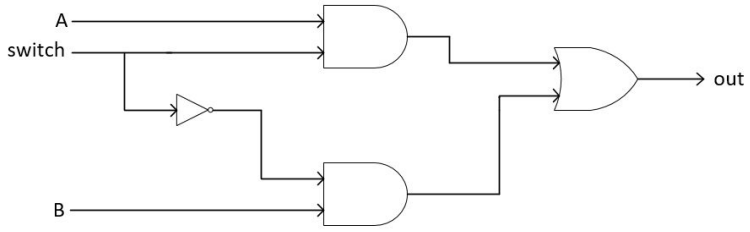
Experiment 3: Design of 4×2 Multiplexer using 2×1 mux

Aim: To design and simulate verilog code for 4×2 Multiplexer using 2×1 mux using Xilinx software

Apparatus: Xilinx software, Computer

Theory: A multiplexer is a device that can transmit several digital signals on one line by selecting certain switches.

2 x 1 Multiplexer Diagram:



Truth table for 2×1 mux is given below:

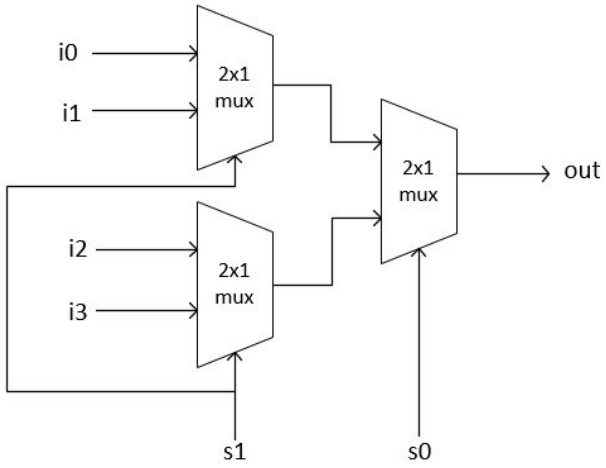
Input			Output
A	B	S	Out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

From above table, boolean expression can be written as: Out =AS+BS′

Verilog Code for 2×1 Mux:

```
module mux2x1(out,a,b,s);
input a,b,s;
wire and_1,and_2,s_c;
output out;
not (s_c,s);
and (and_1,a,s_c);
and (and_2,b,s);
or (out,and_1,and_2);
endmodule
```

4×2 mux using three 2×1 mux:



Verilog Code for 4×2 Mux

```
module mux4x2(out,i0,i1,i2,i3,s1,s0);
input i0,i1,i2,i3,s1,s0;
output out;
wire mux1,mux2;
mux2x1 mux_1(mux1,i0,i1,s1);
mux2x1 mux_2(mux2,i2,i3,s1);
mux2x1 mux_3(out,mux1,mux2,s0);
endmodule
```

This 4×2 mux is tested for selective inputs given below:

i0	i1	i2	i3	s0	s1	out
1	0	1	1	0	1	0
0	1	0	0	0	1	1
0	0	1	0	1	0	1
0	0	0	1	1	1	1
1	0	0	0	0	0	1

Test Bench code for 4×2 Mux:

```
module mux4x2_tb;
wire t_out;
reg t_a, t_b, t_c, t_d, t_s1, t_s0;
mux4x2 my_4x2_mux( .i0(t_a), .i1(t_b), .i2(t_c), .i3(t_d), .s1(t_s1),
.s0(t_s0), .out(t_out) );
initial
begin
// 1
t_a = 1'b1;
t_b = 1'b0;
t_c = 1'b1;
t_d = 1'b1;
t_s0 = 1'b0;
t_s1 = 1'b1;
#5 //2
t_a = 1'b0;
t_b = 1'b1;
t_c = 1'b0;
```

```
t_d = 1'b0;
t_s0 = 1'b0;
t_s1 = 1'b1;
#5 //3
t_a = 1'b0;
t_b = 1'b0;
t_c = 1'b1;
t_d = 1'b0;
t_s0 = 1'b1;
t_s1 = 1'b0;
#5 //4
t_a = 1'b0;
t_b = 1'b0;
t_c = 1'b0;
t_d = 1'b1;
t_s0 = 1'b1;
t_s1 = 1'b1;
#5 //5
t_a = 1'b1;
t_b = 1'b0;
t_c = 1'b0;
t_d = 1'b0;
t_s0 = 1'b0;
t_s1 = 1'b0;
end
endmodule
```

Simulated Waveform:

Result: 4 x 2 multiplexer is designed and waveforms are obtained using Xilinx software.

Experiment 4: Parity Checker

Aim: To design and simulate verilog code for parity checker using Xilinx software

Apparatus: Xilinx software, Computer

Theory: Parity refers to the evenness or oddness of the number of bits with value one within a given set of bits, and is thus determined by the value of all the bits.

Parity bit checking is used occasionally for transmitting ASCII characters, which have 7 bits, leaving the 8th bit as a parity bit.

Code:

```
module bus_parity #(
    parameter WPARIN = 8
) (
    input  wire [WPARIN-1:0] parity_in,
    output reg               parity_out
);

always @* begin : parity
    integer i;
    reg     result;

    result = 1'b0;
    for(i=0; i < WPARIN-1; i=i+1) begin
        result = result ^ parity_in[i];
    end

    parity_out = result;
end

endmodule
```

Output:

```
# parity_in = 10010110, parity_out = 0
# parity_in = 11010101, parity_out = 1
# parity_in = 10101011, parity_out = 1
# parity_in = 10000010, parity_out = 0
# parity_in = 01011000, parity_out = 1
```

Result: Parity checker is designed and waveforms are obtained using Xilinx software.

Experiment 5: Encoder

Aim: To design and simulate verilog code for 8:3 Encoder using Xilinx software

Apparatus: Xilinx software, Computer

Theory: An encoder is a device, circuit, transducer, software program, algorithm or person that converts information from one format or code to another, for the purposes of standardization, speed or compression

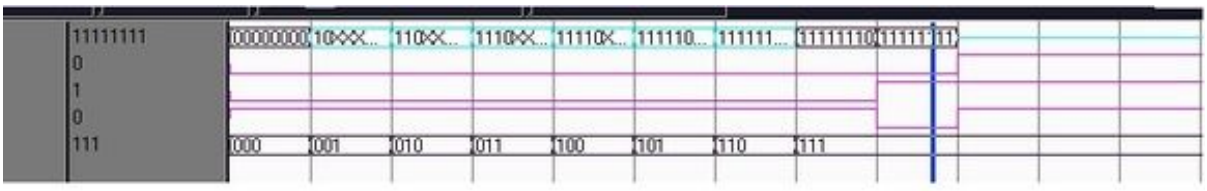
Code:

```
module encodermod(d, a, b, c);
input [0:7] d;
output a;
output b;
output c;
or(a,d[4],d[5],d[6],d[7]);
or(b,d[3],d[2],d[6],d[7]);
or(c,d[1],d[3],d[5],d[7]);
Endmodule
```

Testbench:

```
module encodert_b;
reg [0:7] d;
wire a;
wire b;
wire c;
encodermod uut (.d(d), .a(a), .b(b),.c(c) );
initial begin
#10 d=8'b10000000;
#10 d=8'b01000000;
#10 d=8'b00100000;
#10 d=8'b00010000;
#10 d=8'b00001000;
#10 d=8'b00000100;
#10 d=8'b00000010;
#10 d=8'b00000001;
#10$stop;
end
endmodule
```

Waveforms:



Result: Encoder is designed and waveforms are obtained using Xilinx software.