EE6325

VLSI Design

Project #1 Arbitrary Digital Design

Arithmetic Logic Unit with Speed boost

Due Date: September 06, 2018

Submitted By,

Akshay Nandkumar Patil (anp170330)

Akash Anil Tadmare (aat171030)

Submission Date: September 06, 2018

Arithmetic Logic Unit:

Arithmetic and Logic Unit (ALU) is the heart of every computing system. It performs all arithmetic viz. addition, subtraction, division, multiplication and logical operations viz. AND, OR, NOT, XOR etc. and stores the result in a memory called accumulator. ALU also performs unsigned binary operations i.e. shift left, shift right, compare etc.

In this project we are using two 32 bit operands and 8 operations, i.e. 4 arithmetical and 4 logical operations. A 3 bit select line is used to select a particular operation and the ALU performs only the selected operation avoiding the other unnecessary operations. The operations include addition, subtraction, division, modulus, bitwise AND, bitwise OR, bitwise XOR, bitwise XNOR. Two operands use a separate 1:8 de-multiplexer as shown in the figure 1.0 to select a particular operation, selected operation is performed in the ALU and an 8:1 multiplexer is used to select the calculated result to be stored in the accumulator. Here the accumulator is a register which stores the result.

This design is scalable and a number of inputs can be added as well as size of operating bits can be increased for the higher levels of data. The number of operation options performed can be increased by increasing the select line length.

Limitation of design:

- This design does not perform some mathematical operations such as multiplication.

Scope for improvement:

- Code convertor can be added along with the operations to convert the binary operands into gray code, excess- 3 code, binary coded decimals etc.

We will work on the limitations and improvements in the later stages of semester as we progress with the project.

Block Diagram:

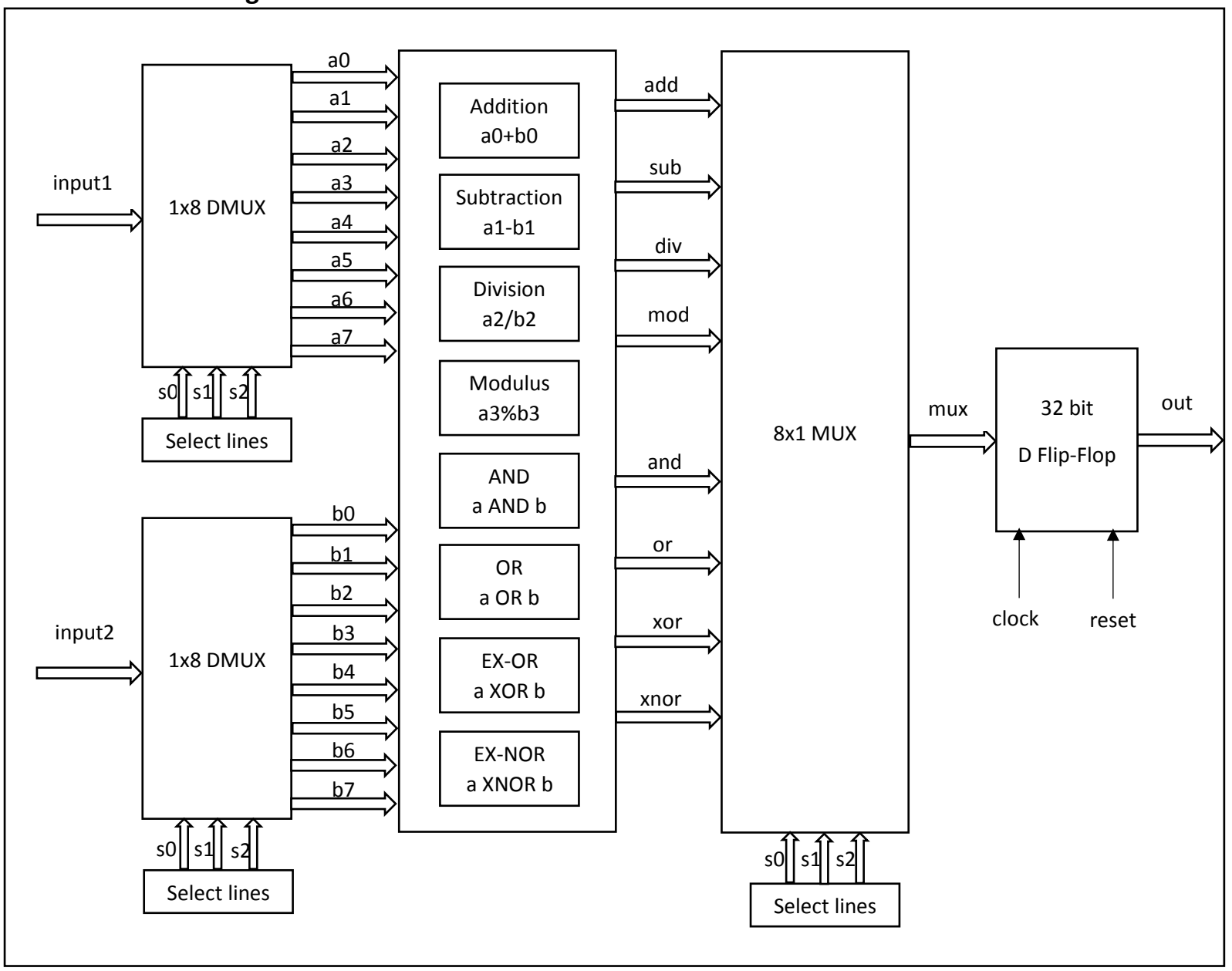

Figure 1.0 Block diagram of Arithmetic and Logic Unit (ALU)

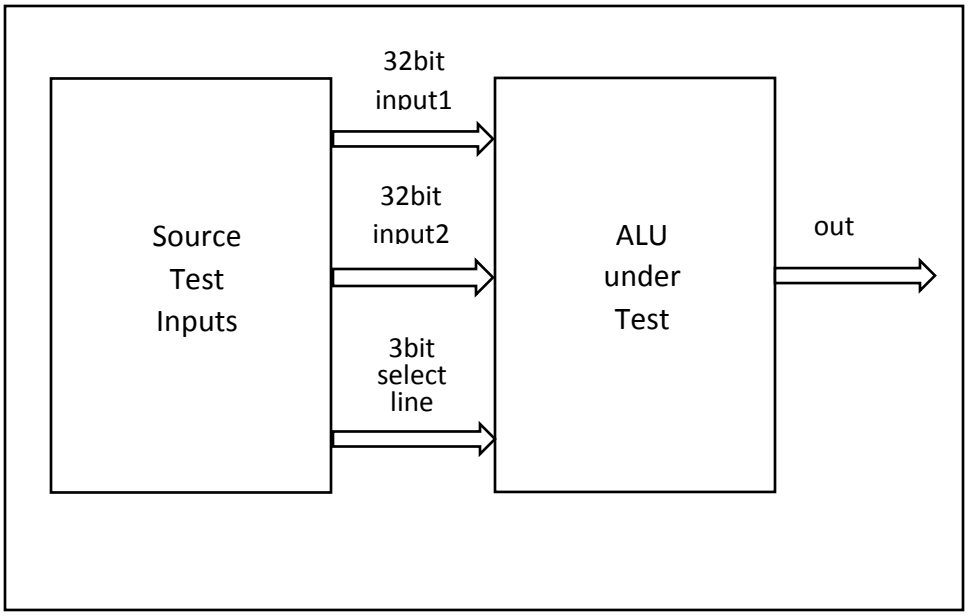

Figure 2 Connection of Testbench and ALU module

Source Code:

```
module
boosted_alu(input1,input2,select,clock,reset,out,mux,adn,sub,div,remainder,andd,orr,exxor,ex
xnor);
input [31:0]input1,input2; //Two input numbers
input clock,reset; //clock and reset for flipflop circuit
input [2:0]select; //select input to determine one of the eight operations
output [32:0]out,mux,adn,sub;

/*  out is a final output
mux is an main output available at the output of multiplexer connected to the input of flipflop
adn is the addition output with carryout as MSB
sub is the subtraction output with borrow as MSB */

output [31:0]div,remainder,andd,orr,exxor,exxnor;

/*div is division of input numbers
remainder is remainder after division of input numbers*/
```

```
reg [31:0]a1,a2,a3,a4,a5,a6,a7,a0,b1,b2,b3,b4,b5,b6,b7,b0;
/*a0 through a7 is input1 at the output of demux1
 b0 through b7 is input2 at the output of demux2*/
reg [32:0]out,mux;
reg [31:0]div,remainder,andd,orr,exxor,exxnor;
reg [32:0]adn,sub;

//1:8 demux1 code for input1
always@(*)
begin
case (select)
//input is sent to selected output line and other lines are made don't care to avoid latches
3'b000 :begin  a0<=input1;
a1<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a2<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a3<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a4<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a5<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a6<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a7<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
end

3'b001 :begin a1<=input1;
a0<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a2<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a3<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a4<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a5<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a6<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a7<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
end
3'b010 :begin a2<=input1;
a0<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a1<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a3<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a4<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a5<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a6<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a7<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
end
```

```
3'b011 :begin a3<=input1;
a0<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a1<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a2<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a4<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a5<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a6<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a7<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
end
3'b100 :begin a4<=input1;
a0<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a1<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a2<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a3<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a5<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a6<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a7<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
end
3'b101 :begin a5<=input1;
a0<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a1<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a2<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a3<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a4<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a6<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a7<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
end
3'b110 :begin a6<=input1;
a0<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a1<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a2<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a3<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a4<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a5<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a7<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
end
3'b111 :begin a7<=input1;
a0<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a1<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a2<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a3<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
```

```
a4<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a5<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
a6<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
end
default $display("select input is wrong");
endcase
end

//1:8 demux2 code for input2
always@(*)
begin
case (select)
//input is sent to selected output line and other lines are made don't care to avoid latches
3'b000 :begin  b0<=input2;
b1<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b2<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b3<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b4<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b5<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b6<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b7<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
end
3'b001 :begin b1<=input2;
b0<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b2<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b3<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b4<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b5<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b6<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b7<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
end
3'b010 :begin b2<=input2;
b0<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b1<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b3<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b4<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b5<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b6<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b7<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
end
3'b011 :begin b3<=input2;
```

```
b0<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b1<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b2<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b4<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b5<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b6<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b7<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
end
3'b100 :begin b4<=input2;
b0<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b1<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b2<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b3<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b5<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b6<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b7<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
end
3'b101 :begin b5<=input2;
b0<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b1<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b2<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b3<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b4<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b6<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b7<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
end
3'b110 :begin b6<=input2;
b0<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b1<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b2<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b3<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b4<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b5<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b7<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
end
3'b111 :begin b7<=input2;
b0<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b1<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b2<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b3<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b4<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
```

```
b5<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
b6<=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
end
default $display("select input is wrong");
endcase
end

//code for ALU
always@(*) //Arithmatic and Logical operations
begin

//ARITHMATIC
adn=a0+b0; //addtion module
sub=a1-b1; //subtraction module
div=a2/b2; //input1 is divided by input2
remainder=a3%b3; //remainder after input1 is divided by input2

//LOGICAL
andd=a4&b4; //input1 AND input2
orr=a5|b5; //input1 OR input2
exxor=a6^b6; //input1 XOR input2
exxnor=a7^~b7; //input1 XNOR input2
end

//8:1 mux code for forwarding the output of selected operation
always@(*)
begin
case (select)
3'b000 : mux<=adn;
3'b001 : mux<=sub;
3'b010 : mux<=div;
3'b011 : mux<=remainder;
3'b100 : mux<=andd;
3'b101 : mux<=orr;
3'b110 : mux<=exxor;
3'b111 : mux<=exxnor;
default $display("select input is wrong");
endcase
end

//Flip-flop code mudule
```

```
always@(posedge clock)
begin
   if (reset) begin
    out<=0;
   end
   else begin
    out<=mux;
   end
end
endmodule
```

Testbench:

```
module boosted_alu_tb;
// Inputs
reg [31:0] input1;
reg [31:0] input2;
reg [2:0] select;
reg clock;
reg reset;
// Outputs
wire [32:0] out;
wire [32:0] mux;
wire [32:0] adn;
wire [32:0] sub;
wire [31:0] div;
wire [31:0] remainder;
wire [31:0] andd;
wire [31:0] orr;
wire [31:0] exxor;
wire [31:0] exxnor;
// Instantiate the Unit Under Test (UUT)
boosted_alu uut (
.input1(input1),
.input2(input2),
.select(select),
.clock(clock),
.reset(reset),
.out(out),
.mux(mux),
```

```
.adn(adn),
.sub(sub),
.div(div),
.remainder(remainder),
.andd(andd),
.orr(orr),
.exxor(exxor),
.exxnor(exxnor)
);

initial begin
// Initialize Inputs
input1 = 32'b00001000111101010011010110000110;
input2 = 32'b00001000111101010011010110000001;
reset = 0;
clock = 1;
select = 3'b000; //addition
#20
select = 3'b001; //subtraction
#20
select = 3'b010; //division
#20
select = 3'b011; //remainder
#20
select = 3'b100; //bitwise AND
#20
select = 3'b101; //bitwise OR
#20
select = 3'b110; //bitwise ExOR
#20
select = 3'b111; //bitwise ExNOR
end
//clock
always begin
#5 clock=~clock; //invert after every 5ns
end
endmodule
```

Simulation results:

The below output waveforms shows that the ALU is calculating only one operation which is selected by the input lines. Other operation are ignored and not calculated even internally.

Figure 3: Addition

input1 = 00001000111101010011010110000110
input2 = 00001000111101010011010110000001
out = 00001000111101010011010110000111

Figure 4: Subtraction

input1 = 00001000111101010011010110000110
input2 = 00001000111101010011010110000001
out = 00000000000000000000000000000101

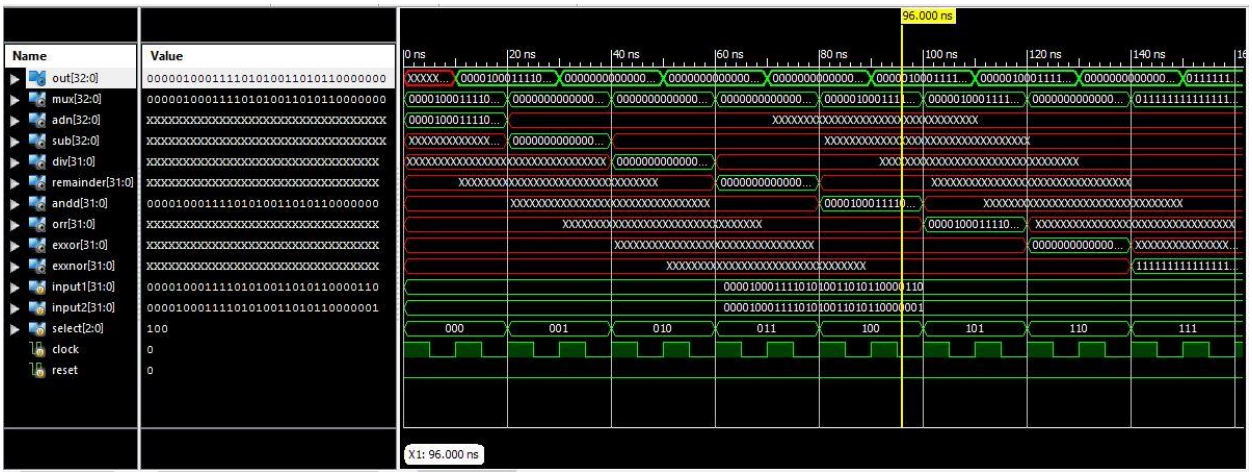

Figure 5: Bitwise AND

input1 = 00001000111101010011010110000110
input2 = 00001000111101010011010110000001
out = 00001000111101010011010110000000

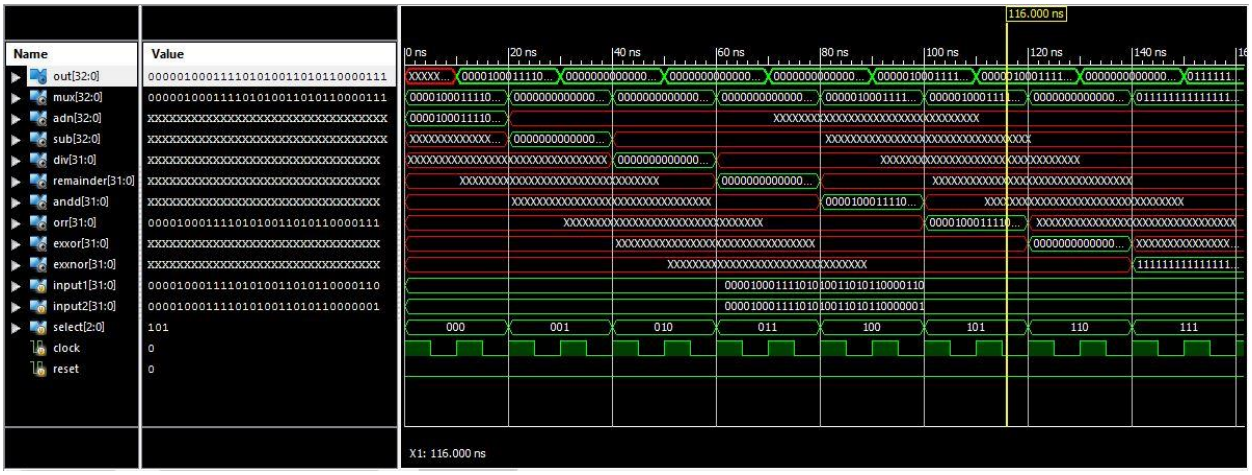

Figure 6: Bitwise OR

input1 = 00001000111101010011010110000110
input2 = 00001000111101010011010110000001
out = 00001000111101010011010110000111