Arithmetic Logic Unit

Harinee Rathod (20BEC101) , Rohan Sarvaiya (20BEC104)

Introduction

The arithmetic logic unit (ALU) is a key component of a computer's central processing unit (CPU). It performs arithmetic and logical operations on binary numbers, including addition, subtraction, multiplication, division, AND, OR, and NOT operations. Also ALU handles bitwise operations such as shifting and rotating bits in a binary number. Without the ALU, a computer would be unable to execute instructions or perform the necessary calculations for operation. A testbench is essentially a simulation environment that allows engineers and developers to test the functionality of the ALU before it is integrated into the larger system. This saves significant time and resources in the long run, as errors that are caught later in the development process can be much more costly to fix.

Code

```
module ALU(
    input [7:0] a,
    input [7:0] b,
    input [2:0] opcode,
    output reg [7:0] result,
    output reg zero,
    output reg carry,
    output reg overflow
);

// Declare internal signals
reg [7:0] temp;

// Compute result based on opcode
always @(*) begin


    case (opcode)
        3'b000: temp = a & b; // AND operation

        3'b001: temp = a | b; // OR operation

        3'b010: {carry,temp} = a + b; // ADD operation

        3'b011: {carry,temp} = a - b; // SUB operation

        3'b100: temp = a << b; // Left shift operation
```

```
        3'b101: temp = a >> b; // Right shift operation

        3'b110: temp = ~a; // NOT operation

        3'b111: temp = a ^ b; // XOR operation

        default: temp = 8'b0;
    endcase
end

// Compute zero flag
always @(*) begin
    zero = (result == 8'b0);
end

// Compute carry flag


// Compute overflow flag
always @(*) begin
            if( (a[7] == b[7]) && ( (opcode == 3'b010) || (opcode == 3'b011) ) )begin
    overflow = (temp[7] != a[7]);
            end
            else overflow = 1'b0;

end

always @(*)begin
  result = temp;
end


endmodule
```

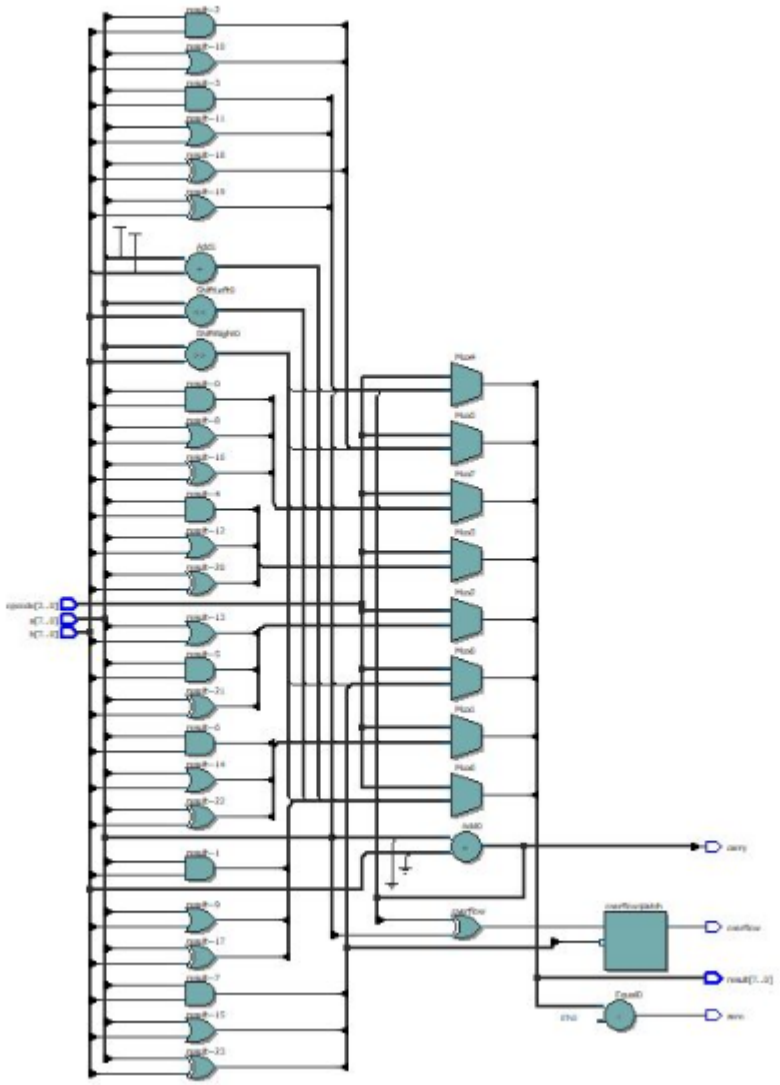
Testbench

```
module ALU_tb();

reg [7:0] a,b;
reg [2:0]opcode;
wire [7:0]result;
wire carry,zero,overflow;

integer i;

ALU tb(
```

```
.a(a),
.b(b),
.opcode(opcode),
.zero(zero),
.carry(carry),
.overflow(overflow),
.result(result));

integer golden_result;
integer golden_carry;
integer golden_zero;
integer golden_overflow;

initial
 begin

 for (i = 0; i < 10; i=i+1)
 begin
        // Generate random inputs
        a = $random;
        b = $random;
        opcode = $urandom_range(0, 8);

        // initialising the golden output
        case (opcode)
          3'b000: golden_result = a&b;
          3'b001: golden_result = a|b;
          3'b010: {golden_carry,golden_result} = a+b;
          3'b011: {golden_carry,golden_result} = a-b;
          3'b100: golden_result = a<<b;
          3'b101: golden_result = a>>b;
          3'b110: golden_result = ~a;
          3'b111: golden_result =  a^b;
                                    default: golden_result=0;
       endcase


   // initializing golden flag


   golden_zero = (golden_result==0);
   golden_overflow = ( (opcode == 3'b010 || opcode==3'b011) && (a[7] == b[7]) && (a[7] !=
golden_result[7]) );



  #100;
```

```
 // if the golden_result != result
 if(golden_result!=result||golden_carry!=carry||golden_zero!=zero||golden_overflow!=overflow)
  begin
            $monitor("Test %3d failed: a=%8b |,b=%8b,| opcode=%3b | Expected result=%15b,
carry=%3d, overflow=%3d, zero=%3d, Actual result=%15b, carry=%3d, overflow=%3d, zero=%3d",
             i, a,b,opcode,golden_result, golden_carry, golden_overflow,golden_zero,result,carry,
overflow, zero);

           end

   else
           begin

         $monitor("Test %3d passed: a=%8b | b=%8b | opcode=%3b |Expected result=%8b|,
carry=%3d|, overflow=%3d|, zero=%3d|, Actual result=%10b|, carry=%3d|, overflow=%3d|,
zero=%3d|",
             i,a,b,opcode, golden_result, golden_carry, golden_overflow,golden_zero,result,carry,
overflow, zero);

   end
  end

end
endmodule
```

RTL



Result

```
ModelSim> vsim -gui work.ALU_tb
# vsim -gui work.ALU_tb
# Loading work.ALU_tb
# Loading work.ALU
VSIM 2> run -all
# Test   1 passed: a=00001001 | b=01100011 | opcode=010 |Expected result=01101100|, carry=  0|, overflow=  0|, zero=  0|, Actual result=  01101100|, carry=  0|, overflow=  0|, zero=  0|
# Test   2 passed: a=00001101 | b=10001101 | opcode=000 |Expected result=00001101|, carry=  0|, overflow=  0|, zero=  0|, Actual result=  00001101|, carry=  0|, overflow=  0|, zero=  0|
# Test   3 passed: a=01100101 | b=00010010 | opcode=001 |Expected result=01110111|, carry=  0|, overflow=  0|, zero=  0|, Actual result=  01110111|, carry=  0|, overflow=  0|, zero=  0|
# Test   4 passed: a=00000001 | b=00001101 | opcode=101 |Expected result=00000000|, carry=  0|, overflow=  0|, zero=  1|, Actual result=  00000000|, carry=  0|, overflow=  0|, zero=  1|
# Test   5 passed: a=01110110 | b=00111101 | opcode=100 |Expected result=00000000|, carry=  0|, overflow=  0|, zero=  1|, Actual result=  00000000|, carry=  0|, overflow=  0|, zero=  1|
# Test   6 passed: a=11101101 | b=10001100 | opcode=111 |Expected result=01100001|, carry=  0|, overflow=  0|, zero=  0|, Actual result=  01100001|, carry=  0|, overflow=  0|, zero=  0|
# Test   7 passed: a=11111001 | b=11000110 | opcode=011 |Expected result=00110011|, carry=  0|, overflow=  1|, zero=  0|, Actual result=  00110011|, carry=  0|, overflow=  1|, zero=  0|
# Test   8 passed: a=11000101 | b=10101010 | opcode=000 |Expected result=10000000|, carry=  0|, overflow=  0|, zero=  0|, Actual result=  10000000|, carry=  0|, overflow=  0|, zero=  0|
# Test   9 passed: a=11100101 | b=01110111 | opcode=100 |Expected result=00000000|, carry=  0|, overflow=  0|, zero=  1|, Actual result=  00000000|, carry=  0|, overflow=  0|, zero=  1|
# Test  10 passed: a=11100101 | b=01110111 | opcode=100 |Expected result=00000000|, carry=  0|, overflow=  0|, zero=  1|, Actual result=  00000000|, carry=  0|, overflow=  0|, zero=  1|

VSIM 3>
```

Conclusion

In this project we learnt about 8-bit Arithmetic logic unit (ALU), its importance and its functionality, also we learnt about developing the layered testbench for the same. the 8-bit Arithmetic logic unit (ALU) is a vital component in many computer systems, as it performs arithmetic and logical operations on binary numbers. Using a layered testbench allows developers to catch and fix errors in the ALU's design before it is integrated into the larger system, which saves time and resources in the long run.