PROJECT REPORT

OBJECTIVE

The main objective of this project is to implement 8 bit ALU on XILINX. We will make ALU which perform 16 different operations.

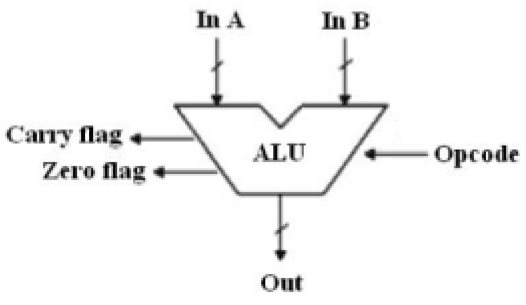
INTRODUCTION

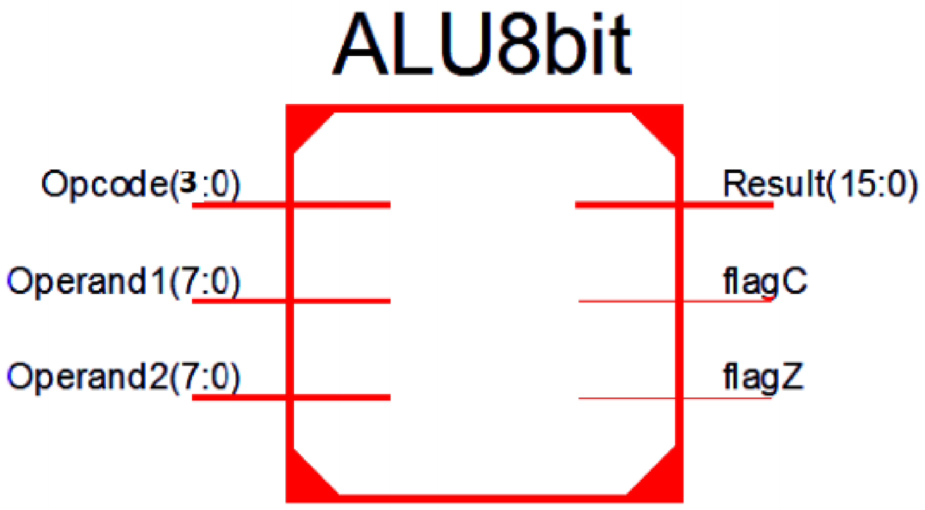
An arithmetic-logic unit (ALU) is the part of a computer processor (CPU) that carries out arithmetic and logic operations on the operands in computer instruction words. In some processors, the ALU is divided into two units, an arithmetic unit (AU) and a logic unit (LU). Some processors contain more than one AU - for example, one for fixed-point operations and another for floating-point operations. (In personal computers floating point operations are sometimes done by a floating point unit on a separate chip called a numeric coprocessor.)

WHAT IS XILNIX

Xilinx ISE (Integrated Synthesis Environment) is a product device created by Xilinx for blend and examination of HDL structures, empowering the engineer to orchestrate ("order") their plans, perform timing investigation, analyze RTL graphs, recreate a structure's response to various improvements, and design the objective gadget

Block Diagram





VERILOG CODES FOR ALU

Some of the Verilog codes for ALU are:

```
--------------------------------------------------------------
--------
| 0000 |   ALU Out = A + B;
--------------------------------------------------------------
--------
| 0001 |   ALU Out = A - B;
--------------------------------------------------------------
--------
| 0010 |   ALU Out = A * B;
--------------------------------------------------------------
--------
| 0011 |   ALU Out = A / B;
--------------------------------------------------------------
--------
| 0100 |   ALU Out = A << 1;
--------------------------------------------------------------
--------
| 0101 |   ALU Out = A >> 1;
--------------------------------------------------------------
--------
| 0110 |   ALU Out = A rotated left by 1;
```

```
-------------------------------------------------------------
--------
| 0111  |   ALU Out = A rotated right by 1;
-------------------------------------------------------------
--------
| 1000 |   ALU Out = A and B;
-------------------------------------------------------------
--------
| 1001 |   ALU Out = A or B;
-------------------------------------------------------------
--------
| 1010 |   ALU Out = A xor B;
-------------------------------------------------------------
--------
| 1011 |   ALU Out = A nor B;
-------------------------------------------------------------
--------
| 1100 |   ALU Out = A nand B;
-------------------------------------------------------------
--------
| 1101 |   ALU Out = A xnor B;
-------------------------------------------------------------
--------
| 1110 |   ALU Out = 1 if A>B else 0;
-------------------------------------------------------------
--------
| 1111 |   ALU Out = 1 if A=B else 0;
```

CODES

VERILOG MODULE

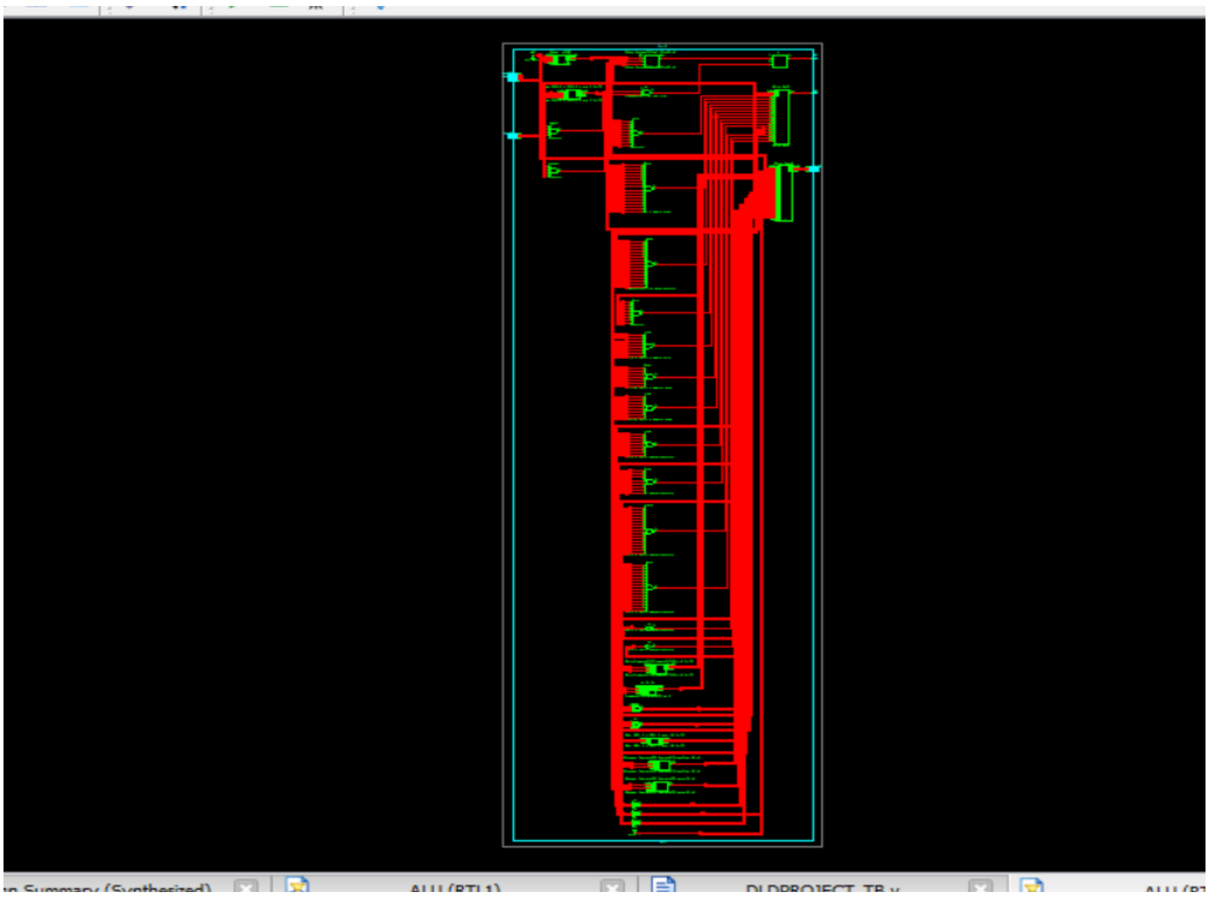
```
module ALU(
    input [7:0] Operand1,
    input [7:0] Operand2,
    input [3:0] Opcode,
    output reg[15:0] Result = 16'b0,
       output reg flagC = 1'b0,
    output reg flagZ      = 1'b0
    );


always @(*)
    begin
        case(Opcode)
        4'b0000: // Addition
               begin
           Result = Operand1 + Operand2 ;
                     flagC  = Result[8];
                     flagZ  = (Result == 16'b0);
                   end

        4'b0001: // Subtraction
        begin
           Result = Operand1 - Operand2 ;
                     flagC  = Result[8];
                     flagZ  = (Result == 16'b0);
                   end

        4'b0010: // Multiplication
         begin
           Result = Operand1 * Operand2 ;
                     flagZ  = (Result == 16'b0);
                   end

        4'b0011: // Division
          begin
           Result = Operand1 / Operand2 ;
                     flagZ  = (Result == 16'b0);
                   end

        4'b0100: // Logical shift left
                    begin
```

```
  Result =  Operand1<<1;
            flagZ  = (Result == 16'b0);
          end

4'b0101: // Logical shift right
            begin
  Result = Operand1>>1 ;
            flagZ  = (Result == 16'b0);
          end

4'b0110: // Rotate left
           begin
  Result = {Operand1[6:0],Operand1[7]} ;
            flagZ  = (Result == 16'b0);
           end

4'b0111: // Rotate right
            begin
  Result = {Operand1[7],Operand1[6:0]};
            flagZ  = (Result == 16'b0);
          end

 4'b1000: //  Logical and
             begin
  Result = Operand1 & Operand2 ;
    //      flagC  = Result[8];
            flagZ  = (Result == 16'b0);
          end

 4'b1001: //  Logical or
             begin
  Result = Operand1 | Operand2 ;
            flagZ  = (Result == 16'b0);
          end
```

```
4'b1010: //  Logical xor
          begin
 Result = Operand1 ^ Operand2 ;
           flagZ  = (Result == 16'b0);
         end

4'b1011: //  Logical nor
            begin
 Result = ~(Operand1 | Operand2) ;
           flagZ  = (Result == 16'b0);
         end

4'b1100: // Logical nand
            begin
 Result = ~(Operand1 & Operand2) ;
           flagZ  = (Result == 16'b0);
         end

4'b1101: // Logical xnor
            begin
 Result = ~(Operand1 ^ Operand2) ;
           flagZ  = (Result == 16'b0);
         end

4'b1110: // Greater comparison
       begin
 Result = (Operand1 > Operand2)?8'd1:8'd0 ;
           flagZ  = (Result == 16'b0);
          end

4'b1111: // Equal comparison
                 begin
 Result = (Operand1 == Operand2)?8'd1:8'd0 ;
```

```
                    flagZ  = (Result == 16'b0);
                  end

          default:
                        begin
                    Result = 16'b0;
                          flagC  = 1'b0;
                          flagZ  = 1'b0;
                         end

        endcase
    end

endmodule
```

OUTPUT



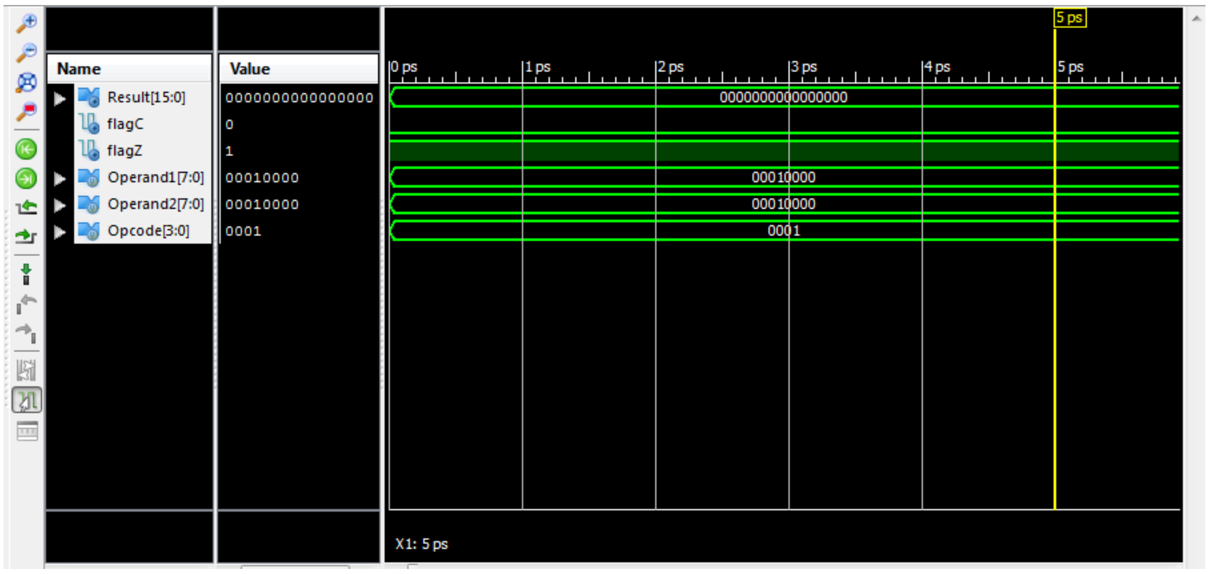
VERILOG TEXT FIXTURE

CODE

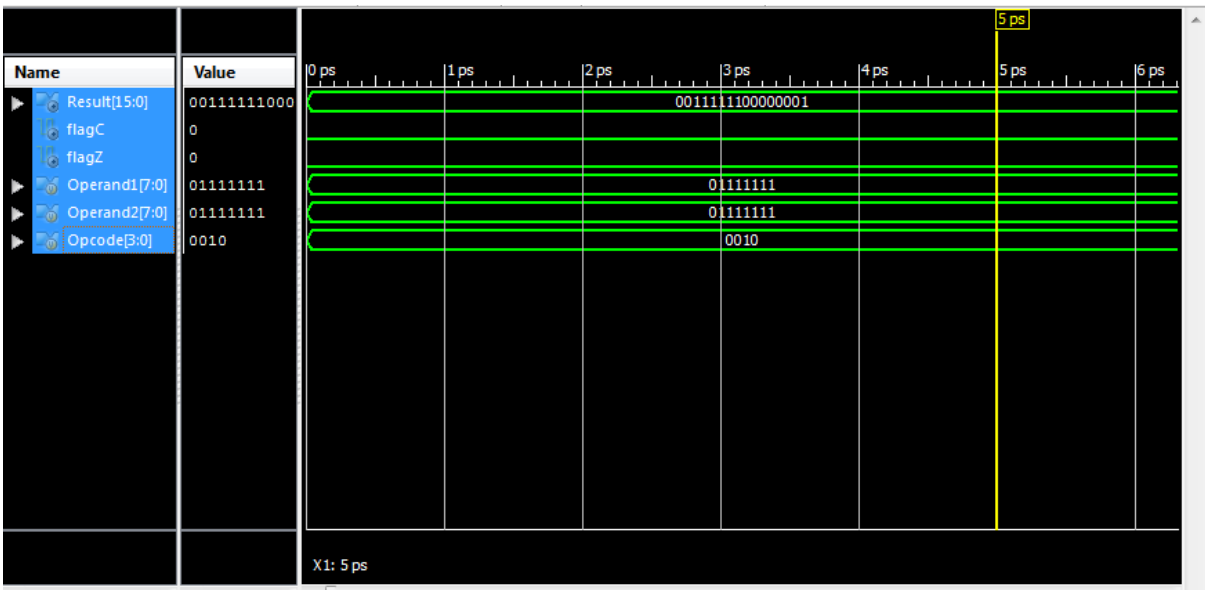
```
module ALU_tb;

        // Inputs
        reg [7:0] Operand1;
        reg [7:0] Operand2;
        reg [3:0] Opcode;

        // Outputs
        wire [15:0] Result;
        wire flagC;
        wire flagZ;
```

```
	// Instantiate the Unit Under Test (UUT)
	ALU uut (
		.Operand1(Operand1), 
		.Operand2(Operand2), 
		.Opcode(Opcode), 
		.Result(Result), 
		.flagC(flagC), 
		.flagZ(flagZ)
	);

	initial begin
		// Initialize Inputs
		Operand1 = 16;
		Operand2 = 16;
		Opcode = 1;

		// Wait 100 ns for global reset to finish
		#100;
        
		// Add stimulus here

	end
      
endmodule
```

OUTPUT



Name	Value
Result[15:0]	0000000000000000
flagC	0
flagZ	1
Operand1[7:0]	00010000
Operand2[7:0]	00010000
Opcode[3:0]	0001



Name	Value
Result[15:0]	00111111000
flagC	0
flagZ	0
Operand1[7:0]	01111111
Operand2[7:0]	01111111
Opcode[3:0]	0010