DESIGN AND SIMULATION OF AN 8-BIT ALU

Objective:

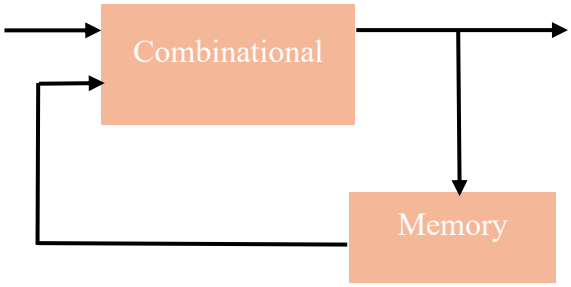
To design and verify a 8-bit Arithmetic Logic Unit (ALU) implementing basic arithmetic and logical operations using Verilog HDL.

Software used:

Visual Studio Code, Icarus Verilog and GTKWave.

Methodology:

The ALU was modeled using combinational logic in Verilog. A testbench was developed to check the functionality of the main code. Functional verification was performed using iVerilog and GTKWave.

Combinational	Sequential
Output only depends on the present input.	Output depends on the present input and past output.
Memory element is absent.	Memory element is present.
No clock signal is applied.	Clock signal is required.
	
Example: Half adder, Full adder, Multiplexer	Example: Flipflops, Counters, Registers

Operations:

Opcode	Operation
000	A + B (Addition)
001	A - B (Subtraction)
010	A & B (AND)
011	A | B (OR)
100	A^B (XOR)
101	~A (NOT)
110	A<<1 (Logical left Shift)
111	A>>1(Logical Right Shift)

Verilog Code:

```
module alu (
        input [7:0] A,
        input [7:0] B,
        input [2:0] opcode,
        output reg [7:0] result,
        output reg carry,
        output reg zero
);
always @(*) begin
        carry = 0;
        case (opcode)
                3'b000: {carry, result} = A + B;        // ADD
                3'b001: {carry, result} = A - B;        // SUB
                3'b010: result = A & B;                 // AND
                3'b011: result = A | B;                 // OR
                3'b100: result = A ^ B;                 // XOR
                3'b101: result = ~A;                    // NOT
                3'b110: result = A << 1;                // LSHIFT
```

```
            3'b111: result = A >> 1;            // RSHIFT
            default: result = 8'b0;
        endcase
        zero = (result == 0);
end
endmodule
```

Test bench:

```
module alu_tb;
reg [7:0] A, B;
reg [2:0] opcode;
wire [7:0] result;
wire carry, zero;

alu uut (
        .A(A),
        .B(B),
        .opcode(opcode),
        .result(result),
        .carry(carry),
        .zero(zero)
);

initial begin
        $dumpfile("alu.vcd");
        $dumpvars(0, alu_tb);

        A = 8'd10; B = 8'd5;
        opcode = 3'b000; #10;  // ADD
        opcode = 3'b001; #10;  // SUB
```

```
        opcode = 3'b010; #10;  // AND
        opcode = 3'b011; #10;  // OR
        opcode = 3'b100; #10;  // XOR
        opcode = 3'b101; #10;  // NOT
        opcode = 3'b110; #10;  // LSHIFT
        opcode = 3'b111; #10;  // RSHIFT
        $finish;
end
endmodule
```

Expected outcome:

Opcode	Operation	A	B	Expected Result (R)
000	A + B	10	5	15
001	A - B	10	5	5
010	A & B	10	5	0
011	A | B	10	5	15
100	A^B	10	5	15
101	~A	10	-	5
110	A<<1	10	-	20
111	A>>1	10	-	5

Calculations:

In decimal, A = 8'd10; B = 8'd5

In binary, A = 4'b1010; B = 4'b0101

In hexadecimal, A = 0xA; B= 0x5

Addition:

In decimal, A + B = 10 + 5 = 8'd15

In hexadecimal, A + B = 0xA + 0x5 = 0xF

Subtraction:

In decimal, A - B = 10 + 5 = 8'd5

In hexadecimal, A - B = 0xA + 0x5 = 0x5

AND operation:

In binary,

A = 1010

B = 0101

R = 0000

In hexadecimal, R = 0x0

OR operation:

In binary,

A = 1010

B = 0101

R = 1111

In hexadecimal, R = 0xF

XOR operation:

In binary,

A = 1010

B = 0101

R = 1111

In hexadecimal, R = 0xF

NOT operation:

In binary,

A = 1010

~A = R = 0101

In hexadecimal, R = 0x5

Left Shift by 1-bit:

In binary,

A = 0000 1010

R = 0001 0100

In hexadecimal, R = 0x14

Right Shift by 1-bit:

In binary,

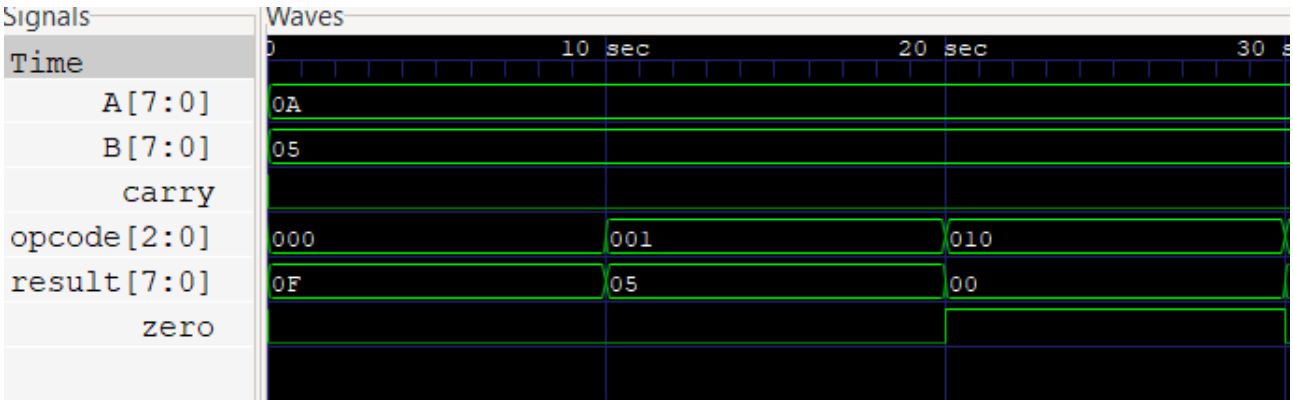
A = 0000 1010

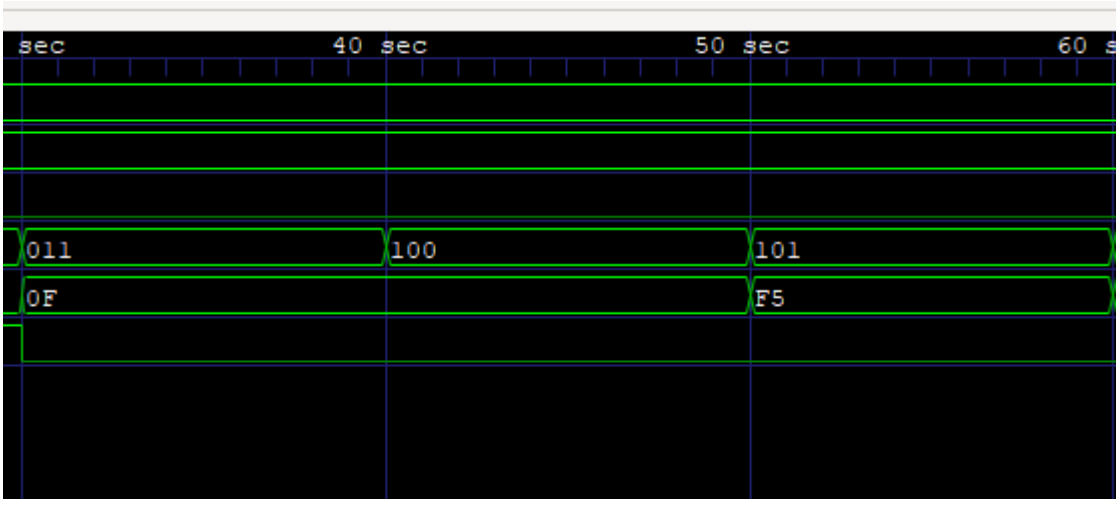
R = 0000 0101

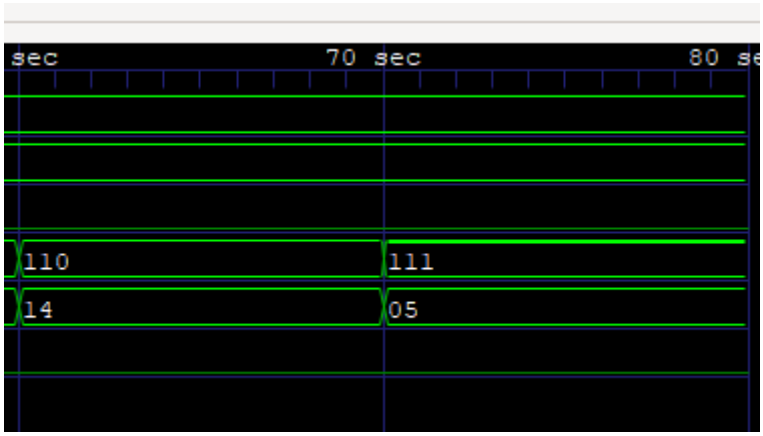
In hexadecimal, R = 0x5

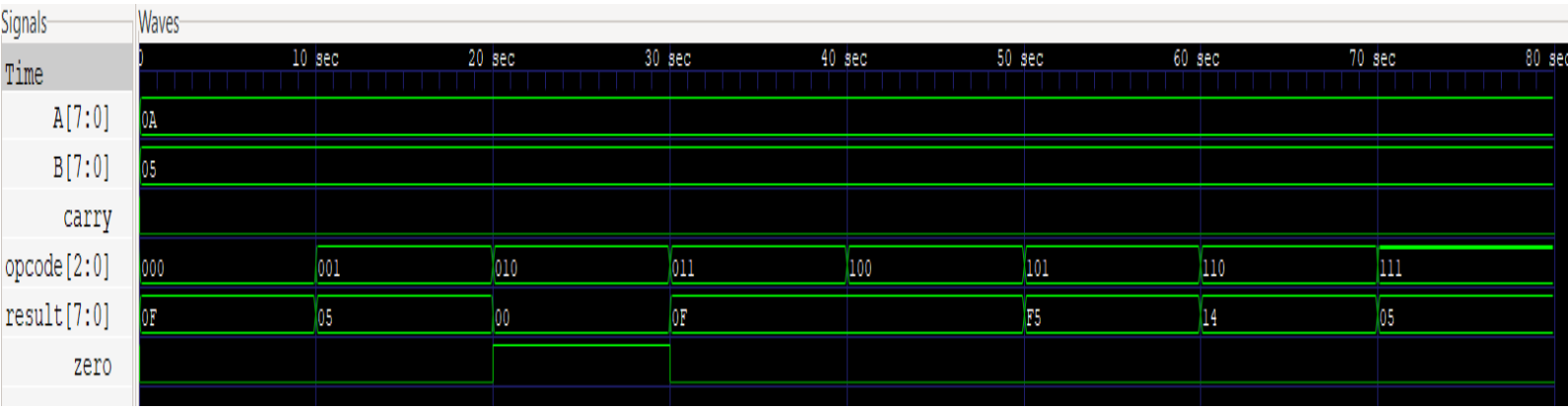
Results:

Simulation waveforms confirmed correct execution of all operations including arithmetic, logical, and shift functions. Zero and carry flags behaved as expected.






sec
70 sec
80 se
110
111
14
05


Signals
Waves
Time
10 sec
20 sec
30 sec
40 sec
50 sec
60 sec
70 sec
80 sec
A[7:0]
0A
B[7:0]
05
carry
opcode[2:0]
000
001
010
011
100
101
110
111
result[7:0]
0F
05
00
0F
F5
14
05
zero