

Report:

Title: Design and Verification of a 4-Bit Arithmetic Logic Unit using Quartus, Verilog HDL, and Timing Diagrams

Authors:

Md. Aber Islam	20301102
Israt Jahan Ayshi	20101618
Ms Rodsy Tahmid	20101021
Humaira Mir Prothoma	19101622
Bousra Jahan Lia	20301335

Abstract:

In this paper, we present the design and verification of a 4-bit Arithmetic Logic Unit (ALU) capable of performing four different arithmetic or logical operations. The ALU was designed using Quartus and implemented using Verilog HDL. A timing diagram was used to verify the functionality of the ALU. The paper outlines the design process, including the specification of the required operations, the creation of the Verilog HDL code, and the simulation and verification of the ALU using a timing diagram. The results of the simulation demonstrate the successful implementation of the four operations and the proper functioning of the ALU.

Introduction:

An arithmetic logic unit (ALU) is a fundamental component of digital circuits that performs arithmetic and logical operations on binary numbers. In this project, we aim to design and implement a 4-bit ALU capable of performing four different operations: addition, subtraction, bitwise AND, and bitwise OR. The design of the ALU was carried out using Quartus, a software tool for designing digital circuits. Verilog HDL, a hardware description language, was used to implement the ALU. To verify the functionality of the ALU, we used a timing diagram, a graphical representation of the signals in a digital circuit over time. The use of a timing diagram allowed us to ensure that the ALU was

operating correctly under various input conditions. The successful implementation of the ALU has potential applications in various digital circuits, including microprocessors and microcontrollers. The remainder of this paper outlines the design and verification process of the 4-bit ALU.

Operation:

- State Diagram

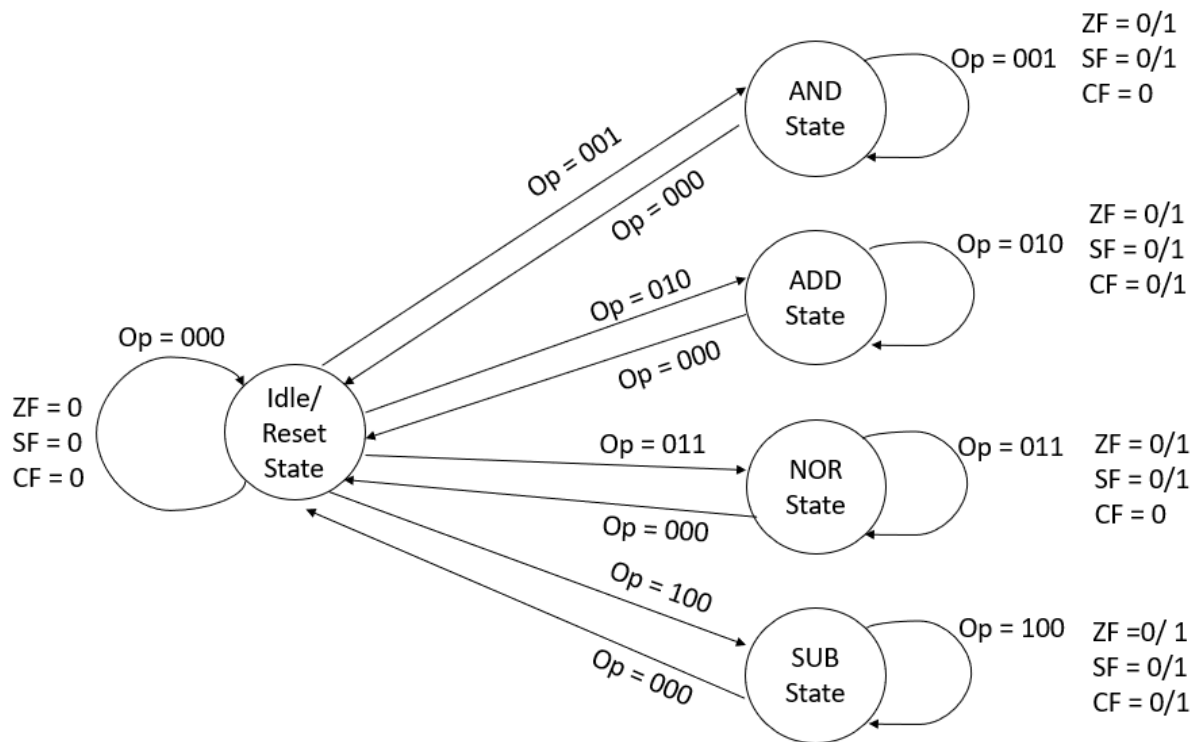


Fig. no: Simplified state diagram of an ALU with five operations: RESET, AND, ADD, NOR, SUB.

The state diagram of the system is shown in the figure above. Here 5 operations are done. RESET, AND, ADD, NOR and SUB. For each operation we use 000,001,010,011 and 100 opcodes respectively.

Truth Table:

In decimal	A				B				RESET (op = 000)	ADD (op = 001)	AND (op = 010)				NOR (op = 011)				SUB (A-B) (op = 100)
-	A3	A2	A1	A0	B3	B2	B1	B0	-	-	Y3	Y2	Y1	Y0	Y3	Y2	Y1	Y0	-
0 (+/-) 0	0	0	0	0	0	0	0	0	-	0000 (ZF=1, SF=0, CF=0)	0	0	0	0	1	1	1	1	0000
0 (+/-) 15	0	0	0	0	1	1	1	1	-	1111 (ZF=0, SF=1, CF=0)	0	0	0	0	0	0	0	0	10001 (ZF=0, SF=0, CF=1)
15 (+/-) 0	1	1	1	1	0	0	0	0	-	1111 (ZF=0, SF=1, CF=0)	0	0	0	0	0	0	0	0	0000
.	-
.	-
.	-
15 (+/-) 13	1	1	1	1	1	1	0	1	-	11100 (ZF=0, SF=1, CF=1)	1	1	0	1	0	0	0	0	0010 (ZF=0, SF=0, CF=0)
15 (+/-) 14	1	1	1	1	1	1	1	0	-	11101 (ZF=0, SF=1, CF=1)	1	1	1	0	0	0	0	0	0001
15 (+/-) 15	1	1	1	1	1	1	1	1	-	11110 (ZF=0, SF=1, CF=1)	1	1	1	1	0	0	0	0	0000

The table above shows some of the results and a few of their notable ZFs (Zero Flags), SFs (Sign Flags) and CFs (Carry Flags). When the output is zero or $Y = 0000$, $ZF = 1$ otherwise $ZF = 0$. $SF = 1$ when output, $Y = 1xxx$ or when the MSB bit is 1, else $SF = 0$. $CF = 1$ when carry bit is 1 or when $Y = 1xxxx$ (overflow), otherwise $CF = 0$.

For AND gate and NOR gate, it is said to take bitwise inputs of A and B, means if $A = 1010$ ($A3 = 1$, $A2 = 0$, $A1 = 1$, $A0 = 0$) and $B = 0101$ ($B3 = 0$, $B2 = 1$, $B1 = 0$, $B0 = 1$) AND operation would be $Y = 0000$ ($Y3 = 0$ [because $A3 = 1$ & $B3 = 0$ means 0], $Y2 = 0$ [because $A2 = 0$ & $B2 = 1$ means 0], $Y1 = 0$ [because $A1 = 1$ & $B1 = 0$ means 0], $Y0 = 0$ [because $A0 = 1$ & $B0 = 0$ means 0]). And NOR operation would be $Y = 0000$ ($Y3 = 0$ [because $A3 = 1 \square B3 = 0$ means 0], $Y2 = 0$ [because $A2 = 0 \square B2 = 1$ means 0], $Y1 = 0$ [because $A1 = 1 \square B1 = 0$ means 0], $Y0 = 0$ [because $A0 = 1 \square B0 = 0$ means 0]). This also applies to other 4-bit combinations of A, B.

16 possible inputs for A and 16 possible inputs for B. So, $16 * 16 = 256$. 256 total combinations of outputs possible including repetitions.

Table Simplified:

	A3	A2	A1	A0
Input A				
	B3	B2	B1	B0
Input B				
	Y3	Y2	Y1	Y0
Output Y				

Y4 = Carry

- Timing Diagram:

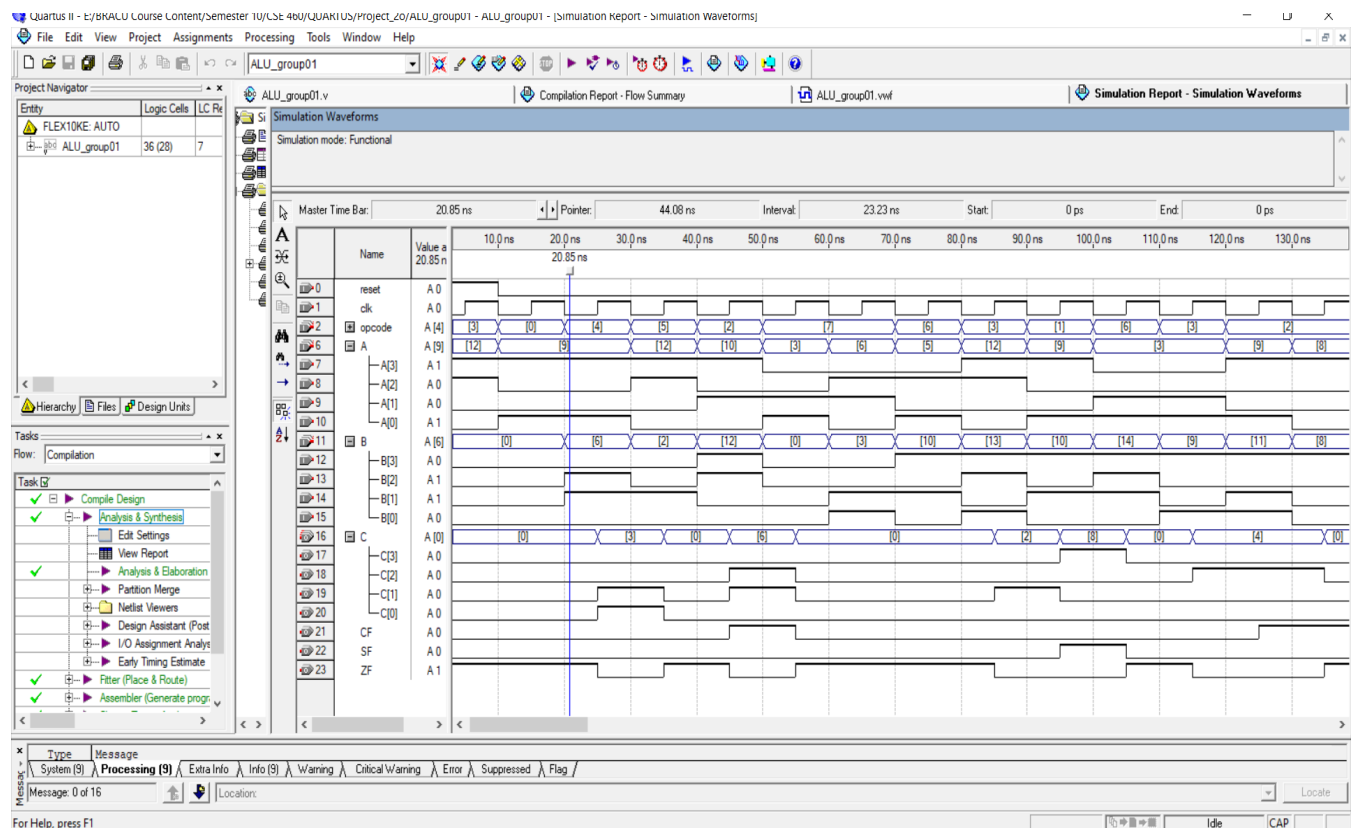


Figure1: Timing Diagram after simulating the Verilog code from Appendix 1

- Explanation

Conclusion:

In this research, we present an effective method for verifying VHDL behavioral coding. Additionally, we have put up a number of algorithms with various design levels.

Quartus has been used to implement our suggestions in Verilog. The number of bus lines has been reduced, and all designs have been put into practice and evaluated. This VHDL-based ALU design was successfully created, implemented and tested.

Appendix 1 (Verilog Code):

```
module ALU_group01(A, B, opcode, clk, reset, C, CF, ZF,SF);
    input clk,reset;
    input [3:0] A;
```

```

input [3:0] B;
input [2:0] opcode;
output reg [3:0] C;
output reg CF;
output reg ZF;
output reg SF;

// Temporary variable to store the result
reg [4:0] temp;
reg [3:0] prev_C = 4'b0000;

always @(posedge clk, posedge reset)
begin
    if (reset == 1)
    begin
        temp = 5'b00000;
        C = temp[3:0];
        CF = 0;
        ZF = 1;
        SF = 0;
    end

    else
    begin
        prev_C = C;

        case (opcode)
            3'b000: //RESET operation
            begin
                C = prev_C;
                CF = CF;
            end
        endcase
    end
end

```

end

3'b001: //AND operation

begin

temp = (A & B);

C = temp[3:0];

CF = 0;

end

3'b010: //ADD operation

begin

temp = (A + B);

C = temp[3:0];

if (temp[4] == 1)

begin

CF = 1;

end

else

begin

CF = 0;

end

end

3'b011: //NOR operation

begin

temp = ~(A | B);

C = temp[3:0];

CF = 0;

end

```

3'b100: //SUB operation
begin
    C = A - B;
    if (A >= B)
        begin
            CF = 0;
        end
    else
        begin
            CF = 1;
        end
    end
end

default:
begin
    temp = 5'bxxxxx;
    C = 4'bxxxx;
    CF = 1'bx;
end
endcase

if (C == 4'b0000)
begin
    ZF = 1;
end
else
begin
    ZF = 0;
end

SF = C[3];

```



```

end
end
endmodule

```

Appendix 2 (Compilation Report):

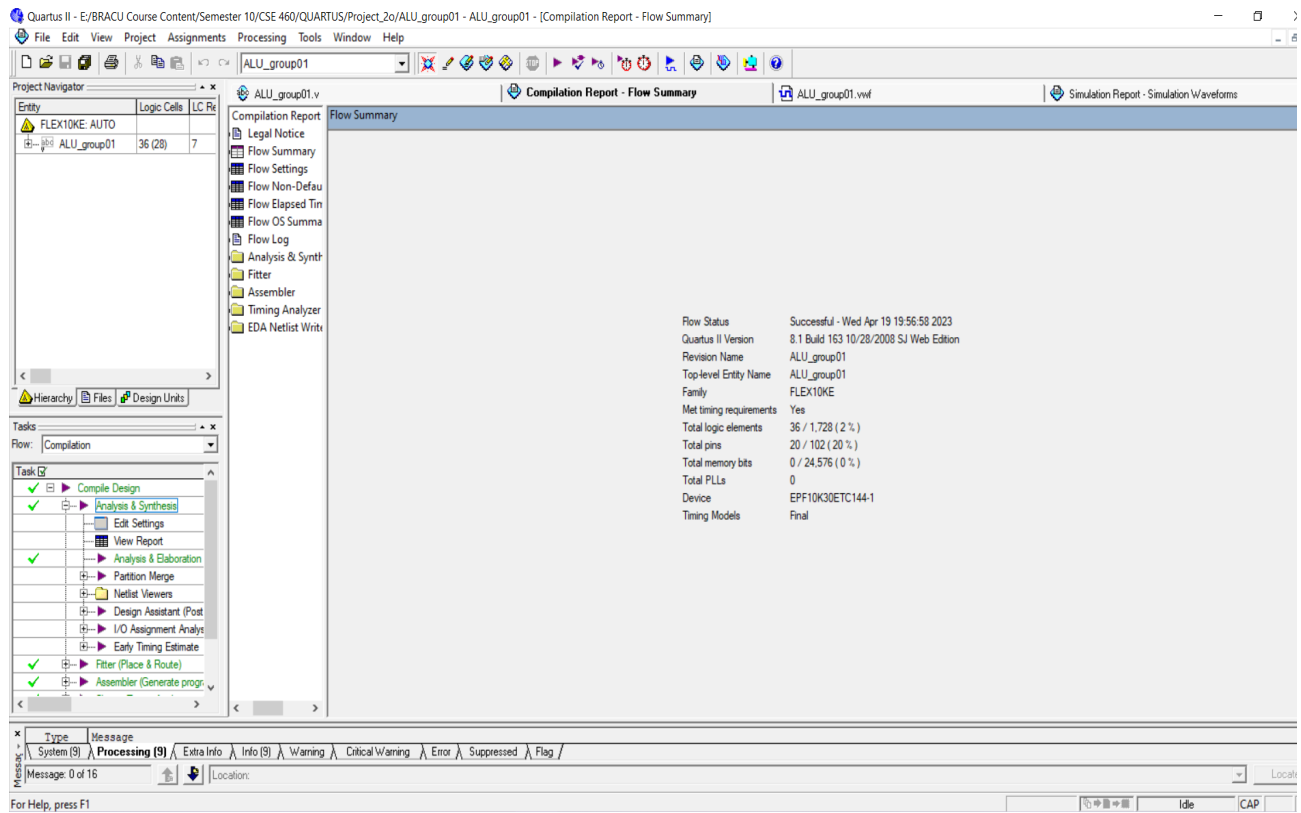


Figure 2: Compilation report after running the Verilog code from Appendix 1

IEEE format [Humaira]