ALU Arithmetic Operations Section:03 (Group 7)

Irin Hoque Orchi
Dept. of cse
Brac University
irin.hoque.orchi@g.bracu.ac.bd

Md.Omar Faroque
Dept. of cse
Brac University
md.omar.faroque@g.bracu.ac.bd

Asit Kumar Halder
Dept. of cse
Brac University
Asit.kumar.halder@g.bracu.ac.bd

Fahim kabir khan
Dept. of cse
Brac University
fahim.kabir.khan@g.bracu.ac.bd

Munem Shahriar Khan
Dept. of cse
Brac University
munem.shahriar.khan@g.bracu.ac.bd

***Abstract*—This paper is about ALU arithmetic operations. The ALU can do four different kinds of math. The Arithmetic Reasoning Unit (ALU) is used to perform math and reasoning tasks. It stands for the main part of the central processing unit (CPU) of a computer. ALUs in modern CPUs are very powerful and complex. The goal of this project is to make a 4-bit, 4-operation arithmetic logic unit using adder logic. In this project, the ALU was made by changing how adders and multiplexers work. The ALU can do four operations: ADD, NOR, SUB, and XNOR. The important thing about this project is that it doesn't use logic gates for any of its operations. Instead, it uses adders and multiplexers.**

***Keywords*—ALU, Arithmetic/ Logical operations, ADD, SUB, RESET, NOR, XNOR, zero flag, sign flag, carry flag, Quartus, Verilog HDL, clock cycles, LSB, MSB, Timing diagram.**

I. Introduction

The Arithmetic Logic Unit is part of a computer that performs mathematical calculations on binary integers. On the other hand, the FPU (Floating Point Unit) works with decimal numbers. This ALU is comprised of the CPU (central processing unit), FPU (floating point unit), and GPU (graphics processing unit). As a consequence, more than one ALU may be contained in a single CPU or FPU. The input to an ALU is the data on which we must conduct operations. They are referred to as "operands." They do the necessary activity, and the output is the result of the action we performed. The ALU may thus have input, output, or even both. They also include registries and the results of current or previous operations. The Central Processing Unit stores, retrieves, and processes data via registers. Processor registers are the registers that the CPU uses to process data. Modern computers have very complex Arithmetic Logic Units. They could also consist of the Control Unit (CU). Data transfer between the ALU, memory, and registers is handled by the central processing unit (CPU).

II. Methodology

An essential part of a computer system's central processing unit is the arithmetic logic unit (ALU). On the instruction words, it performs the appropriate arithmetic and logical

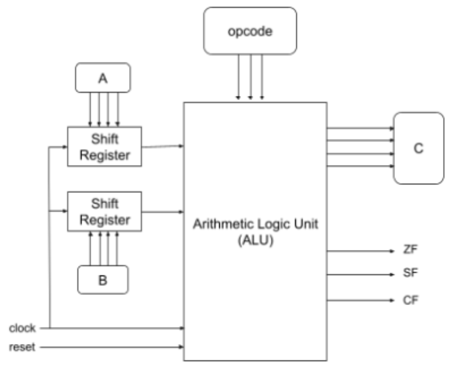

Fig. 1. Block Diagram Of ALU

operations. The arithmetic unit (AU) and the logic unit (LU) are two ways that the arithmetic unit (ALU) is divided into several microprocessor designs. Engineers may create an ALU to compute any operation. The ALU becomes more costly, occupies more CPU space, and generates more heat as the operations get more complicated. Engineers ensure that the ALU is strong enough for the CPU to be equally strong and rapid while not being too complicated to make it prohibitively costly or have other issues. An integer unit (IU) is another name for an arithmetic logic unit. The main tasks of the ALU are bit-shifting operations, arithmetic calculations, and logic calculations. Almost all of the data that the CPU processes require these basic actions to be carried out.

III. Operation

This is a simplified state diagram that shows how the system operates. The system starts in an idle state and waits for an opcode to be input. There are four possible opcodes that correspond to four different operations (ADD, SUB, NOR, XNOR). The opcode should not be changed during an operation. Once an operation is started, the system needs a certain number of clock cycles to complete it. After completion, the opcode needs to be manually reset to 000 using the input shown in the timing diagram to reset the system.

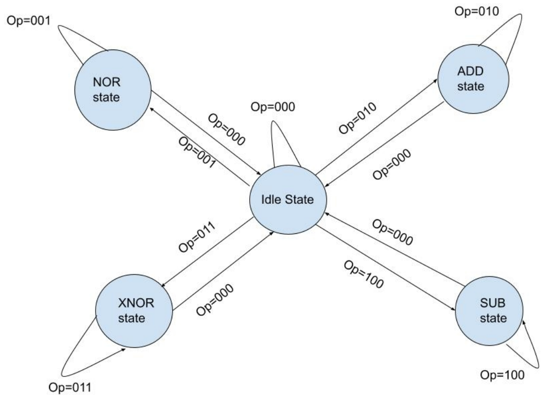

Fig. 2.

IV. Timing Diagram

The timing diagrams are shown from the output of the waveform.

A. NOR

When the op code is 001, it will perform a NOR operation. On A and B, the ALU will do a bitwise NOR operation. At the positive edge of the input clock, the bitwise operation will be carried out sequentially. For example, during the first clock cycle, the operation will be carried out on the LSBs (A0, B0), storing the result in C0. The next clock cycle, the operation will be carried out on A1 and B1, updating C1. Finally, the operation will be carried out on A3 and B3, saving the result to C3 as long as the opcode is active.

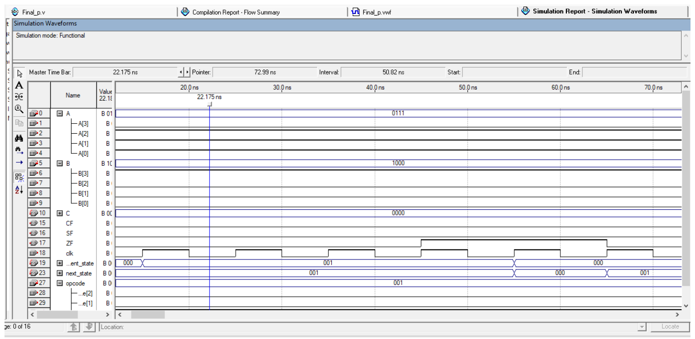

Fig. 3. NOR Operation

B. ADD

In a digital circuit design, an opcode of 010 signifies that the arithmetic logic unit (ALU) will execute the ADD operation on inputs A and B. This operation will be performed in a serial manner, meaning that it will be executed one bit at a time, sequentially from the LSBs to the MSBs. The operation will be triggered by the positive edge of the input clock, which acts as a synchronous signal for the circuit.

During the first clock cycle, the ALU will perform the ADD operation on the least significant bits (A0 and B0), and the result will be stored in C0. In the subsequent clock cycles, the ALU will perform the ADD operation on the next bits (A1 and B1), and the result will be stored in C1. This process will continue until the ADD operation has been executed on all bits of A and B, and the final result is stored in C. The advantage of performing the ADD operation in a serial manner is that it reduces the complexity of the circuit and minimizes the number of components required. Additionally, performing the operation sequentially also reduces the power consumption and increases the speed of the circuit since it can operate at a higher clock frequency.

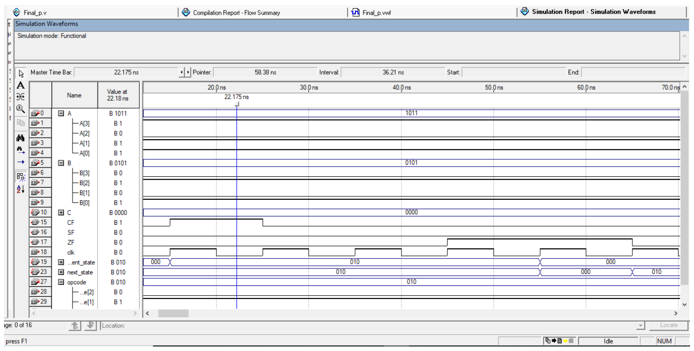

Fig. 4. ADD Operation

C. XNOR

When the op code is 011, it will perform XNOR operation. On A and B, the ALU will execute a bitwise XNOR operation. At the positive edge of the input clock, the bitwise operation will be carried out sequentially. For example, during the first clock cycle, the operation will be carried out on the LSBs (A0, B0), saving the result in C0. Then, during the next clock cycle, the operation will be carried out on A1, B1, updating C1, and finally, on A3 and B3, saving it to C3, as long as the opcode is active.
In this instance, we provide the ALU with two inputs (A, B) and a specific opcode (011) for performing a SNOR operation. By examining the timing diagram, it is evident that the desired output is obtained for the given input and opcode. The ALU operates on a bitwise clock, and the output is determined based on it. The ALU typically utilizes sign, zero, and carry flags to operate fully, but they are not relevant for the SNOR operation since it is a NOR operation. This operation requires only a 4-bit A and B input and a 3-bit opcode, and it is designed to work on a positive clock bitwise operation.

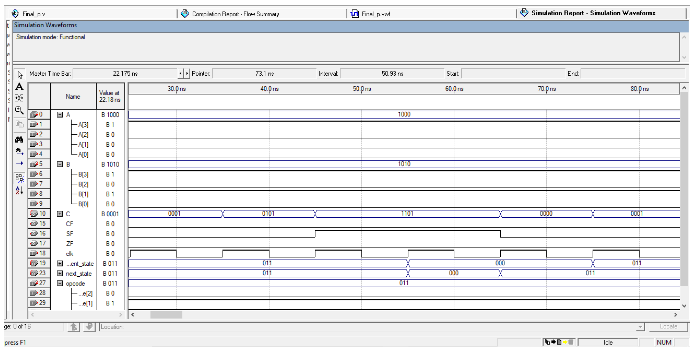

Fig. 5. XNOR Operation

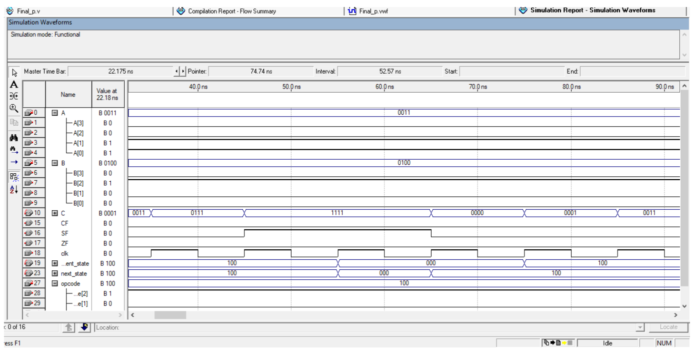

Fig. 6. SUB Operation

D. SUB

When the opcode is 100, the SUB operation is performed by the ALU which subtracts B from A. The operation is carried out in serial while the opcode is active and at the positive edge of the input clock. The operation starts with the LSBs (A0, B0) during the first clock cycle, and the result is stored in C0. Then, during the next clock cycle, A1 is updated, and C1 is updated accordingly. Finally,during the last clock cycle, A3 and B3 are operated on, and the result is saved in C3. A3 and B3 should be treated as sign bits during the SUB operation as they represent the MSBs of both A and B. Thus, the subtraction is performed between two signed binary numbers, where the values of operands are stored in the remaining bits, and the MSBs are the sign bits.
So, To subtract 4 from 3 in binary using 2's complement, we first need to represent both numbers in binary form.

3 in binary is 0011, and 4 in binary is 0100. To find the 2's complement of 4, we first invert all the bits (i.e., change all 0's to 1's and all 1's to 0's) to get 1011, and then add 1 to the result to get the final 2's complement of 4, which is 1100.Now, we can perform the subtraction by adding the 2's complement of 4 to 3 as follows:
0011 (3 in binary)
1100 (2's complement of 4)
1 0001 (result in binary)
Since the leftmost bit in the result is 1, this indicates that the result is negative. To find the decimal value of the result, we need to convert the binary result back to decimal form, taking the 2's complement of the result and adding a negative sign to get the final answer. To find the 2's complement of the binary result, we first invert all the bits to get 1110, and then add 1 to the result to get the final 2's complement of the binary result, which is 1111.

E. RESET

When the opcode is 000, it will perform a reset in the alu operation. The ALU will be in the idle state (a state with no operations), and output C will still show the outcome of the most recent operation.

V. Conclusion

Numerous activities, including the construction of circuits, hardware-level programming, the solution of real-world problems, the creation of control systems, etc., can be carried out using logical and mathematical operations. Our project's ALU can be applied to real-world scenarios where these operations are required since it efficiently performs XNOR, SUB, NOR, and ADD operations (in accordance with timing diagrams and truth tables). The ALU can also be improved so that it can carry out more tasks in the future. Finally,We have successfully developed a 4-bit ALU that can execute XNOR, SUB, NOR and ADD.

VI. Appendix(Verilog Code)

```
module Final_p
(A, B, C, opcode, ZF, SF, CF,
current_state, next_state, clk, reset);

input clk;
input [2:0] opcode;
input [3:0] A, B;

output reg ZF, SF, CF, reset;
output reg [2:0] current_state, next_state;
output reg [3:0] C;

parameter [2:0] idle = 3'b000, NOR = 3'b001,
ADD = 3'b010, XNOR = 3'b011, SUB = 3'b100;
reg [2:0] i = 0;
reg [3:0] ii = 0;
reg [3:0] compliment_2s = 0;
reg [4:0] C_t;
reg F;

always@ (posedge clk)
begin
if(opcode == idle || reset == 1)
begin
current_state = idle;
next_state = idle;
```

```
reset = 0;
i = 0;
end
else
begin
current_state = next_state;
case(current_state)
idle:
begin
if(opcode == 1)
begin
next_state = NOR;
C = 0;
ZF = 0;
CF = 0;
SF = 0;
end
else if(opcode == 2)
begin
next_state = ADD;
C = 0;
ZF = 0;
CF = 0;
SF = 0;
end
else if(opcode == 3)
begin
next_state = XNOR;
C = 0;
ZF = 0;
CF = 0;
SF = 0;
end
else if(opcode == 4)
begin
next_state = SUB;
C = 0;
ZF = 0;
CF = 0;
SF = 0;
end
end
NOR:
begin
C[i] = ~(A[i] | B[i]);
i = i + 1;
if(i == 4)
begin
i = 0;
reset = 1;
CF = 0;
if(C[3] == 1)
begin
SF = 1;
end
if(C == 0)
begin
ZF = 1;
end
end

end
ADD:
begin
ii = i+1;
{C_t[ii],C_t[i]} = (C_t[i]+A[i]+B[i]);
C[i] = C_t[i];
CF = A[i] & B[i];
i = i + 1;
if(i == 4)
begin
i = 0;
reset = 1;
if(C[3] == 1)
begin
SF = 1;
end
if(C == 0)
begin
ZF = 1;
end
end


end
XNOR:
begin
C[i] = ~(A[i] ^ B[i]);
i = i + 1;
if(i == 4)
begin
i = 0;
reset = 1;
CF = 0;
if(C[3] == 1)
begin
SF = 1;
end
if(C == 0)
begin
ZF = 1;
end
end


end
SUB:
begin
compliment_2s = -B;
C[i] = A[i] ^ compliment_2s[i];
C[i] = C[i] ^ F;
if(F == 0)
```

```
begin
F = A[i] & compliment_2s[i];
end
else
begin
F = A[i] | compliment_2s[i];
end

i = i + 1;
if(i == 4)
begin
CF = 0;
i = 0;
reset = 1;
if(C[3] == 1)
begin
SF = 1;
end
if(C == 0)
begin
ZF = 1;
end
end

end
endcase

end

end
endmodule
```