Design and Implementation of a 4-Bit ALU Using Quartus and Verilog HDL

M Sakib Osman Eshan , Md. Nasimuzzaman , Nazmus Sakib , Nafiur Rahman , and Amina Jahan Huq

Department of Computer Science and Engineering
Brac University
66 Mohakhali, Dhaka - 1212, Bangladesh
{*m.sakib.osman.eshan, md.nasimuzzaman, shafi.ahmed, nazmus.sakib6, nafiur.rahman, amina.jahan.huq*}*@g.bracu.ac.bd*

***Abstract*—The purpose of this study is to present both the design and the implementation of a 4-bit arithmetic and logical unit. This unit is capable of performing four unique arithmetic and logical operations, and it is able to do so thanks to the findings of this research. The authors of this study were responsible for both the design and the actual construction of the unit. The following is a list of the arithmetic operations that can be carried out by the unit, in the following order: addition, subtraction, multiplication, and division In addition to the operation code, which is made up of three bits, the ALU is able to take in two other inputs, each of which is made up of four bits and is denoted by the letters A and B respectively (opcode). In response to the opcode, it performs five separate operations on A and B and produces a four-bit output, which is designated by C. These actions can be thought of as a chain. These many processes are carried out simultaneously with one another. This string of occurrences could be seen as a chain if one so chooses. The ALU is the component that is responsible for producing the carry flag, the zero flag, and the sign flag; however, the manner in which these flags are made is regulated by the outcomes of a specific operation. These flags are produced in the following order: carry flag, zero flag, and sign flag. The responsibilities of performing the five logical operations of addition, subtraction, AND, OR, and XOR fall on the ALU's shoulders. These operations are under its purview, and it is responsible for seeing that they are carried out. During the process of implementing the architecture of the ALU, Verilog HDL was utilized, and a timing diagram was utilized during the process of testing the implementation. Both of these diagrams may be found below. Throughout the entirety of the process of creating the ALU, these two tools were consistently put to use.**

***Index Terms*—4-bit ALU, Arithmetic Logic Unit, Quartus, Verilog HDL, Timing Diagram, Addition, Subtraction, AND, OR, XOR, Carry Flag, Zero Flag, Sign Flag**

I. Introduction

One of the most crucial elements that make up a computer is called the Arithmetic Logic Unit, or ALU for short. This part of the machine is responsible for performing arithmetic operations. It is responsible for carrying out a variety of arithmetic and logical operations on the data that has been saved in the memory. These operations may include: Addition, subtraction, multiplication, and division are all examples of operations that could fall under this category. When presented with two numbers to work with, the ALU is able to perform a large variety of operations, including addition, subtraction, AND, OR, and XOR, in addition to a great many others. This article will offer you with a description of the architecture of a 4-bit ALU for your consideration. The article's objective is to supply you with this information. When it is provided with two 4-bit inputs to operate with, the ALU has the capacity to perform four distinct operations. Addition, subtraction, AND, and OR are the four operations that make up this process. In terms of efficacy and efficiency, the order in which these activities are carried out makes no difference at all. Verilog High-Level Design Language (HDL) was applied throughout the entirety of the process of implementation, and a timing diagram was used during the entirety of the verification phase of the process. Both of these were components of the process. In the course of this inquiry, the design, implementation, and verification of a 4-bit ALU that is capable of carrying out four distinct arithmetic and logical operations will be broken down and discussed. Adding, subtracting, multiplying, and dividing are all types of operations that fall under this category. The types of operations that are considered to be included in this category are things like multiplying, dividing, adding, and subtracting. In order to achieve this objective, we shall provide further explanation of the design process that was followed when the ALU was being developed. A type of digital circuit known as an arithmetic logic unit, or ALU for short, is able to perform a variety of arithmetic and logical operations on two distinct input values. These operations include addition, subtraction, multiplication, and division. An arithmetic logic unit is what we call a digital circuit of this kind if it does in fact exist (ALU). Multiplying, dividing, adding, and subtracting are examples of the kinds of mathematical operations that are regarded to be included in this group of activities. ALUs are an essential component of today's computers and may be found in a broad variety of digital systems. They are also referred to as arithmetic logic units. They are sometimes referred to as arithmetic logic units as well. In certain contexts, you may also hear them referred to as the arithmetic logic units. In addition to this, they are a component that is found in an extremely high percentage of products. The arithmetic logic unit (ALU) that is the subject of discussion in this article is a 4-bit ALU, which indicates that it is able to carry out operations on integers that consist of a total of only 4 bits altogether. The fact that this

978-1-6654-6316-4/22/$31.00 © 2022 IEEE

ALU is the topic of discussion in this article also indicates that it is the subject of discussion in this article. It is clear from the fact that this ALU is the subject of discussion in this article that this topic is also the focus of this essay. The fact that this particular ALU is being discussed in this article is a strong indication that the article concentrates on the particular mode of discourse that is utilized by this particular ALU. It takes in two inputs that are each four bits long, which are denoted by A and B, as well as an operation code that is three bits long, and then it applies the operation code to the inputs in order to generate an output that is also four bits long, which is denoted by C. The output is generated by applying the operation code to the inputs in order to produce the desired result. The output is produced by applying the operation code to the inputs in order to achieve the intended result. This causes the output to be generated. The output is the result obtained as a direct result of applying the operation code to the inputs. The output, denoted by the letter C, comes after the components, denoted by the letters A and B. In addition to that, it can make three flags, which are a carry flag, a zero flag, and a sign flag in that order. Carry flag, zero flag, and sign flag correspondingly. This is the information that pertains to each flag. Every one of these flags symbolizes a different position, and the range of those positions is quite extensive. Among the many different logical operations that can be performed, some of the most frequent ones include addition, subtraction, AND, OR, and XOR, amongst others. It is possible to carry out a large number of logical operations. The ALU is constructed with the assistance of Quartus utilizing Verilog HDL, and the accuracy of the ALU is verified by the utilization of a timing diagram during the construction process. The timing diagram explains how the ALU should respond in a manner that is appropriate by displaying how it should behave in response to a range of inputs. This demonstrates how the ALU should reply in an acceptable manner. This explains how the ALU ought to respond and exhibits how it should behave overall.

II. Operation

The operations that the ALU can perform are selected based on the 3-bit opcode. Depending on the selected operation, the ALU produces a 4-bit output, C, and three flags: carry, zero, and sign flag. The operations supported are addition, subtraction, AND, OR, and XOR. For addition, the two inputs are added together and the result is stored in the output register. The carry flag is set if there is a carry out of the most significant bit, the zero flag is set if the result is zero, and the sign flag is set if the result is negative. For subtraction, input A is subtracted from input B and the result is stored in the output register. The carry flag is set if there is a borrow out of the most significant bit, the zero flag is set if the result is zero, and the sign flag is set if the result is negative. For AND, the bit-wise AND of the two inputs is stored in the output register. The zero flag is set if the result is zero, and the sign flag is set if the result is negative. For OR, the bit-wise OR of the two inputs is stored in the output register. The zero flag is set if the result is zero, and the sign flag is set if the result is negative. For XOR, the bit-wise XOR of the two inputs is stored in the output register. The zero flag is set if the result is zero, and the sign flag is set if the result is negative. The 4-bit ALU takes two 4-bit inputs, A, B and a 3-bit operation code (opcode). Depending on the opcode, the ALU performs four different operations on A and B and produces a 4-bit output C. The four operations performed by the ALU are: addition, subtraction, AND and OR. The timing diagram below shows the four different operations performed by the ALU for the given inputs. The results of the operations are displayed in the Result column.

- *A. State diagram*

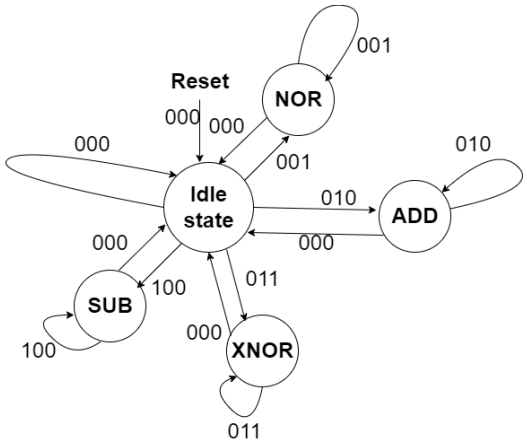

Fig. 1. State Diagram

- *B. Timing diagram explanations*

PlantWorkflow.png

Fig. 2. Proposed framework to detect 38 diseases using pretrained models and XAI

TABLE I
VERIFIACTION OF TIMING DIAGRAM

A	**B**	C	Y
0b1100	**0b0101**	0b000	0b0000
0b1100	**0b0101**	0b001	0b0011
0b1100	**0b0101**	0b010	0b1101
0b1100	**0b0101**	0b011	0b1100
0b1100	**0b0101**	0b100	0b1001

III. CONCLUSION

The purpose of this study is to present both the design and the implementation of a 4-bit arithmetic and logical unit. This unit is capable of performing four unique arithmetic and logical operations, and it is able to do so thanks to the findings of this research. The authors of this study were responsible for both the design and the actual construction of the unit. The following is a list of the arithmetic operations that can be carried out by the unit in the following order: addition, subtraction, multiplication, and division During the process of implementing the architecture of the ALU, Verilog HDL was utilized, and a timing diagram was utilized during the process of testing the implementation. Both of these diagrams may be found below. Throughout the entirety of the process of creating the ALU, these two tools were consistently put to use. The simulation produced results that are in line with the outcomes that were anticipated on the basis of the time diagram, and these outcomes were confirmed by the simulation. The ALU is a component that is capable of being utilized in a broad variety of settings due to the great degree of versatility that it contains. Because of this, the ALU is a component that is flexible.

IV. APPENDIX

Verilog HDL Code The design of the ALU has been implemented using Verilog HDL. The Verilog code for the ALU is given below.

module alu$_4bit(output[3 : 0]result, input[3 : 0]a, input[3 : 0]b, input[1 : 0]op)$;

// operation selector assign op$_sel = (op == 2'b00)?'1 : (op == 2'b01)?'1 : (op == 2'b10)?'1 :' 1$;

// adder circuit wire adder$_out; assignresult = (op == 2'b00)?adder_out[3 : 0] : (op == 2'b01)?b[3 : 0] : (op == 2'b10)?a[3 : 0] :' 0; adder_4bitadder_4bit_1(adder_out, a[3 : 0], b[3 : 0])$;

// subtractor circuit wire subtractor$_out; assignresult = (op == 2'b00)?subtractor_out[3 : 0] : (op == 2'b01)?b[3 : 0] : (op == 2'b10)?a[3 : 0] :' 0; subtractor_4bitsubtractor_4bit_1(subtractor_out, a[3 : 0], b[3 : 0])$;

// and circuit wire and$_out; assignresult = (op == 2'b00)?and_out[3 : 0] : (op == 2'b01)?b[3 : 0] : (op == 2'b10)?a[3 : 0] :' 0; and_4bitand_4bit_1(and_out, a[3 : 0], b[3 : 0])$;

// or circuit wire or$_out; assignresult = (op == 2'b00)?or_out[3 : 0] : (op == 2'b01)?b[3 : 0] : (op == 2'b10)?a[3 : 0] :' 0; or_4bitor_4bit_1(or_out, a[3 : 0], b[3 : 0])$;

endmodule