ELKHA : Jurnal Teknik Elektro, Vol. 15 No.2, October 2023, pp. 98 - 105
ISSN: 1858-1463 (print), 2580-6807 (online)

Design and Implementation of 12-Bit Arithmetic Logic Unit with 8 Operation Codes to Field Programmable Gate Array

Arwin Datumaya Wahyudi Sumari$^{1*)}$, Sukriya Hijriana$^{2)}$, and Denny Dermawan$^{3)}$

$^{1, 2, 3)}$ Electrical Engineering Study Program, Faculty of Industrial Technology, Adisutjipto Institute of Aerospace Technology, Indonesia
$^{1, 2, 3)}$ Master of Applied Electrical Engineering Study Program, Department of Electrical Engineering, Politeknik Negeri Malang, Indonesia
Corresponding Email: $^{*)}$ arwin@itda.ac.id

Abstract – **Digital system has been a part of human life since the invention of the computer with a microprocessor as the central brain. At the heart of a processor is an Arithmetic Logic Unit (ALU) that handles arithmetic and logic operations. The need for high-speed computation to handle complex computations demands microprocessors with higher performance. The existing 4-opcode 8-bit ALU cannot handle multiplication operations, so a solution is needed. In this research, while raising the appeal of beginners, a 12-bit ALU with eight operation codes (opcode) was designed and implemented in Xilinx's Field Programmable Gate Array using a schematic diagram approach through logic gates. The designed and implemented ALU provides addition, subtraction, multiplication, square, AND, OR, NAND, and XOR operations. The multiplication operation was tested by performing the computation to provided datasets to obtain the distance travelled by ten military aircraft based on their maximum speed and air travel duration to ensure its performance. The computation performance comparison with an 8-bit ALU with four opcodes was also done. The computation was done for air travel between 10 to 60 minutes with a 10-minute difference. It was found that the 12-bit ALU with eight opcodes outperformed its contender with computation differences between 130.815 ns and 1,468.214 ns. This high performance is supported by the multiply operation that does repeated addition at one time. Based on this finding, the 8-opcode 12-bit ALU is more efficient in the context of computation time, with consistent accuracy. Moreover, the computation time required to calculate military aircraft data with different maximum speeds and air travel duration is only 119.501 ns.**

Keywords: Arithmetic logic unit, field programmable gate array, microprocessor, military aircraft, operation code

I. INTRODUCTION

The invention of the transistor in the 1940s has led to the development of various digital systems, including computer-based systems. As the human helper in delivering solutions for complex computing tasks, the microprocessor plays vital roles as the instruction executor, controller, and central to digital data processing. Hence, the microprocessor is said to be a Central Processing Unit (CPU). Generally, a microprocessor consists of three central units: Control Unit (CU), Arithmetic Logic Unit (ALU), and register. At the heart of it, ALU is the vital part of various gadgets [1]–[3] that handle arithmetic and logical operations, such as addition, subtraction, multiply, square, AND, OR, NAND, and XOR. It is the most used element of the microprocessor because it is the only one that does mathematical and logical operations required by computing-based systems [2], [4]. The ALU is an indispensable part of the CPU, the central core of all digital systems that certainly use a microprocessor as their primary computing engine [5], [6]. The better the ALU performance, the better the CPU performance, which has an impact on the better the performance of the digital system [7].

The need for high-speed computation to handle complex computations demands higher-performance microprocessors. This means it needs ALU with better performance. The challenge is how to design ALU with that demand. Another challenge is how to attract electrical engineering students to have an interest in this field. One of the simple approaches is to use a schematic diagram followed by its implementation to Field Programmable Gate Array (FPGA). The main objective of this research is to arouse the appeal of beginners by showing easy ways to design an ALU using available tools and implement them through FPGAs.

It is a programmed digital Integrated Circuit (IC) that consists of various logical blocks that are easy to reconfigure or allow the designers to change the design according to the need at any time. It requires a small amount of power, supports simultaneous operations, and is easy to learn [8], [9]. It is the primary reason it is often used to implement digital circuits. Moreover, the use of FPGA stimulates creativity. It facilitates the students to think, explore, and learn [10]–[13] how to design and implement digital circuits from a simple one to a more complex one.

Some works in designing and implementing microprocessor's ALU have been done, such as ALU with two logical operations, namely AND and OR [14], and 8-bit ALU with four operation codes (opcode) that can carry out arithmetic operation's addition and subtraction, and

Manuscript received 2023-03-31; revised 2023-09-17 ; accepted 2023-09-19

This work is licensed under a Creative Commons Attribution 4.0 License For more information, see https://creativecommons.org/licenses/by-nc-sa/4.0/

logical operation's AND and OR [15]. However, the primary ALU's operation is the addition [16]. For example, a multiply operation in essential is a repetitive addition operation, such as $2 * 3 = 2 + 2 + 2$. Therefore, it is not an efficient operation faced with increasing data. Furthermore, referring to [15], the addition operation offered by 4-opcode 8-bit ALU will not be efficient and will take a long time. On the other hand, the use of FPGA for designing and simulating the ALU has been done for developing 4-bit [4], 32-bit [9], and 64-bit [2], [3].

Based on the above hypothesis, we designed and implemented 8-opcode 12-bit ALU by increasing the number of operations to cover more arithmetic and logical operations. Furthermore, to ensure that our ALU can perform better, we employed it to calculate the distance traveled by military aircraft based on their maximum speed and travel time. The computation was done using multiply operation compared to addition operation done by 4-opcode 8-bit ALU and by 8-opcode 12-bit ALU. The rest of the article will present our research method, results, and analysis and be wrapped up with conclusions and further works.

II. Methodology

A. Research Stages

The research was started by carrying out the literature study followed by an 8-opcode 12-bit ALU design using a schematic diagram approach. We used Xilinx ISE 9.2i software for this purpose and did the test and data collection for analysis. The research steps are depicted in Figure 1.

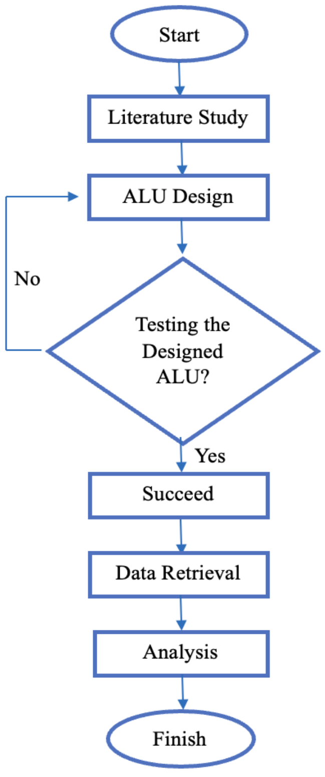

Figure 1. Research stages flow.

B. Data Collection Method

For testing the ALU, two datasets were created. As presented in Table 1, one dataset contains random data used to test whether the eight operations are well carried out and result in the correct outputs. Another dataset contains actual data that consists of the name and type of military aircraft and each maximum speed in kilometers/hour (km/h) and its conversion to kilometer/minute (km/min).

ALU will compute the distance traveled by each aircraft, that is, the maximum speed time and the traveling time. For this purpose, we set up the traveling time from 10 to 60 minutes with 10 minutes difference between one another. The actual dataset for ALU testing is shown in Table 2, where the aircraft type and maximum speed are taken from [17]. As additional information, the F-16 Fighting Falcon, JF-17 Thunder, and Dassault Rafale are multi-role fighter aircraft manufactured by the United States of America, the People's Republic of China, and France. At the same time, the Su-33 is an air superiority aircraft manufactured by the Russian Federation. The F/A-18 Hornet is a multi-role and carrier-capable landing aircraft manufactured by the United States of America.

Table 1. Random Dataset for 12-Bit ALU Testing

No	Input		Function	Output
	A	B		
1	1089	1442	Add	2531
2	514	321	Subtract	193
3	2194	324	Multiply	710856
4	3712	-	Square	13778944
5	100110000110	101100111010	AND	100100000010
6	001000110100	001011001011	NAND	110111111111
7	010110101011	001100011001	OR	011110111011
8	110011010111	101011100011	XOR	011000110100

Table 2. Real Dataset for 12-Bit ALU Testing.

Aircraft Name	Type	Maximum Speed	
		(km/h)	(km/min)
F-16	Fighter	2120	35
Su-33	Fighter	2500	42
JF-17	Fighter	1960	33
Rafale	Fighter	2225	37
F/A-18	Fighter	1915	32
Super Tucano	Attack	590	10
C-130	Cargo	600	10
A400M	Cargo	780	13
Super Puma	Helicopter	276	5
Colibri	Helicopter	280	5

On the other hand, the EMB 314 Super Tucano is a turbo-propeller light-attack aircraft that is used to fight the guerillas manufactured by Brazil. There are two military cargo aircraft, namely, C-130 Hercules and Airbus A400M, that the United States of America and Spain manufacture. The last two aircraft, NAS-332 Super Puma and Eurocopter EC120 Colibri, were manufactured in Indonesia and France. NAS-332 Super Puma is a heavy-load carrier helicopter, while EC120 colibri is a light-utility helicopter.

C. *Designing the ALU Operations*

It needs eight different opcodes to develop a 12-bit ALU that can do eight arithmetic and logical operations. The developed ALU can do arithmetic operations, that is, addition, subtraction, multiplication, and square, and logical operations, that is, AND, NAND, OR, and XOR functions. Xilinx 9.2i was used to implement and simulate our designed ALU. The two most essential subsystems for our designed ALU are the multiplier that does the multiplication task and the multiplexer that does the task selection function. Their designs are depicted in Figure 2 and Figure 3. Table 3 gives a short explanation regarding the operations that the designed ALU will carry out.

Table 3. A Short Explanation of Each ALU Operation

Operation	Explanation
Addition	Arithmetic operations are in the form of add operations where the value of one variable (A) is added with the value of another variable (B) to obtain a new value (C). The form of operation is **C = A + B**
Subtraction	Arithmetic operations in the form of subtract operations where the value of one variable (A) is subtracted with the value of another variable (B) to obtain a new value (C). The form of operation is **C = A – B**
Multiplication	Arithmetic operations are in the form of multiply operations where the value of one variable (A) is multiplied by the value of another variable (B) to obtain a new value (C). The form of operation is **C = A x B**
Square	Arithmetic operations are square operations where the value of one variable (A) is squared with itself to obtain a new value (C). The form of operation is **C =** $\mathbf{A^2}$
AND	The logical operation that compares the values of inputs to produce output follows the rules of the Boolean Rule, where if both inputs (A and B) are '1', the output © is '1'. In contrast, combining other inputs will produce an output of '0'. The form of operation is **C = A AND B**
NAND	The output of NAND logical operations is the opposite of that of AND logical operations. The form of operation is **C = A NAND B.**
OR	The logical operation that compares the values of inputs to produce output follows the rules of the Boolean Rule, where if both inputs (A and B) are '0', the output (C) is '0'. In contrast, combining other inputs will produce an output of '1'. The form of operation is **C = A OR B**
XOR	The logical operation that compares the values of inputs to produce output follows the rules of the Boolean Rule, where if both inputs (A and B) are '0' or '1', the output (C) is '0'. In contrast, combining other inputs will produce an output of '1'. The form of operation is **C = A XOR B**

1. *12-Bit Addition and Subtraction Operations*

Adder plays an essential role in ALU computation [16], [18], [19]. The basic adder schematic is a full-adder circuit. In our design, the ALU is equipped with a 12-bit adder and subtractor with two inputs, A and B, $control$ [20], and one input $cout$ (CO). The circuit will function as an adder if $the\ control$ is given logical 0 input. If it is given logical 1 input, then it functions as a subtractor. In the schematic, the XOR gate is used to reverse the logic at input B to become $(-B)$ to change the function of the circuit from adder to subtractor.

2. *12-Bit Multiplication Operation*

The 12-bit multiplier will have two 12-bit inputs, A and B, and one 24-bit output, and its schematic is developed based on the multiplication operation algorithm [21]. The AND gate is tasked to carry out the multiplication to each bit, and the 12-bit adder is tasked to sum the result of the multiplication [22]–[24]. The block diagram and the schematic of the 12-bit multiplier are depicted in Figure 2 and Figure 3.

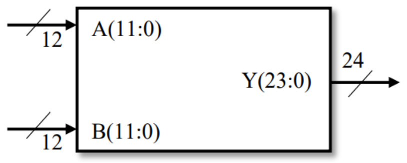

Figure 2. The 12-bit multiplier block diagram.

3. *12-Bit Square*

The schematic of the square operation is the same as that of the multiplication operation. It uses the AND gate to carry out each bit multiplication, and the 12-bit adder will sum the multiplication results. The distinct difference between multiplication and square is that the input to the square operation will be multiplied by its input.

4. *12-Bit AND, NAND, OR, and XOR Operations*

For each function, the design will consist of 12 AND gates [11]–[12] for AND operation, 12 NAND gates to support NAND operation, 12 OR gates to perform OR operation, and 12 XOR gates to carry out XOR operation.

5. *Multiplexer*

In its operation, ALU will select an operation to be carried out by using a device called a multiplexer. It will select an operation based on the control line or opcode [19]–[20]. It can be designed based on the Boolean algebra function [11] and has 24-bit data wide. During operation, the multiplexer will receive eight inputs, namely addition, subtraction, multiplication, square, AND, NAND, OR, and XOR, but only deliver one output based on the inputted opcode. The block diagram and the schematic of the multiplexer are shown in Figure 4 and Figure 5.

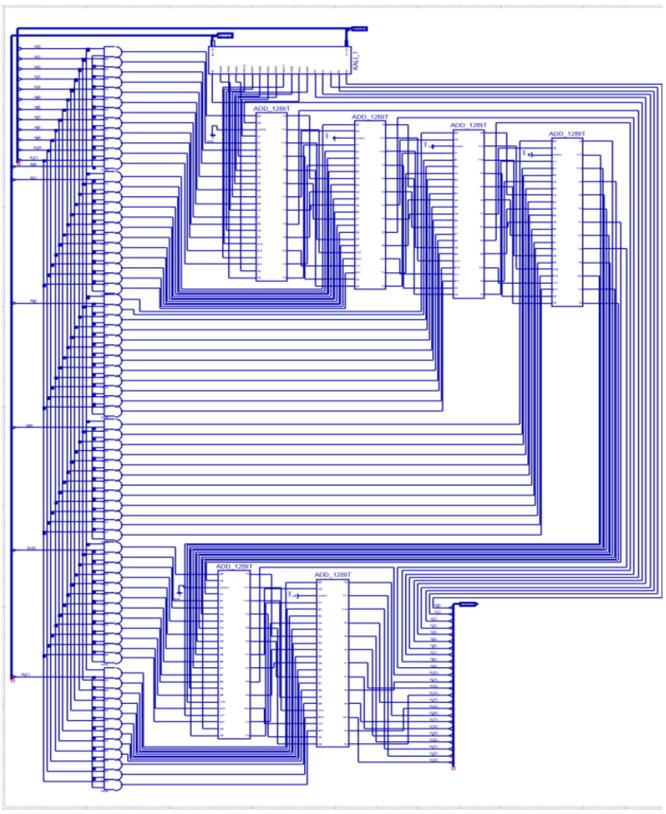

Figure 3. The schematic of the 12-bit multiplier. It is taken initially from the Xilinx 9.2i display.

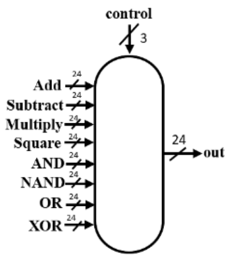

Figure 4. The multiplexer block diagram.

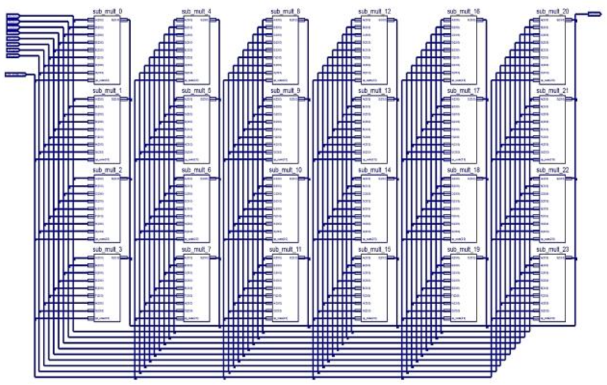

Figure 5. The schematic of the multiplexer. It is taken initially from the Xilinx 9.2i display.

D. *Designing the ALU*

12-bit ALU needs two 12-bit inputs and one 24-bit output with three control inputs or opcodes to select a particular operation from 8 available ones: addition, subtraction, multiplication, square, AND, NAND, OR, and XOR. The block diagram view of the designed 12-bit ALU and its schematic is depicted in Figure 6 and Figure 7. As shown, besides two inputs, A and B, there is also a 3-bit ALU control input that consists of OP_CODE0, OP_CODE1, and OP_CODE2. The control functions select one of eight functions supported by the ALU. Each function has a different combination of opcodes, as presented in Table 4, where the multiplexer uses these combinations to output a particular function or operation.

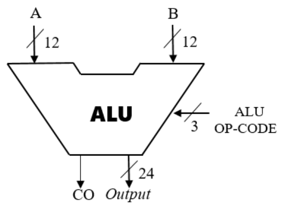

Figure 6. The 12-bit ALU block diagram.

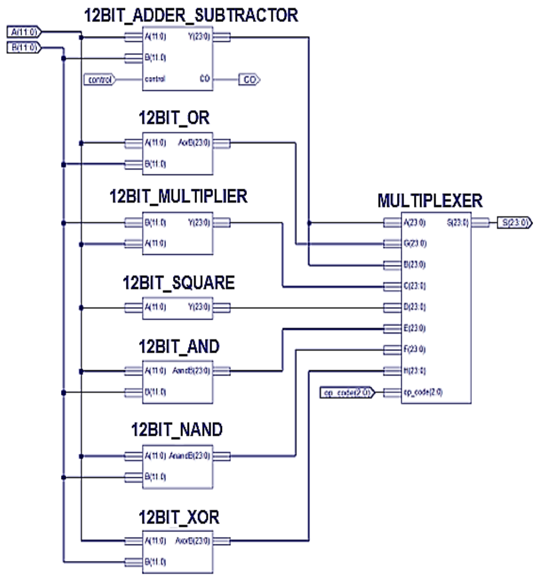

Figure 7. The schematic of 8-opcode 12-bit ALU.

Table 4. The Truth Table for 8-Opcode 12-bit ALU

Input			Function/ Operation
OPCODE2	OPCODE1	OPCODE0	
0	0	0	**A + B**
0	0	1	**A – B**
0	1	0	**A x B**
0	1	1	**A^2**
1	0	0	**A AND B**
1	0	1	**A NAND B**
1	1	0	**A OR B**
1	1	1	**A XOR B**

III. RESULTS AND DISCUSSION

To ensure that the developed 8-opcode 12-bit ALU can carry out all eight operations, we did two tests using random and real datasets, as mentioned in Section II. The ALU inputs are 12-bit wide each, where input $A(11:0)$

comprises $A0 \dots A11$, input $B(11{:}0)$ comprises $B0 \dots B11$, while output $Y(23{:}0)$ comprises $Y0 \dots Y23$, and 1 bit CO. The results of the simulation of ALU computation are as follows.

A. Simulation Results for Random Dataset – Operational Test

Firstly, the 8-opcode 12-bit ALU was tested using a random dataset before a real one. As planned, the 12-bit ALU will have four arithmetic and four logical operations. For this purpose, a random dataset in Table 1 was used to test the operations and select the correct opcode to perform such operations. The arithmetic operation's addition and subtraction results are presented in Figure 8, while multiplication and square operations are given in Figure 9. On the other hand, the results of logical operations, namely, AND, NAND, OR, and XOR, are presented in Figure 10 to Figure 13. Numbers in yellow squares are the random dataset, as shown in Table 1.

As presented in Figure 6 to Figure 11, the 12-bit ALU got correct outputs according to the selected operation based on the inputted opcode to the multiplexer. Opcode 000 results in $A + B$, opcode 001 results in $A - B$, opcode 010 results in $A * B$, opcode 011 results in A^2, opcode 100 results in $A\ AND\ B$, opcode 101 results in $A\ NAND\ B$, opcode 110 results in $A\ OR\ B$, and opcode 111 results in $A\ XOR\ B$. The ALU output is set to 24 bits to accommodate the results of multiplication and square. Therefore, data width for other operations must be the same, that is, 24 bits, and this can be done by adding 12 bits of logical '0' after the Most Significant Bit (MSB).

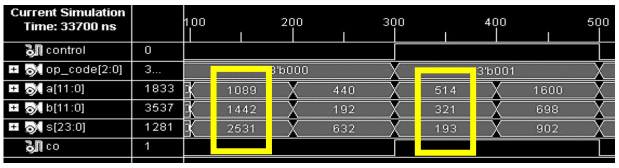

Figure 8. Addition and subtraction operations' results.

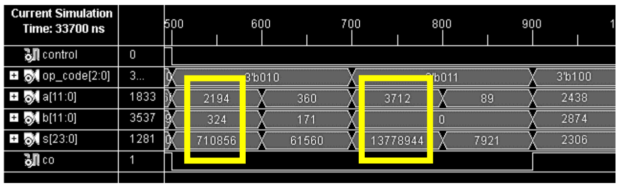

Figure 9. Multiplication and square operations' results.

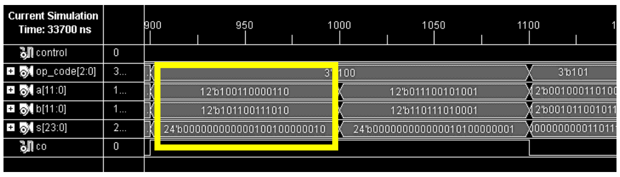

Figure 10. AND operation's result.

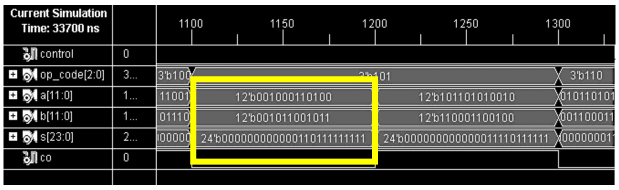

Figure 11. NAND operation's result.

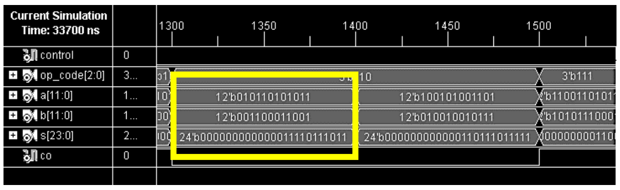

Figure 12. OR operation's result.

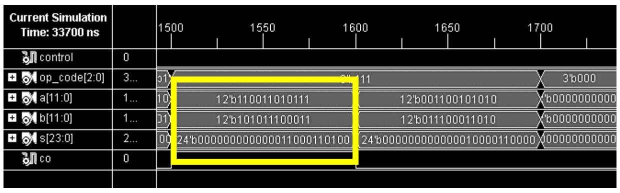

Figure 13. XOR operation's result.

B. Testing Results for Real Dataset – Computation Test

Section A above has shown that the developed ALU succeeded in the operational tests, and each opcode combination in Table 3 has resulted in the correct operation as planned. In this section, the test objective is to ensure that the ALU can deliver the correct result using multiplication operations.

For this purpose, it is tasked to obtain the distance traveled by some military aircraft with a dataset, as shown in Table 2. The inputs are the aircraft's maximum speed, A, and air travel, B, where the traveling time is set up from 10 to 60 minutes with a 10-minute difference between one another. The traveled distance is calculated using Equation (1). Then, we compared it with manual computation using a spreadsheet application to check the results. Some computation results are presented in Table 5 to Table 7. The simulation results are depicted in Figure 14 to Figure 16.

$$s = v \times t \tag{1}$$

where:

s is the traveled distance (km)

v is the maximum speed (km/min)

t is time (min)

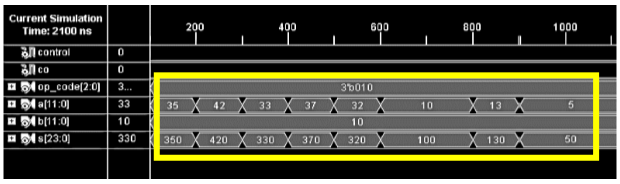

Figure 14. The computation results for 10-minute airtime.

Table 5. Computation Results for 10-Minute Airtime

Aircraft Type	Speed (km/min)	Manual (km)	ALU (km)	Discrepancy
F-16	35	350	350	0
Su-33	42	420	420	0
JF-17	33	330	330	0
Rafale	37	370	370	0
F/A-18	32	320	320	0
Super Tucano	10	100	100	0
C-130	10	100	100	0
A400M	13	130	130	0
Super Puma	5	50	50	0
Colibri	5	50	50	0

Table 6. Computation Results for 30-Minute Airtime

Aircraft Type	Speed (km/ min)	Manual (km)	ALU (km)	Discre-pancy
F-16	35	1050	1050	0
Su-33	42	1260	1260	0
JF-17	33	990	990	0
Rafale	37	1110	1110	0
F/A-18	32	960	960	0
Super Tucano	10	300	300	0
C-130	10	300	300	0
A400M	13	390	390	0
Super Puma	5	150	150	0
Colibri	5	150	150	0

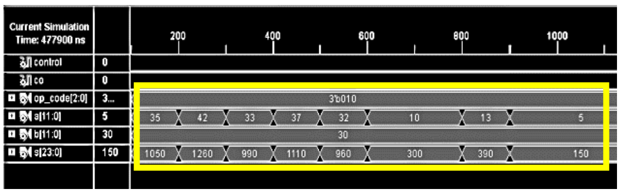

Figure 15. The computation results for 30-minute airtime.

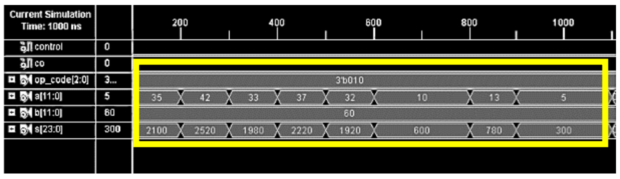

Figure 16. The computation results for 60-minute airtime.

From the results above, there is no discrepancy result between the ALU computation and spreadsheet one. Therefore, our developed ALU succeeded in the computation tests. Moreover, it shows outstanding performance with a computation accuracy of 100%.

Table 7. Computation Results for 60-Minute Airtime

Aircraft Type	Speed (km/min)	Manual (km)	ALU (km)	Discrepancy
F-16	35	2100	2100	0
Su-33	42	2520	2520	0
JF-17	33	1980	1980	0
Rafale	37	2220	2220	0
F/A-18	32	1920	1920	0
Super Tucano	10	600	600	0
C-130	10	600	600	0
A400M	13	780	780	0
Super Puma	5	300	300	0
Colibri	5	3000	3000	0

C. Computation Time Comparison between Multiplication and Addition Operation

The development of 8-opcode 12-bit ALU is to enhance the one developed by [15]. Therefore, the crucial matter of the test is to compare the performance of the two ALUs to show that our developed one can perform better in computation time. 4-opcode 8-bit ALU has only four opcodes and is not supported with multiplication operations. Therefore, this operation is carried out in repetitive addition-like operations. Hypothetically, our designed ALU, powered by the multiplication function through the multiplier, will show much better performance. The test was done successfully, the results are shown in Table 8.

Table 8. Computation Time Comparison

Air-time (min)	4-opcode 8-bit ALU (ns)	8-opcode 12-bit ALU (ns)	Discre-pancy (ns)	Computation Time Increase (%)
10	250.316	119.501	130.815	209.468
20	518.337	119.501	398.836	433.751
30	786.357	119.501	666.856	658.034
40	1054.376	119.501	934.875	882.316
45	1188.385	119.501	1068.884	994.456
50	1322.395	119.501	1202.894	1106.597
60	1587.714	119.501	1468.213	1328.620

Even though multiplication is essential in a repetitive addition, in the view of ALU, they are very different operations. As shown in Table 7, using the same dataset, the addition operation carried out by 4-opcode 8-bit ALU needs to be performed more than the multiplication operation done by 8-opcode 12-bit ALU. The longer the airtime, the more significant the time discrepancy is. Whatever the value of the airtime, the computation time required by the 8-opcode 12-bit ALU stays at a consistent value, that is, 119.501 nanoseconds (ns).

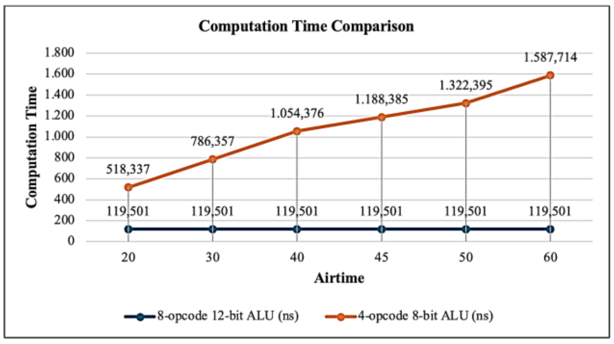

Figure 17. Significant increase in computation time of 4-opcode 8-bit ALU as the value of airtime increases.

On the other hand, for 4-opcode 8-bit ALU, there is a significant increase in computation time as the airtime increases. For example, the longest computation time for 4-opcode 8-bit ALU is obtained for 60' airtime with the value of 1,587.714 ns, which is 1,328.620% longer than that of 8-opcode 12-bit ALU. It also explains that the longer the airtime, the longer the time is taken for the repetitive addition operation. Therefore, it concludes that multiplication operation outperforms repetitive addition, as depicted in Figure 17. It proves that our designed 8-opcode 12-bit ALU outperforms the 4-opcode 8-bit one with a significant achievement. This finding shows that having an ALU equipped with multiplication operations can speed up the computation process, reducing the computation time.

IV. Conclusion

Based on the results and discussion, we get some conclusions. Our developed 8-opcode 12-bit ALU has passed the operational and computation tests with outstanding performance. Each combination of opcodes has succeeded in instructing the ALU to carry out the correct operation, while in computation, it delivers much better performance than the 4-opcode 12-bit ALU with more than 1,300%. Whatever the airtime values, our ALU shows a stable and consistent computation time much lower than its contender. Moreover, it is proved that multiplication operation performs much better than repetitive addition, making it an essential operation in the ALU, which handles large data multiplication. Our ALU also achieved 100% accuracy when carrying out the computation to find the distance traveled by various military aircraft. We concluded that an 8-opcode 12-bit ALU equipped with multiplication operation can speed up the computation process and reduce the computation time needed for high-speed computing-based systems. A schematic diagram delivers a straightforward approach to designing ALU. In tune with that, Xilinx 9.2i software is a helpful electronics circuit designing software that provides such an approach and can help students design other electronic circuits. The plan for the future is to design and implement an 8-opcode 16-bit ALU, followed by comparing its performance with the ALU implemented in this study.

References

[1] S. M. Swamynathan and V. Banumathi, “Design and analysis of FPGA based 32 bit ALU using reversible gates,” *Proceedings - 2017 IEEE International Conference on Electrical, Instrumentation and Communication Engineering, ICEICE 2017*, vol. 2017-Decem, no. April 2017, pp. 1–4, 2017.

[2] V. Sharma, K. Nayanam, S. Shukla, and N. Bhatia, “Design of Energy Efficient ALU on FPGA,” *International Journal of Advances in Engineering and Management (IJAEM)*, vol. 2, no. 2, pp. 5–10, 2008.

[3] N. S. Bedir and F. Kaçar, “Design and Simulation of 64 Bit FPGA based Arithmetic Logic Unit,” *Electrica*, vol. 19, no. 2, pp. 158–165, 2019.

[4] A. K. Panigrahi, S. Patra, M. Agrawal, and S. Satapathy, “Design and Implementation of a high-speed 4bit ALU using BASYS3 FPGA Board,” in *2019 Innovations in Power and Advanced Computing Technologies (i-PACT)*, 2019, pp. 1–6.

[5] R. Aliabadian, M. Golsorkhtabaramiri, S. R. Heikalabad, and M. K. Sohrabi, “Design of an ultra-high-speed coplanar QCA reversible ALU with a novel coplanar reversible full adder based on MTSG,” *The European Physical Journal Plus*, vol. 138, no. 5, p. 481, May 2023.

[6] M. Alharbi, G. Edwards, and R. Stocker, “Reversible Quantum-Dot Cellular Automata-Based Arithmetic Logic Unit,” *Nanomaterials*, vol. 13, no. 17, Sep. 2023.

[7] K. V. B. V. Rayudu, D. R. Jahagirdar, and P. S. Rao, “Modern Design and Testing of High-Speed Vedic ALU Controller using Vedic Algorithms,” *Journal of The Institution of Engineers (India): Series B*, vol. 104, no. 1, pp. 221–230, Feb. 2023.

[8] H. Wang and X. Chen, “Development and Optimization Design of Digital Logic device based on FPGA,” *J Phys Conf Ser*, vol. 1345, no. 6, 2019.

[9] B. Özkilbaç, “Implementation and Design of 32 Bit Floating-Point ALU on a Hybrid FPGA-ARM Platform,” *Brilliant Engineering*, vol. 1, no. 1, pp. 26–32, Dec. 2019.

[10] A. Borodzhieva, I. Stoev, and V. Mutkov, "Application of Active Learning Methods in the Course 'Digital Electronics' in the Topic Digital Comparators Using FPGA Design," *SIITME 2019 - 2019 IEEE 25th International Symposium for Design and Technology in Electronic Packaging, Proceedings*, no. October 2019, pp. 160–163, 2019.

[11] A. N. Borodzhieva, I. I. Stoev, and V. A. Mutkov, “FPGA implementation of boolean functions using multiplexers,” in *2019 28th International Scientific Conference Electronics, ET 2019 - Proceedings*, 2019.

[12] A. Borodzhieva, I. Stoev, and V. Mutkov, “FPGA Implementation of Boolean Functions Using Decoders and Logic Gates,” *SIITME 2019 - 2019 IEEE 25th International Symposium for Design and Technology in Electronic Packaging, Proceedings*, no. October, pp. 164–167, 2019.

[13] F. Suryawan, “A project-based approach to FPGA-aided teaching of digital systems,” *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, vol. 2017-Decem, no. September, pp. 19–21, 2017.

[14] A. Rani and N. Grover, “An enhanced FPGA based asynchronous microprocessor design using VIVADO and ISIM,” *Bulletin of Electrical Engineering and Informatics*, vol. 7, no. 2, pp. 199–208, 2018.

[15] D. Dermawan, M. A. M. Putra, C. B. Waluyo, and B. Sudibya, “Rancang Bangun Arithmetic Logic Unit 8 Bit Pada Spartan 2 Field Programmable Gate Array,” *Conference SENATIK STT Adisutjipto Yogyakarta*, vol. 6, pp. 185–198, 2020.

[16] S. K. Pattnaik, U. Nanda, D. Nayak, S. R. Mohapatra, A. B. Nayak, and A. Mallick, “Design and implementation of different types of full adders in ALU and leakage minimization,” *Proceedings - International Conference on Trends in Electronics and Informatics, ICEI 2017*, vol. 2018-Janua, pp. 924–927, 2018.

[17] Military Factory - Global Defense Reference, “Militaryfactory.com”,2023.https://www.militaryfactory.com/

[18] M. S. Ali, “Cascaded ripple carry adder based SRCSA for efficient FIR filter,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 9, no. 2, pp. 253–256, 2018.

[19] N. S. Bedir and F. Kaçar, “Design and simulation of 64 bit FPGA based arithmetic logic unit,” *Electrica*, vol. 19, no. 2, pp. 158–165, 2019.

[20] A. N. Borodzhieva, I. I. Stoev, I. D. Tsvetkova, S. L. Zaharieva, and V. A. Mutkov, “Computer-Based Education in the Course ‘Digital Electronics’ Teaching the Topic ‘Adders-Subtractors,’” in *2020 43rd International Convention on Information, Communication and Electronic Technology, MIPRO 2020 - Proceedings*, 2020.

[21] A. Borodzhieva, I. Tsvetkova, S. Zaharieva, D. Dimitrov, and V. Mutkov, “Project-based learning approach applied in the course ‘digital electronics’ for studying the topic ‘binary multipliers,’” in *ACM International Conference Proceeding Series*, 2021.

[22] F. Nasser and I. Hashim, “Power Optimization of Binary Multiplier Based on FPGA,” *Engineering and Technology Journal*, 2021.

[23] J. L. Imana, “Low-Delay FPGA-Based Implementation of Finite Field Multipliers,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2021.

[24] M. P. Véstias and H. C. Neto, “Decimal multiplication in FPGA with a novel decimal adder/subtractor," *Algorithms*. 2021.

[25] P. K. Rai, S. Srivastava, and A. Johri, "Design, Layout and Simulation of 8-bit Arithmetic and Logic Circuits Pad frame using C5 Process for deep submicron CMOS," in *2018 International Conference on Advanced Computation and Telecommunication, ICACAT 2018*, 2018.

[26] J. R. Shinde and S. J. Shinde, “An optimization design strategy for arithmetic logic unit,” *Universal Journal of Electrical and Electronic Engineering*, 2019.

[27] N. Yadav and P. Kumari, “Design of ALU using dual-mode logic with optimized power and speed,” *IMPACT 2017 - International Conference on Multimedia, Signal Processing and Communication Technologies*, pp. 41–45, 2018.