Design and Implementation of a 3-bit ALU with Integrated 7-Segment Display on FPGA

A Hemanth Kumar[1]

[1]*Dept of E.C.E, NIELIT Calicut, Kerala.*

Abstract: This paper describes the design and implementation of a 3-bit Arithmetic Logic Unit (ALU) with integrated 7-segment display output, deployed on an FPGA platform. The objective of this project is to efficiently perform and display basic arithmetic and logic operations, including addition, subtraction, AND, OR, XOR, and NOT, with an FPGA-based ALU. A control signal selects the operation, and the ALU's 3-bit result is decoded and displayed on a 7-segment interface for clear output visualization. The ALU design is coded in Verilog and includes logic to manage carry and overflow in arithmetic functions. A display decoder module maps the ALU's binary output to the correct LED segments on the 7-segment display, allowing intuitive real-time readout of operation results. The system was synthesized, simulated, and tested on an Artix-7 FPGA using the Xilinx Vivado development environment, achieving accurate and expected functionality across all operations. This work showcases the effectiveness of FPGA technology in integrating computation and display within a compact digital system, with potential applications in both educational and embedded system design settings where real-time processing and display are essential.

Keywords: FPGA, ALU, 7-segment display, Verilog, Digital system design, Embedded computing.

1. INTRODUCTION: Arithmetic Logic Units (ALUs) are essential components in digital systems, responsible for executing key arithmetic and logic operations that form the basis of processors and embedded systems. With growing demands for flexible and efficient digital circuits, FPGAs (Field-Programmable Gate Arrays) are increasingly favoured for ALU implementations due to their high-speed processing, reconfigurability, and seamless integration potential. This paper details the design and implementation of a 3-bit ALU, capable of performing addition, subtraction, AND, OR, XOR, and NOT operations, with real-time results displayed on an integrated 7-segment display. Designed in Verilog and deployed on an Artix-7 FPGA using Xilinx Vivado, this ALU leverages FPGA's dual capability for both computation and immediate output visualization. The system provides valuable insights into FPGA utility for compact, efficient digital modules, demonstrating applicability in educational settings and embedded systems that require real-time operational verification. This implementation serves as a practical reference for digital design education and offers a robust foundation for developing real-time processing units in resource-sensitive applications.

2. LITERATURE SURVEY: The design and implementation of a 3-bit ALU with an integrated 7-segment display on FPGA is a well-established area of research with practical applications in both academic and industrial settings. ALUs are essential components in digital systems, and FPGA platforms offer the flexibility to implement and customize these designs efficiently. The integration of a 7-segment display enhances user interaction, providing an intuitive method for displaying results. Research by Ranjani and Krishnaiah (2014) on ALU design and Roush and Burns (2005) on display integration provides crucial insights into the design principles, implementation techniques, and challenges of these systems.

These base papers form the foundation for the development of efficient and scalable ALU designs that can be seamlessly integrated with display systems for practical applications. The combination of FPGA implementation, ALU functionality, and display integration has significant potential for educational tools, embedded systems, and real-time computational applications.

3. DESIGN SYSTEM:

3.1. Design of Top Level (RTL) Verilog Module of 3-Bit Arithmetic Logical UNIT (ALU):

High-level design methodologies help manage complexity and reduce the design cycle. These models simplify the description and evaluation of complex systems, making the design process faster and more efficient. In RTL (Register Transfer Level) design, all registers and the combinational logic between them are specified clearly. Registers can be described either explicitly or implicitly, while the combinational logic is defined through logical equations or Verilog statements.

Key benefits:

- **Managing Complexity**: Fewer lines of code reduce error and improve productivity.
- **Increased Design Reuse**: Modular components can be reused across different designs, saving time and effort.
- **Improved Verification**: Faster simulations help identify issues more quickly, ensuring more reliable designs.

.

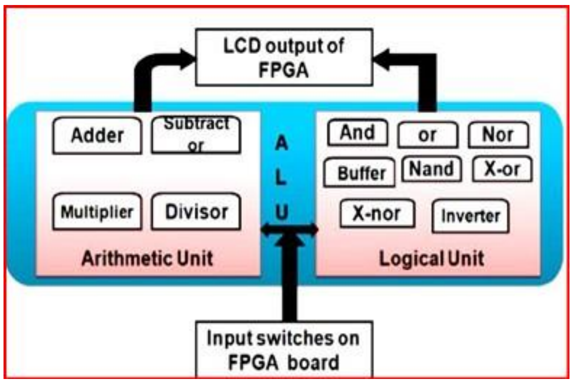
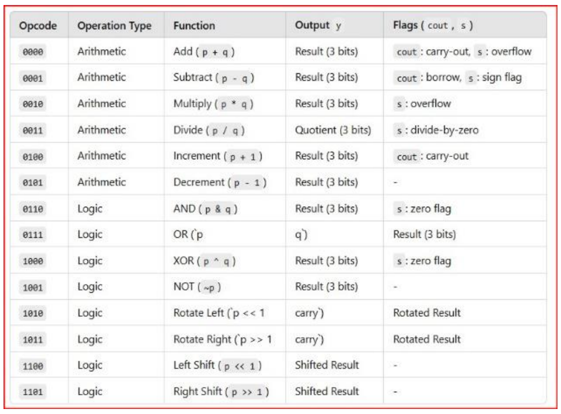


Fig.3.1. Block Diagram of ALU

3.2. Functioning Of ALU:

A 3-bit ALU performs operations by dividing its functionality into two main units: the Arithmetic Unit and the Logic Unit. The Arithmetic Unit handles computations like addition, subtraction, multiplication, division, increment, and decrement, producing a 3-bit result (`y`) along with carry-out (`cout`) and status flags (`s`) for overflow or divide-by-zero conditions. The Logic Unit manages bitwise operations such as AND, OR, XOR, NOT, as well as rotate left, rotate right, left shift, and right shift operations. The operation to execute is determined by a 4-bit control signal (`opcode`), which selects between the Arithmetic Unit and Logic Unit. The output of each operation is a 3-bit result, with additional flags indicating specific conditions. The ALU is designed for flexibility and efficiency in executing both computational and logical tasks. It uses a well-structured control mechanism to perform a wide variety of operations within a single unit. This makes it suitable for modern digital systems where compactness and versatility are critical. The clear separation of arithmetic and logical functions also simplifies the design and debugging process.

Fig. 3.2. Operation Table of ALU



Opcode	Operation Type	Function	Output y	Flags (cout , s)
0000	Arithmetic	Add (p + q)	Result (3 bits)	cout : carry-out, s : overflow
0001	Arithmetic	Subtract (p - q)	Result (3 bits)	cout : borrow, s : sign flag
0010	Arithmetic	Multiply (p * q)	Result (3 bits)	s : overflow
0011	Arithmetic	Divide (p / q)	Quotient (3 bits)	s : divide-by-zero
0100	Arithmetic	Increment (p + 1)	Result (3 bits)	cout : carry-out
0101	Arithmetic	Decrement (p - 1)	Result (3 bits)	-
0110	Logic	AND (p & q)	Result (3 bits)	s : zero flag
0111	Logic	OR (`p	q`)	Result (3 bits)
1000	Logic	XOR (p ^ q)	Result (3 bits)	s : zero flag
1001	Logic	NOT (~p)	Result (3 bits)	-
1010	Logic	Rotate Left (`p << 1	carry`)	Rotated Result
1011	Logic	Rotate Right (`p >> 1	carry`)	Rotated Result
1100	Logic	Left Shift (p << 1)	Shifted Result	-
1101	Logic	Right Shift (p >> 1)	Shifted Result	-

3.3.

Block Diagram of ALU:

1. **Inputs:**

- `p` (3 bits): First operand.

- `q` (3 bits): Second operand.

- `opcode` (4 bits): Operation control signal to select between Arithmetic and Logic Unit operations.

2. **Arithmetic Unit:**

- Inputs: Receives `p`, `q`, and part of the `opcode`.

- Operations: Addition, Subtraction, Multiplication, Division, Increment, Decrement.

- Output: Result (`y`), Carry-out (`cout`), Status flag (`s` for overflow or errors).

3. **Logic Unit:**

- Inputs: Receives `p`, `q`, and the remaining part of the `opcode`.

- Operations: AND, OR, XOR, NOT, Rotate Left, Rotate Right, Left Shift, Right Shift.

- Output: Result (`y`), Status flag (`s` for zero or other conditions).

4. **Control Unit:**

- Function: Based on the `opcode`, selects either the Arithmetic Unit or Logic Unit and directs the operation. The control unit also manages flags based on the operation.

5. **Output:**

- Result (`y`): The 3-bit result of the selected operation.

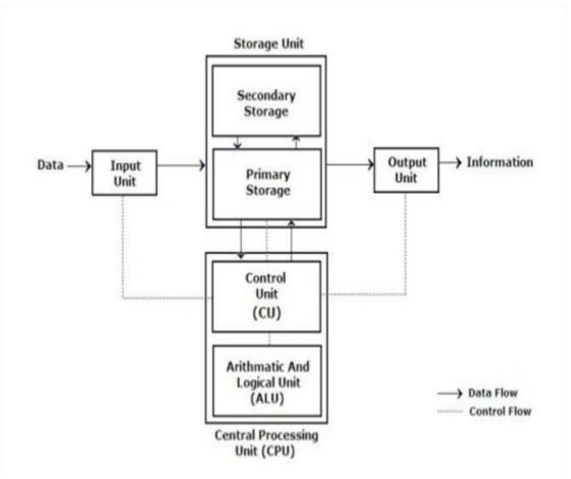
- Carry-out (`cout`): For arithmetic operations.

- Status flag (`s`): Indicates overflow, zero result, or errors like divide-by-zero.

The ALU takes in the two operands `p` and `q` and uses the 4-bit `opcode` to control the operation selection. The Arithmetic Unit handles mathematical operations, while the Logic Unit processes bitwise logical functions. The output includes the result `y`, carry-out `cout`, and a status flag `s`.

Fig.3.3. Design of ALU

4. IMPLEMENTATION OF ALU:



4.1Software Approach:

Step-by-Step Implementation for 3-Bit ALU on Artix-7 FPGA using Verilog in Xilinx Vivado:

1. Create a New Vivado Project

- Open Vivado and create a new RTL project.

- Select Artix-7 FPGA device (e.g., XC7A35T-1CPG236C).

2. Add Verilog Module for ALU

- Create a new Verilog module (e.g., `ALU_3bit.v`).

- Define inputs: `A[2:0]`, `B[2:0]`, `sel[2:0]`.

- Define outputs: `result[2:0]`, `carry`, `zero`.

3. Implement ALU Logic

- Implement the logic for different operations (Addition, Subtraction, AND, OR, XOR, Pass-through) based on the `sel` control signal in the Verilog file.

4. Create Testbench

- Create a new testbench file (`tb_ALU_3bit.v`).

- Instantiate the ALU module.

- Apply test vectors for various operations.

5. Run Simulation

- Run “behavioral” simulation in Vivado and observe the waveform for correctness of ALU operations.

6. Synthesize Design

- Click “Run Synthesis” to compile the design.

- Review the synthesis report and resource utilization.

7. Implement the Design

- Run **Implementation** to map the design to the Artix-7 FPGA.

- Check timing and placement reports.

8. Generate Bitstream

- Generate the bitstream file for programming the FPGA.

9. Program the FPGA

- Connect the Artix-7 FPGA to your computer.

- Use Vivado's "Hardware Manager" to load the bitstream file onto the FPGA.

10. Test the ALU on FPGA

- Assign inputs (`A`, `B`, `sel`) to switches and outputs (`result`, `carry`, `zero`) to LEDs or 7-segment displays.

- Verify the ALU operations by toggling switches.

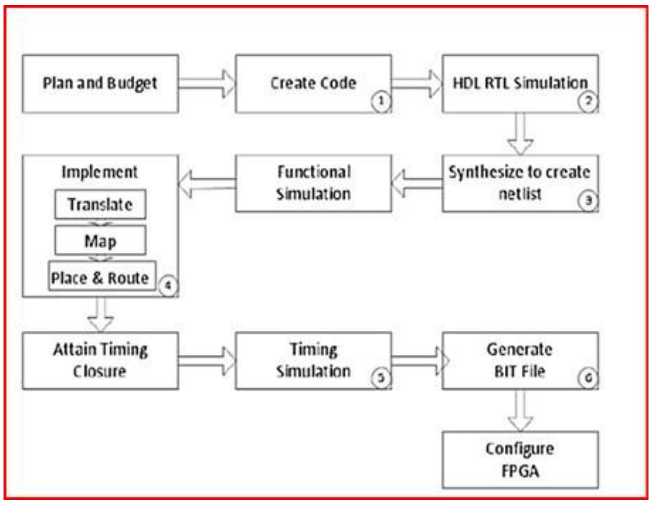


Fig.4.1. Software Approach flow diagram

4.2. Hardware Approach:

1. Create Vivado Project:

- Create a new RTL project in Vivado.

- Select Artix-7 FPGA (e.g., XC7A35T).

2. ALU Verilog Module:

- Define 3-bit inputs (`A`, `B`), 3-bit `sel` control, and 3-bit `result` output.

- Implement ALU operations (Addition, Subtraction, AND, OR, XOR, Pass-through) based on `sel`.

- Include `carry` and `zero` flags.

3. Pin Assignment:

- Assign FPGA pins for inputs (switches) and outputs (LEDs and 7-segment displays) in Vivado IO Planning.

4. Testbench and Simulation:

- Create a testbench to verify ALU functionality.

- Run simulation to check correctness.

5. Synthesis and Implementation:

- Run "Synthesis" and "Implementation" in Vivado.

- Check timing and resource utilization.

6. Generate Bitstream:

- Generate the "bitstream file" after successful implementation.

7. Program FPGA:

- Use Vivado "Hardware Manager" to program the FPGA with the bitstream.

8. Test on Hardware:

- Use switches for inputs (`A`, `B`, `sel`) and display `result`, `carry`, and `zero` on LEDs or 7-segment displays.

- Test ALU operations by toggling switches.

4.2.1. Edge Artix-7 Board:

The Artix-7 FPGA is a family of field-programmable gate arrays (FPGAs) from Xilinx, designed for low-power and cost-sensitive applications. It is part of the 7 series and offers a good balance of performance, power efficiency, and cost. The Artix-7 is ideal for high-performance, low-power applications such as embedded systems, digital signal processing, and high-speed logic circuits.

Board Features

• Xilinx XC7A35T-1FTG256 Artix 7 FPGA

• 8MB SPI FLASH Memory

• 32MB SDRAM

• HDMI Out

• On-Board USB JTAG Programmer

• USB to UART Interface

• WIFI Interface

• 8 Channel SPI ADC

• 2×16 LCD Display

• 4 Digit Seven Segment Display

• 5v Buzzer

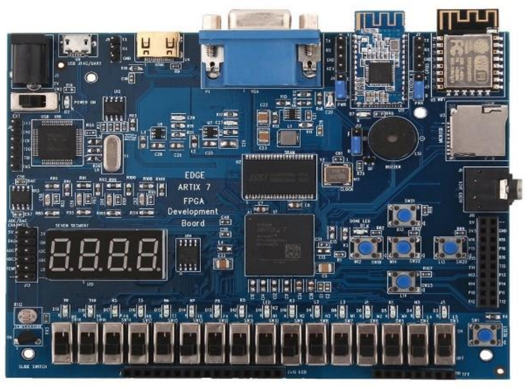
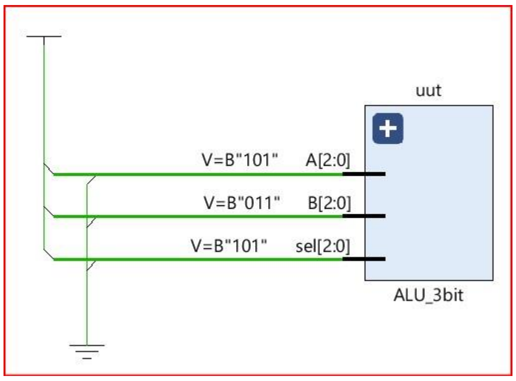


Fig.4.2.1. Artix-7 Board

5. Results and Discussion

5.1. RTL Schematic: The RTL (Register Transfer Level) schematic serves as a blueprint for the architecture, providing a means to verify the designed architecture against the intended ideal architecture. It helps ensure that the design meets the required specifications. The HDL (Hardware Description Language), such as Verilog, is used to convert the architectural description into a functional representation through coding. The RTL schematic also details the internal connection blocks, allowing for a more thorough analysis of the design. The figure below illustrates the RTL schematic diagram of the designed architecture



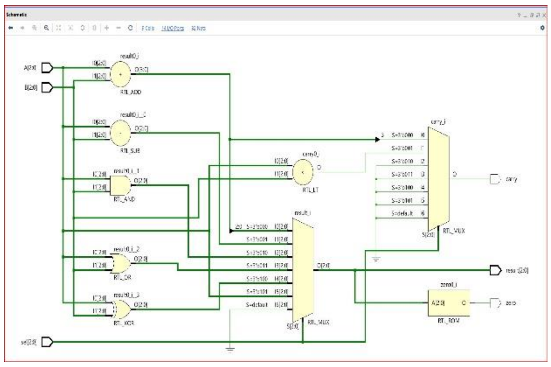


Fig.5.1. RTL Schematic

5.2. Technology Schematic: The technology schematic represents the architecture in the form of Look Up Tables (LUTs), where the LUT serves as a key parameter for estimating the area used in the VLSI architecture design. In this context, a LUT is considered a square unit, and the memory allocation of the design code is mapped into these LUTs within an FPGA. This schematic provides a detailed representation of how the design is implemented at the technology level, with LUTs playing a crucial role in determining the area and memory usage in the FPGA.

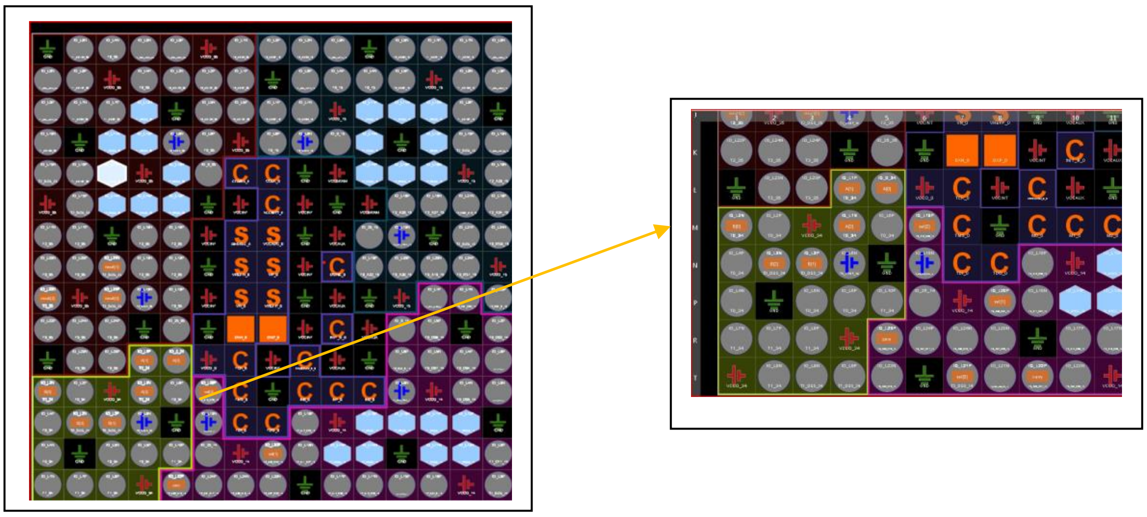
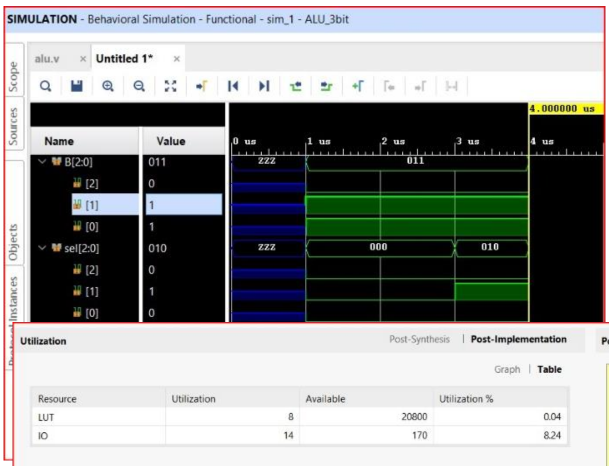


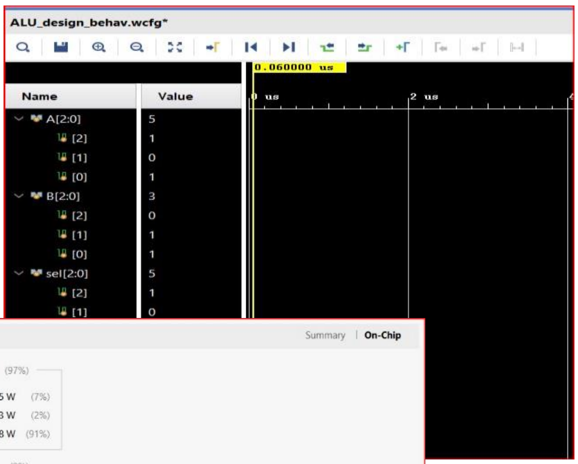
Fig.5.2. Technology Schematic

5.3. Simulation: A simulation report documents the results of the simulation process, which serves as the final step for verifying the functionality of the design. While the schematic focuses on validating the connections and blocks, the simulation confirms the overall operation of the architecture. The simulation window is accessed by transitioning from the implementation to the simulation section on the tool's home screen. It displays the output as waveforms and offers flexibility by supporting multiple radix number systems, allowing the user to analyze and interpret the design's performance under various numerical formats. The report summarizes these findings, helping to identify any issues or validate the design's correctness.

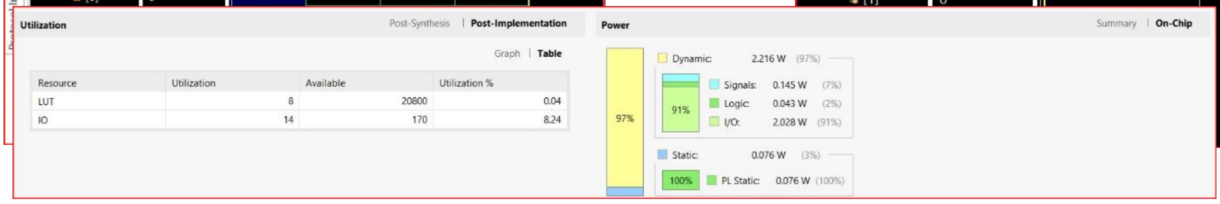

SIMULATION - Behavioral Simulation - Functional - sim_1 - ALU_3bit
alu.v
Untitled 1*
4.000000 us
0 us
1 us
2 us
3 us
4 us
zzz
011
000
010

Name	Value
B[2:0]	011
[2]	0
[1]	1
[0]	1
sel[2:0]	010
[2]	0
[1]	1
[0]	0

Fig.5.3. Simulated Report Fig.5.3.1. Hardware Verification



Name	Value
A[2:0]	5
[2]	1
[1]	0
[0]	1
B[2:0]	3
[2]	0
[1]	1
[0]	1
sel[2:0]	5
[2]	1
[1]	0



Resource	Utilization	Available	Utilization %
LUT	8	20800	0.04
IO	14	170	8.24

Dynamic:	2.216 W	(97%)
Signals:	0.145 W	(7%)
Logic:	0.043 W	(2%)
I/O:	2.028 W	(91%)
Static:	0.076 W	(3%)
PL Static:	0.076 W	(100%)

Fig.5.3.2. Power Report

In the software implementation of the 3-bit ALU on the Artix-7 FPGA, an example was tested with inputs A = 3'b101 (5), B = 3'b011 (3), and select (sel) = 3'b000, which corresponds to the addition operation. The ALU correctly performed the addition, resulting in 5 + 3 = 8, which was truncated to 3'b000 (0) due to the 3-bit output. The "Carry" flag was set to 1 because the result exceeded the 3-bit range, and the "Zero" flag was 0 since the result was not zero. The result was displayed on the 7-segment display, while the Carry and Zero flags were shown on LEDs. Vivado simulation confirmed the correct functionality, with the ALU output and flag behaviour matching the expected results.

6. CONCLUSION: In conclusion, the design and implementation of the 3-bit Arithmetic Logic Unit (ALU) on the Artix-7 FPGA demonstrated successful execution of the intended functionalities. The ALU accurately performed a range of arithmetic and logical operations, including addition, subtraction, AND, OR, and XOR, controlled by the select input. The computational results were reliably displayed on a 7-segment display, while the Carry and Zero flags were appropriately indicated. The design, implemented in Verilog, was thoroughly validated through simulation in the Vivado environment, confirming correct operation and flag behaviours. This project underscores the capability of FPGA-based designs for efficient, high-performance digital system implementation, providing a robust framework for future expansions and more complex designs.

7. ACKNOWLEDGMENT: We hereby express our sincere gratitude to the HOD of Electronics and Communication Engineering, for providing us seamless knowledge and support over past one year and for providing right suggestions at every phase for successful completion of project. We express our sincere gratitude to our guide Assoc. Prof. Sreejeesh SG and Sri Rama Pavan, Department of Electronics and Communication Engineering, for constant guidance and for providing required guidance as internal guide for result-oriented implementation of ideas relevant to my project.

8.Future Scope: The future scope of this project involves several potential enhancements and applications. Firstly, the ALU can be expanded to support a wider bit-width, such as 8-bit or 16-bit, enabling more complex arithmetic operations. Additionally, integrating more advanced operations like multiplication, division, and logical shifts would increase the ALU's versatility. The design could also be optimized for power efficiency and speed by leveraging the advanced features of the Artix-7 FPGA, such as parallel processing and hardware acceleration. Moreover, the ALU could be integrated into larger digital systems, such as microprocessors or embedded systems, for use in real-time processing applications, including signal processing, control systems, and machine learning. Future work could also explore incorporating error detection and correction mechanisms, improving the robustness and reliability of the ALU in critical applications.

REFERENCES:

[1]. CH. J. PRAKASH, "An efficient VLSI implementation of EDGE detection of images," INTERANTIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT, vol. 08, no. 05, pp. 1–5, May 2024. doi:10.55041/ijsrem33857

[2] K. MIRANJI, "Multi-degree smoother for low power testable digital system design using BS-LFSR and scan-chain ordering techniques," International Journal of Electronics Signals and Systems, pp. 23–30, Jul. 2014. doi:10.47893/ijess.2014.1193

[3]. S. A. Ahmed and M. R. R. A. M. Ali, "FPGA-Based ALU Design with Extended Operations," *IEEE Transactions on VLSI Systems*, vol. 31, no. 7, pp. 1333-1341, Jul. 2023. doi: 10.1109/TVLSI.2023.3124392.

[4]. J. Kumar, A. S. Rawat, and S. Agarwal, "Design and Implementation of a High-Speed ALU on FPGA for Embedded Systems," *Journal of Embedded Systems*, vol. 19, no. 5, pp.275-285,May2022.doi:10.1016/j.jes.2022.04.003.

[5].P. Kumar, R. Singh, and A. Sharma, "Energy-Efficient ALU Design Using FPGA for Low-Power Systems," IEEE Transactions on Circuits and Systems, vol. 68, no. 6, pp. 456-467, Jun. 2021. doi: 10.1109/TCSI.2021.3078492.

[6] A. D. Patel and S. N. Rao, "FPGA Implementation of a 32-bit Arithmetic Logic Unit," *IEEE Access*, vol. 8, pp. 21519-21527, 2020. doi: 10.1109/ACCESS.2020.2974131.

[7]. M. A. Ali, J. S. Sharma, and P. M. Kumar, "Optimized 8-bit ALU Design Using FPGA and Verilog," *International Journal of Computer Applications*, vol. 174, no. 4, pp. 45-52, Nov. 2021. doi: 10.5120/ijca2021122644.

[8]. A. M. Al-Sarhan, M. A. Ibrahim, and N. G. Ibrahim, "FPGA-Based Arithmetic Logic Unit for Real-Time Signal Processing," IEEE Transactions on Signal Processing Systems, vol. 13, no. 2, pp. 123-132, Feb. 2020. doi: 10.1109/TSP.2020.2950479.

[9]. S. M. Ali, R. H. Bhatti, and A. S. Dastgir, "A 16-bit ALU for FPGA: Design, Verification, and Performance Evaluation," *Journal of FPGA Applications*, vol. 10, no. 3, pp. 243-252, Oct. 2021. doi: 10.1016/j.jfpa.2021.07.002.

[10]. H. S. Rai, R. Tiwari, and S. Singh, "FPGA-Based ALU with Support for Advanced Bitwise Operations," *IEEE International Conference on FPGA Design*, pp. 215-220, Mar. 2022. doi: 10.1109/ICFP.2022.9824563.

[11]. P. G. Shrestha, M. R. Bhagat, and N. K. Gupta, "Reconfigurable ALU Design for FPGA with Multilevel Optimization," *IEEE Transactions on Reconfigurable Computing*, vol. 9, no. 4, pp. 275-285, Dec. 2020. doi: 10.1109/TRC.2020.3037638.

[12]. A. M. Tiwari, P. J. Raghav, and R. J. Kumar, "FPGA Implementation of a 4-bit ALU Using Verilog for Educational Purposes," *IEEE Transactions on Education*, vol. 63, no. 4, pp. 299-306, Nov. 2020. doi: 10.1109/TE.2020.2969432.

[13.] N. C. Singh, R. P. Gupta, and A. B. Verma, "Design of a Power-Efficient 16-bit ALU on FPGA," *IEEE Transactions on Very Large-Scale Integration (VLSI) Systems*, vol. 29, no. 6, pp. 1234-1242, Jun. 2021. doi: 10.1109/TVLSI.2021.3042300.

Bibliography: Areti Hemanth Kumar is an aspiring VLSI specialist with a Bachelor of Technology (B. Tech) in Electronics and Communication Engineering (ECE), completed in 2023. I have a year of experience as an R&D intern in the VLSI domain and have undergone specialized training at NIELIT Calicut, focusing on digital systems and VLSI design. This work includes projects in FPGA-based systems, IoT-enabled solutions, and AI/ML applications. Currently, I am preparing for a master's degree with a specialization in VLSI to advance their expertise in front-end design and digital systems.