DESIGN AND IMPLEMENTATION OF 4-BIT AND 8-BIT ALU USING BASYS 3 FPGA

A MINI PROJECT REPORT

Submitted by

ASHRIN A (ATP22EC011)

to

the APJ Abdul Kalam Technological University

in partial fulfillment for the award of the Degree of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

under the supervision of

DR ANEESH K.

Department of Electronics and Communication Engineering

ISO 9001:2015 Certified Institution. Approved by AICTE & Affiliated to A. P. J. Abdul Kalam Technological University
Ahalia Health, Heritage & Knowledge Village, Palakkad - 678557 Ph: 04923-226666 www.ahalia.ac.in

Accredited by NAAC NATIONAL ASSESSMENT AND ACCREDITATION COUNCIL

MARCH 2025

DECLARATION

I hereby declare that this submission is my own original work. To the best of my knowledge and belief, it contains no material previously published or written by another individual, nor any material that has been submitted for the award of any other degree or diploma at any university or institution of higher learning, except where proper acknowledgment is provided in the text.

Place: Palakkad

Date:

Signature:

Name: ASHRIN A

Reg. No.: ATP22EC011

AHALIA SCHOOL OF ENGINEERING & TECHNOLOGY
ISO 9001:2015 Certified Institution. Approved by AICTE & Affiliated to A. P. J. Abdul Kalam Technological University
Ahalia Health, Heritage & Knowledge Village, Palakkad - 678557 Ph: 04923-226666 www.ahalia.ac.in
Accredited by NAAC NATIONAL ASSESSMENT AND ACCREDITATION COUNCIL

DEPARTMENT OF
ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the report entitled **"DESIGN AND IMPLEMENTATION OF 4-BIT AND 8-BIT ALU USING FPGA"**, submitted by **ASHRIN A (ATP22EC011)** to APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the Bachelor of Technology degree in Electronics and Communication Engineering, is a bona fide record of the project work related to the course **MINI PROJECT**, carried out under our guidance and supervision. This report has not been submitted in any form to any other university or institution for any purpose.

Guide / Supervisor
Dr. Aneesh. K
Department of ECE
Signature:
Date:

Head of Department
Dr. V. Balamurugan
Department of ECE
Signature:
Date:

(Seal of Dept. of ECE)

(Office Seal)

ACKNOWLEDGMENT

First and foremost, I would like to express my deepest gratitude to **GOD ALMIGHTY** for his infinite grace and guidance, without which this project would not have been completed.

I would like to extend my sincere thanks to the **MANAGEMENT** of Ahalia School of Engineering and Technology for their unwavering support in the successful completion of this project.

My heartfelt thanks go to **Dr. P. R. SURESH**, Principal of Ahalia School of Engineering and Technology, for his valuable support and assistance throughout our project journey.

I would like to express my sincere appreciation to our Head of Department **Dr. V. BALAMURUGAN** and Project Coordinator, **DR. ANEESH. K , MS. DIVYA MOHAN**, for continuous guidance and encouragement during this project.

I would also like to convey my heartfelt thanks to **DR. ANEESH. K**, our Project Guide, for his expert guidance, instructions, and support throughout the project.

No work can succeed without the proper reference materials and resources. In this regard, I express my profound gratitude to all the teaching and non-teaching staff of our college, whose support and assistance created a conducive environment for the successful completion of my project.

ABSTRACT

The Arithmetic Logic Unit is a crucial component of any computer processor, responsible for performing both arithmetic and logical operations. This project focuses on the design and implementation of 4-bit and 8-bit ALUs using Verilog in the Xilinx Vivado 2016.2 environment. The ALU is modeled using behavioral modeling techniques, ensuring a high-level abstraction that simplifies the design process. The implemented ALU is synthesized and tested on a BASYS3 FPGA board, demonstrating its functionality and performance.

The designed ALU supports a variety of arithmetic operations, including addition and subtraction, as well as logical operations such as AND, OR, XOR, and NOT. The power consumption of the ALU design is analyzed to assess its efficiency. The on-chip power consumption is measured at 2.715W for the 4-bit ALU and 2.476W for the 8-bit ALU. This power consumption includes both dynamic and static components, with the 4-bit ALU consuming 2.613W dynamically and 0.102W statically, while the 8-bit ALU consumes 2.375W dynamically and 0.101W statically.

Thermal characteristics of the ALU design are also evaluated. The junction temperature of the 4-bit ALU is recorded at 37.4°C, whereas the 8-bit ALU operates at a slightly lower junction temperature of 36.3°C. These thermal readings indicate efficient heat dissipation and stable operation of the ALU under typical conditions.

CONTENTS

LIST OF FIGURES

ABBREVIATIONS

Abbreviation	Expansion
ALU	Arithmetic Logic Unit
FPGA	Field Programmable Gate Array
HDL	Hardware description languages
DSP	Digital signal processing
ASIC	Application-Specific Integrated Circuit
MUX	Multiplexer
RTL	Register Transfer Level
LED	light-emitting diode

Chapter 1

INTRODUCTION

1.1 What is ALU?

An Arithmetic Logic Unit (ALU) is an essential part of a computer's Central Processing Unit (CPU) that carries out arithmetic and logical operations. It performs calculations such as addition, subtraction, multiplication, and division, along with logical functions like AND, OR, XOR, and NOT.

The ALU processes data in binary form and plays a crucial role in executing instructions within a processor. It also handles decision-making by comparing values and performing bitwise operations. ALUs come in various sizes, such as 4-bit, 8-bit, 16-bit, and 32-bit, depending on the complexity of the computing task.

ALUs are widely used in microprocessors, microcontrollers, and digital circuits, making them a fundamental component in modern computing systems.

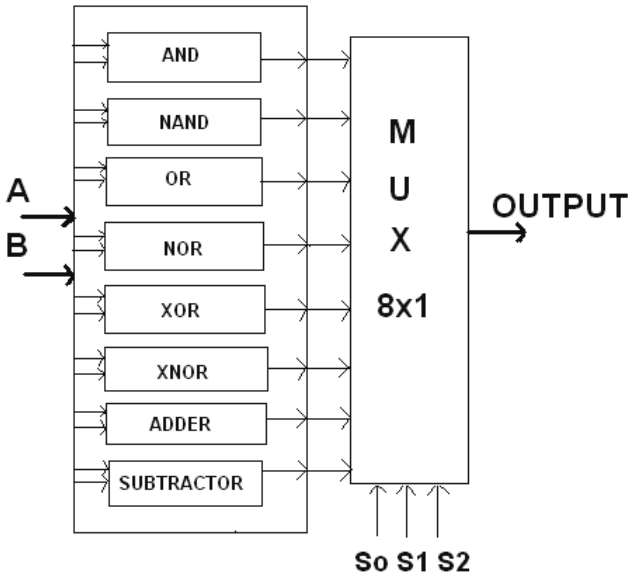

Figure 1.1: Block diagram of ALU

1. **Different types of ALU: 4-bit and 8-bit ALU.**

   1. 4-bit ALU: A 4-bit ALU processes data in 4-bit binary format, meaning it can handle inputs and outputs of 4 bits. It performs basic arithmetic operations like addition, subtraction, multiplication, and division, along with logical functions such as AND, OR, XOR, and NOT. Due to its limited bit width, it is commonly used in simple processors, microcontrollers, and embedded systems requiring low-power computations.

   2. 8-bit ALU: An 8-bit ALU works with 8-bit binary numbers, allowing it to handle larger values and perform more complex calculations than a 4-bit ALU. It supports a broader range of arithmetic and logical operations, making it suitable for advanced microprocessors, controllers, and digital systems where higher precision and processing power are needed.

2. **Functions of ALU:**

   4-bit ALU:

   - A 4-bit ALU processes 4-bit binary numbers and can perform the following operations:

   Arithmetic Operations:

   - Addition: Adds two 4-bit numbers
   - Subtraction: Computes the difference between two values
   - Multiplication: Performs limited multiplication of small numbers
   - Division: Basic division with remainder handling

   Logical operation:

   - AND: Performs bitwise AND operation
   - OR: Executes bitwise OR operation
   - XOR: Computes exclusive OR operation
   - NOT: Inverts individual bits

Bitwise Operations:

- Left Shift: Shifts bits to the left, multiplying the value by 2
- Right Shift: Shifts bits to the right, dividing the value by 2

Comparison Operations:

- Equal to (==): Compares two values
- Greater than (>) and Less than (<): Checks relational conditions

8-bit ALU:

- An 8-bit ALU can handle 8-bit binary numbers, offering greater processing power and additional functions:

Arithmetic Operations:

- Addition and Subtraction: Handles larger values compared to a 4-bit ALU
- Multiplication and Division: Supports more complex calculations
- Increment and Decrement: - Increases or decreases values by one

Logical operation:

- AND, OR, XOR, NOT: Performs bitwise logical functions

1.2 Description of an Arithmetic Logic Unit (ALU) using Verilog code:

The ALU is a digital circuit that performs arithmetic and logical operations on two input operands. It supports addition, subtraction, multiplication, and division, as well as logical operations like AND, OR, XOR, and NOT. The ALU takes two 4-bit input operands, Operand A and Operand B, and a 3-bit operation code, Operation. It produces a 4-bit result, Result, along with carry-out, Carry-Out, and zero, Zero, flags.

The Verilog code defines the ALU module, which includes an always block that performs the specified operation based on the operation code. For addition, the code

uses the built-in "+" operator to add the two operands and assigns the result to the sum wire. The carry-out is also generated and assigned to the Carry-Out output. The Result output is then assigned the value of the sum, and the Zero flag is set based on whether the result is zero.

Similar logic is implemented for subtraction, multiplication, and division operations. However, the code for division is incomplete and would require additional implementation to handle division by zero and other edge cases.

- **Design of ALU using Verilog code**

  An Arithmetic Logic Unit (ALU) is a fundamental component of a processor that performs arithmetic and logical operations. The ALU in Verilog is designed using a multiplexer-based approach, where different operations like addition, subtraction, AND, OR, XOR, and NOT are selected based on a control signal.

  - Inputs: Two operands (A and B) and a selection signal that determines the operation.
  - Outputs: The result of the operation and additional flags like carry-out or zero flag.
  - Implementation: The ALU is coded using an always block in Verilog, where different operations are performed based on the selection signal.

  Verilog makes it possible to simulate the ALU before implementing it on hardware, ensuring correct functionality and performance optimization.

1.3 Importance of Simulating ALU on FPGA

Simulating an ALU on an FPGA (Field Programmable Gate Array) is essential for verifying its performance and detecting errors before hardware implementation.

Key Reasons for Simulation:

- Functionality Testing – Ensures the ALU correctly performs arithmetic and logic operations.

- Performance Analysis – Helps evaluate timing, power consumption, and efficiency.
- Error Detection – Identifies and corrects design issues before hardware deployment.
- Design Optimization – Allows improvements in speed and resource utilization.
- Hardware Integration – Ensures the ALU functions properly in a larger system, such as a processor.

Simulating on an FPGA provides a cost-effective way to refine the design, ensuring a reliable and efficient ALU before final implementation.

1.4 Motivation

The motivation behind this project stems from both academic learning and practical applications in digital computing. Here are some key driving factors:

1. **Educational and Learning Purpose**
    - **Hands-on FPGA Experience:**This project provides practical exposure to FPGA programming using Verilog/VHDL and tools like Xilinx Vivado.
    - **Understanding ALU Operations:** The ALU is a fundamental component of any processor, and implementing it helps in learning computer architecture and digital logic design.
    - **Bridging Theory and Practice:** Concepts like Boolean algebra, logic gates, multiplexers, and arithmetic circuits can be applied in real-world design.
2. **Relevance to Industry and Research**
    - **FPGA-Based Computing:** With the growing adoption of FPGA-based accelerators in AI, cryptography, and embedded systems, this project builds foundational knowledge for future innovations.

- **Processor Design Skills:** Understanding how an ALU works is crucial for developing custom processors, embedded systems, and DSPs.
- **Prototyping for Custom Applications:**FPGA-based ALUs allow for flexibility and optimization in hardware designs, unlike fixed-function CPUs.

3. Real-World Applications

- **IoT and Embedded Systems:** Small-scale ALUs like 4-bit and 8-bit are widely used in microcontrollers, IoT devices, and low-power computing.
- **Cryptographic and Security Applications :** Custom ALUs help in implementing specialized encryption and decryption algorithms on FPGAs.
- **Robotics and Automation:**ALUs are essential in decision-making circuits for robotic control systems.

4. Basys 3 FPGA as a Development Platform

- **Affordable and Beginner-Friendly :** The Basys 3 board is widely used in academia due to its Xilinx Artix-7 FPGA, built-in peripherals, and cost-effectiveness.
- **Real-Time Debugging with LEDs and Switches:** The board provides an interactive way to test ALU operations using buttons, switches, and seven-segment displays.
- **Scalability:** Once a 4-bit and 8-bit ALU is successfully implemented, extending it to 16-bit, or beyond becomes a natural progression.

Chapter 2

LITERATURE REVIEW

Prachi Sharma1 et al., explained “The design and implementation of an Arithmetic Logic Unit using Hardware Description Language on Xilinx Vivado 14.7, targeting Field Programmable Gate Arrays . The ALU plays a pivotal role in digital computer systems, performing Boolean and arithmetic operations. The goal of this study is to develop efficient algorithms that optimize the use of available hardware. Since hardware operations are inherently simple and primitive, they rely on a hierarchical approach, where higher-level operations are constructed from basic Boolean and arithmetic functions.

Efficiency in ALUs is measured by their speed, power consumption, and hardware utilization. This paper presents the simulation and synthesis of various ALU configurations using VHDL on the Xilinx Vivado 14.7 platform and the Basys 3 Artix 7 FPGA board, aiming to analyze and compare the key performance parameters of these designs.” [1]

Swastik Rao et al., explained “The Arithmetic Logic Unit is the core component of any CPU and can also be integrated into programmable reversible computing devices, such as quantum computers. A significant challenge in traditional ALU design, using standard logic gates, is high power consumption. This issue primarily arises from the use of irreversible gates. To address this concern and meet low-power design requirements, reversible gates have been introduced. In reversible gates, the number of inputs equals the number of outputs, and there is a one-to-one mapping between inputs and outputs, ensuring energy-efficient operation. The proposed ALU design is described using VHDL and implemented on an FPGA platform.” [2]

Amrit Kumar Panigrah et al., explained "ALU is a fundamental component of a computer processor, playing a crucial role in executing most instructions. It performs various arithmetic and logic operations, including addition, subtraction, multiplication, bit-shifting, and logical functions. Design and implementation of an ALU using Xilinx Vivado 2016.2 and its deployment on Field Programmable Gate Arrays for performance analysis. The primary objective of this design is to develop efficient algorithms that optimize hardware utilization. Key efficiency metrics include improved processing speed, reduced power consumption, and better resource utilization. simulation and synthesize different ALU parameters using Verilog in Xilinx Vivado 2016.2 and implement them on a BASYS3 FPGA board." [3]

Aarti Jagtap et al.,proposed to develop an autonomous output validation system by integrating a clock mechanism into an 8-bit ALU using Verilog. The ALU, characterized by its 8-bit data width, employs a clock system synchronized with positive edges, allowing it to automatically evaluate outputs after each operation without manual adjustments for specific line inputs. This clock-driven output validation mechanism ensures correctness and dependability by assessing outputs at each positive clock edge, thereby eliminating the need for manual intervention, simplifying the verification process, and enhancing overall operational efficiency. To validate the design's functionality in real-time scenarios, implemented the code on an FPGA platform, ensuring seamless integration and reliable performance. Additionally, comprehensive analysis using Xilinx tools provided insights into path delays and device utilization, which are crucial for optimizing the design for practical applications. [4]

Zhouhao Huang et al., The Arithmetic Logic Unit is a fundamental component of modern processors, responsible for executing arithmetic and logical operations. It plays a crucial role in processor architecture by performing essential computational tasks. The design and implementation of ALUs using hardware description languages and simulation tools to enhance their functionality and efficiency.

The design, implementation, and simulation of 8-bit ALUs using software tools like Quartus II and ModelSim have focused on developing ALUs capable of executing a comprehensive set of operations, including addition, subtraction, multiplication, division, shifting, rotation, and various logical functions such as AND, OR, XOR, NOR, NAND, XNOR, and comparison. The design methodology typically involves meticulous circuit development using Quartus II, followed by extensive validation through simulation in ModelSim.

Simulation results have provided critical insights into ALU behavior and performance, showcasing its operational accuracy for different instructions. The resulting waveforms serve as an essential tool for evaluating the ALU's functionality, reliability, and efficiency. Furthermore, these studies highlight the importance of simulation-driven validation in optimizing ALU design, paving the way for further enhancements in performance, power consumption, and resource utilization. [5]

Chapter 3

PROJECT DESCRIPTION

3.1 Working Principle

The Arithmetic Logic Unit is designed to perform arithmetic and logical operations based on control inputs. When implemented on an FPGA, it operates as a combinational circuit, where the result depends only on the current inputs.

- **1. Input and Control Signals:** The ALU takes two binary inputs (A & B) (4-bit or 8-bit depending on the design). An Opcode (control signal) selects the operation to be performed.
- **2.Operation Execution:** Arithmetic Operations:Addition and subtraction are performed using an adder/subtractor circuit, often built using full adders and 2's complement logic. Logical Operations: AND, OR, XOR, and NOT operations are performed using basic logic gates.
- **3. Result and Status Flags:** The final result is selected using a MUX based on the opcode.
- **4.FPGA Implementation:** The ALU is programmed in Verilog or VHDL and synthesized using tools like Vivado.

  It is deployed on an FPGA board Basys 3 and tested using switches and LEDs for input/output visualization.

The implementation of an ALU on an FPGA provides an efficient way to perform arithmetic and logical operations with high accuracy and speed. By utilizing Verilog or VHDL and tools like Vivado, the ALU can be synthesized and tested on FPGA boards such as Basys 3. The use of multiplexers and status flags ensures precise operation

selection and result evaluation. This design not only enhances processing efficiency but also serves as a foundation for further optimization in digital systems.

3.2 Methodology

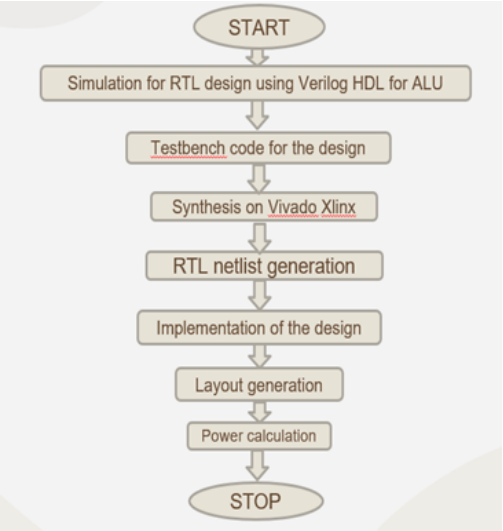

Figure 3.1: Methodology

1. Simulation for RTL design using Verilog HDL for ALU:

- **Purpose:** Define and verify ALU behavior at the Register Transfer Level (RTL)..
- **Process:** 1. Design Specification: Define ALU operations (arithmetic, logic) and control signals.

  2. Verilog Coding: Implement registers, combinational logic, and control mechanisms.

  3. RTL Abstraction: Focus on data flow without gate-level details.

2. Testbench for ALU Verification:

- **Purpose:** Validate ALU functionality by providing input stimuli and checking outputs.

- **Process:** 1.Stimulus Generation: Generate operand and control signal combinations.

  2. Applying Stimuli: Feed inputs to the ALU module.

  3.Output Monitoring: Observe ALU responses.

  4.Verification: Compare actual vs. expected outputs using waveforms or assertions.

  5.Simulation: Run in a Verilog simulator (e.g., ModelSim, VCS) to visualize behavior.

3. Synthesis on Vivado Xilinx

- **Purpose:** Convert Verilog RTL code into a low-level netlist for FPGA implementation.

- **Process:** 1.Tool Invocation: Load ALU Verilog files in Vivado.

  2.Constraints: Apply FPGA-specific parameters (device, clock, I/O).

  3.Optimization: Improve speed, area, and power efficiency.

  4.Netlist Generation: Create a gate-level representation for FPGA deployment.

4. RTL Netlist Generation

- **Purpose:** Intermediate representation of the synthesized design before FPGA placement and routing.

- **Process:** 1.Logic gates (AND, OR, NOT, XOR).

  2.Sequential elements (flip-flops, registers).

  3.Signal interconnections and timing details.

5.Implementation of the Design

- **Purpose:** Maps the synthesized netlist onto the FPGA resources through placement and routing.

- **Process:**

  1.Placement: Assigns physical locations for logic components.

  2.Routing: Establishes signal paths while meeting timing constraints.

  3.Optimization: Enhances performance based on the FPGA architecture.

6.Layout Generation

- **Purpose:** Creates a visual representation of the FPGA design after placement and routing.

- **Process:**

  1.Locations of logic elements and interconnections.

  2.FPGA resource utilization (CLBs, LUTs, flip-flops, routing).

  3.Used for generating the bitstream to configure the FPGA.

7.Power Calculation

- **Purpose:** Estimates power consumption to assess heat dissipation, battery life, and system reliability

- **Process:**

  1.Tool Analysis: Vivado estimates power based on switching activity, frequency, and FPGA characteristics.

  2.Simulation Data: Uses testbench activity for accurate dynamic power estimation.

  3.Power Types: Calculates static (idle) and dynamic (switching) power.

  4.Reporting: Generates power reports to optimize design efficiency

3.3 ALU Architecture

3.3.1 Importance

The ALU is one of the most critical components in digital computing systems, responsible for executing arithmetic and logical operations. It serves as the computational backbone of processors, microcontrollers, and digital circuits, influencing the overall performance and efficiency of a computing system. The architecture of an ALU plays a significant role in determining how efficiently computations are performed, affecting processing speed, power consumption, and hardware complexity. This essay explores the importance of ALU architecture in various aspects, including computational efficiency, digital system design, FPGA implementation, and real-world applications.

- **1. Computational Efficiency and Performance:** The ALU's architecture directly impacts a processor's ability to perform computations efficiently. Modern ALUs are designed to execute operations such as addition, subtraction, multiplication, bitwise logic operations, and shifts at high speeds. By optimizing the internal structure—such as using Carry Lookahead Adders instead of Ripple Carry Adders—computational delays can be significantly reduced.

  Additionally, pipelining and parallel processing techniques in advanced ALU architectures enhance execution speed. For example, modern processors often incorporate multiple ALUs to execute instructions simultaneously, improving throughput and overall system performance. A well-designed ALU architecture enables high-speed calculations, which is crucial for applications such as real-time data processing, gaming, artificial intelligence, and scientific computing

- **2. Role in Digital System Design:** The ALU is a fundamental building block in digital circuits, influencing the design of microprocessors, embedded systems, and custom digital logic designs. A well-structured ALU simplifies control unit design, reduces complexity in data paths, and ensures compatibility with memory and input/output units.

In microprocessor design, ALUs are often optimized for specific use cases. For example:

General-purpose processors require ALUs capable of performing a wide range of operations efficiently. Application-specific processors (e.g., DSPs for signal processing) use specialized ALU architectures optimized for tasks such as fast multiplication and filtering. Low-power embedded systems prioritize energy-efficient ALU designs to extend battery life. The scalability of ALU architecture also plays a crucial role in designing processors with different bit-widths (4-bit, 8-bit, 16-bit, 32-bit, etc.), allowing for flexibility in different applications.

- **3. Real-World Applications and Industry Impact:** The ALU architecture plays a vital role in several real-world applications, including:

  Computers and Microprocessors: Every modern processor, from simple embedded chips to high-performance CPUs and GPUs, relies on efficient ALU designs for fast computations.

  Artificial Intelligence and Machine Learning : Specialized ALUs, such as those in Tensor Processing Units (TPUs), accelerate matrix operations crucial for deep learning.

  Cryptography and Security: ALUs perform high-speed encryption and decryption operations, ensuring secure communication in banking, defense, and cybersecurity.

  Medical Imaging and Signal Processing: ALUs in DSPs enhance the processing speed of CT scans, MRI images, and real-time audio processing.

  Autonomous Systems: Drones, self-driving cars, and robotic automation rely on ALUs for rapid sensor data processing and decision-making.

3.3.2 4 bit ALU

A 4-bit ALU is a digital circuit that performs arithmetic and logical operations on 4-bit binary numbers. It takes two 4-bit inputs (operands) and processes them based

on control signals that determine the specific operation to be executed. The ALU can perform fundamental arithmetic operations such as addition, subtraction, as well as logical operations like AND, OR, XOR, and NOT. A 4-bit ALU is commonly used in simple processors, digital logic circuits, and FPGA-based designs, serving as the basic building block of more complex computational systems.

Control signal	operation	Input A(4-bit)	Input B(8-bit)	Outout(4-bit)
0000	ADD	1010(10)	0011(3)	1101(13)
0001	SUB	1010(10)	0011(3)	0111(7)
0010	AND	1010	0011	0010
0011	OR	1010	0011	1011
0100	XOR	1010	0011	1001
0101	NOT	1010	-	0101

Figure 3.2: 4 bit ALU Operation table

3.3.3 8 bit ALU

An 8-bit ALU is a digital circuit designed to perform arithmetic and logical operations on 8-bit binary numbers. It is a crucial component in microprocessors, microcontrollers, and digital systems, enabling data manipulation and computation. The ALU takes two 8-bit operands as inputs and executes operations based on control signals that determine the specific function to be performed. Common arithmetic operations include addition, subtraction, while logical operations include AND, OR, XOR, and NOT.An 8-bit ALU is widely used in embedded systems, digital signal processing, and FPGA-based applications, serving as a fundamental building block for computational tasks in digital electronics.

Control signal	operations	Input A(4-bit)	Input B(5-bit)	Output(4-bit)
0000	ADD	11001010 (202)	00001101 (13)	11010111(215)
0001	SUB	11001010 (202)	00001101 (13)	11000001(189)
0010	AND	11001010	00001101	00001000
0011	OR	11001010	00001101	11001111
0100	XOR	11001010	00001101	11000111
0101	NOT	11001010	-	00110101

Figure 3.3: 8 bit ALU Operation table

3.4 Software Description

In the ALU project, the code for both the 4-bit and 8-bit ALU is written using VERILOG HDL. VERILOG provides a high-level software interface, making it easier to design the ALU. One of its major advantages is its syntax, which is similar to C language, enabling faster code development and understanding. The VERILOG code is used to describe the ALU's functionality and can be simulated before implementing it in hardware. This simulation helps in identifying and correcting issues early in the design process. VERILOG also helps generate timing diagrams and waveforms, which allow for detailed analysis of the ALU's power consumption, speed, and efficiency. By simulating the ALU's behavior in the software, designers can ensure that the ALU works as expected before hardware implementation.

3.5 Hardware Description

Once the simulation is successful, the VERILOG code is transferred to an FPGA processor for hardware implementation. The FPGA serves as the physical

platform where the ALU design is realized. Using cable interfacing, the code is loaded from the PC to the FPGA hardware. The code is then synthesized and mapped to the FPGA's input-output pins, which correspond to the actual physical connections on the hardware. After the hardware is configured, the VERILOG code is executed on the FPGA processor. The operation of the ALU is verified by comparing the results obtained from the hardware with the results from the simulation. This final step ensures that the 4-bit and 8-bit ALUs work as expected and the design is correct. By combining software simulation and hardware implementation, the project ensures that both the 4-bit and 8-bit ALU designs are thoroughly tested and optimized for efficiency, speed, and power consumption.

3.5.1 BASYS-3 board

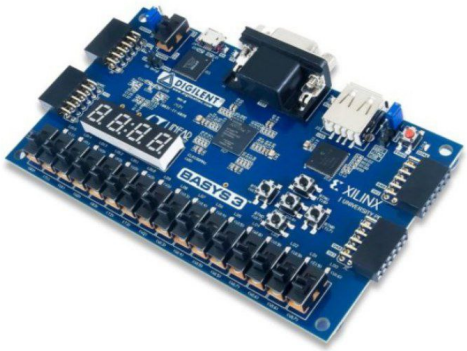

Figure 3.4: BASYS-3 board

The Basys 3 is an FPGA development board by Digilent, featuring the Xilinx Artix-7 XC7A35T FPGA, designed for digital logic design, embedded systems, and computer architecture projects. It offers 33,280 logic cells, 90 DSP slices, 1,800 Kb block RAM, and a 100 MHz onboard clock. The board includes 16 switches, 16 LEDs, 5 push buttons, 4 seven-segment displays, Pmod ports, USB-UART, VGA output, and 4MB Quad-SPI Flash. It supports Verilog and VHDL programming via Vivado Design

Suite and can be powered via USB or an external 5V source, making it ideal for learning FPGA concepts and implementing various digital design projects.

3.6 Implementation code

The following shows the code for the implementation of the 4 bit ALU :

```
module alu_4bit (
    input [3:0] A,    // 4-bit input A
    input [3:0] B,    // 4-bit input B
    input [2:0] ALU_Sel, // 3-bit ALU operation select
    output reg [3:0] ALU_Out, // 4-bit ALU output
    output reg CarryOut // Carry Out flag
);

always @(*) begin
    case (ALU_Sel)
        3'b000: {CarryOut, ALU_Out} = A + B; // Addition
        3'b001: {CarryOut, ALU_Out} = A - B; // Subtraction
        3'b010: ALU_Out = A & B; // AND
        3'b011: ALU_Out = A | B; // OR
        3'b100: ALU_Out = A ^ B; // XOR
        3'b101: ALU_Out = ~(A | B); // NOR
        default: ALU_Out = 4'b0000; // Default case
    endcase
end

endmodule
```

Figure 3.5: 4 bit code

The following shows the code for the implementation of the 8 bit ALU :

```
module ALU (
   input [7:0] A,      // 8-bit input A
   input [7:0] B,      // 8-bit input B
   input [2:0] ALU_Sel, // 3-bit control input to select the ALU operation
   output reg [7:0] ALU_Out, // 8-bit ALU result output
   output reg CarryOut  // Carry Out flag for addition and subtraction
);

always @(*) begin
   case (ALU_Sel)
      3'b000: {CarryOut, ALU_Out} = A + B;       // Addition
      3'b001: {CarryOut, ALU_Out} = A - B;       // Subtraction
      3'b010: ALU_Out = A & B;                   // AND
      3'b011: ALU_Out = A | B;                   // OR
      3'b100: ALU_Out = A ^ B;                   // XOR
      3'b101: ALU_Out = ~A;                      // NOT
      default: ALU_Out = 8'b00000000;            // Default case
   endcase
end

endmodule
```

Figure 3.6: 8 bit code

Chapter 4

RESULTS AND DISCUSSIONS

4.1 RTL SCHEMATIC OF 4-BIT AND 8-BIT ALU

The RTL schematic provides a detailed view of the internal structure of the ALU by translating the code into a gate-level design. It specifies the registers, their connections, and how data flows between them, which simplifies the design and enhances understanding and verification.) schematic provides a detailed view of the internal structure of the ALU by translating the code into a gate-level design. It specifies the registers, their connections, and how data flows between them, which simplifies the design and enhances understanding and verification.

4.1.1 4-bit ALU RTL Schematic:

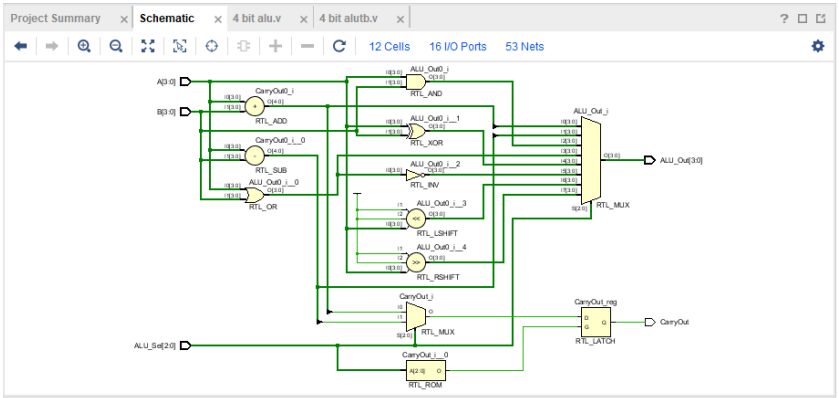

Figure 4.1: 4-bit ALU RTL Schematic

- 4-bit Registers (A and B)- to store operands.
- Control Unit- to select the operation (addition, subtraction, logical operations).
- Combinational Logic- (AND, OR, XOR, etc.) to process the operands.

- Multiplexers- to select inputs based on control signals.

4.1.2 8-bit ALU RTL Schematic:

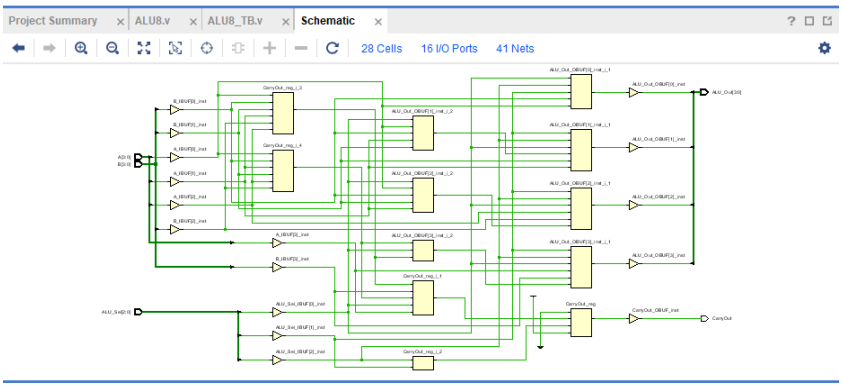

Figure 4.2: 8-bit ALU RTL Schematic

- 8-bit Registers (A and B) for larger operands.
- A Control Unit to manage more complex operations with wider data paths.
- Combinational Logic to handle 8-bit operations.
- Multiplexers and Bus Lines scaled to accommodate 8-bit data.

4.2 SIMULATION

4.2.1 Simulation Of 4-BIT ALU:

- **Inputs:**
- A[7:0] and B[7:0] are the two 8-bit operands.
- ALUSel[2:0] is a 3-bit control signal that selects the operation to be performed.
- **Output:**
- ALUOut[7:0] shows the result of the ALU operation.
- CarryOut indicates if there was a carry generated during the operation.

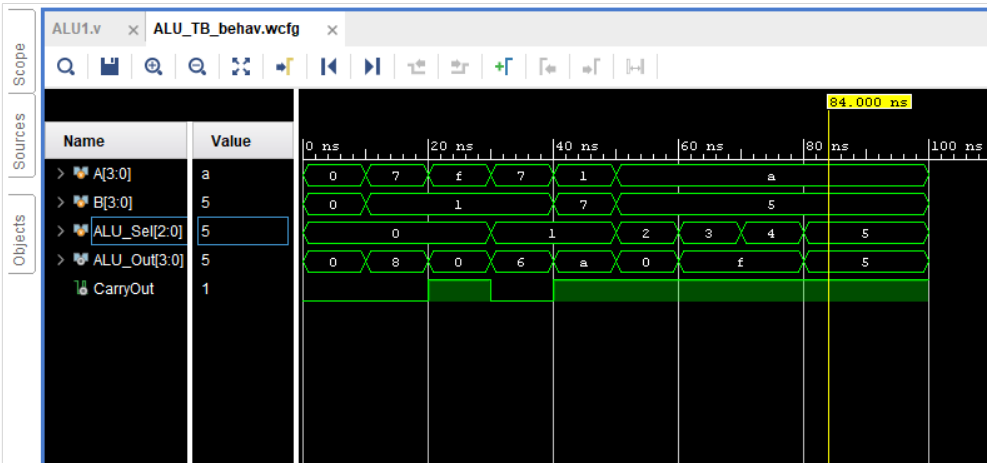

Figure 4.3: Simulation Of 4-BIT

4.2.2 Simulation Of 8-BIT ALU:

- **Inputs:**
- A[3:0] and B[3:0] are the two 4-bit operands.
- ALUSel[2:0] is the operation selector.
- **Output:**
- ALUOut[3:0] is the result of the selected operation.
- CarryOut represents a carry bit if applicable

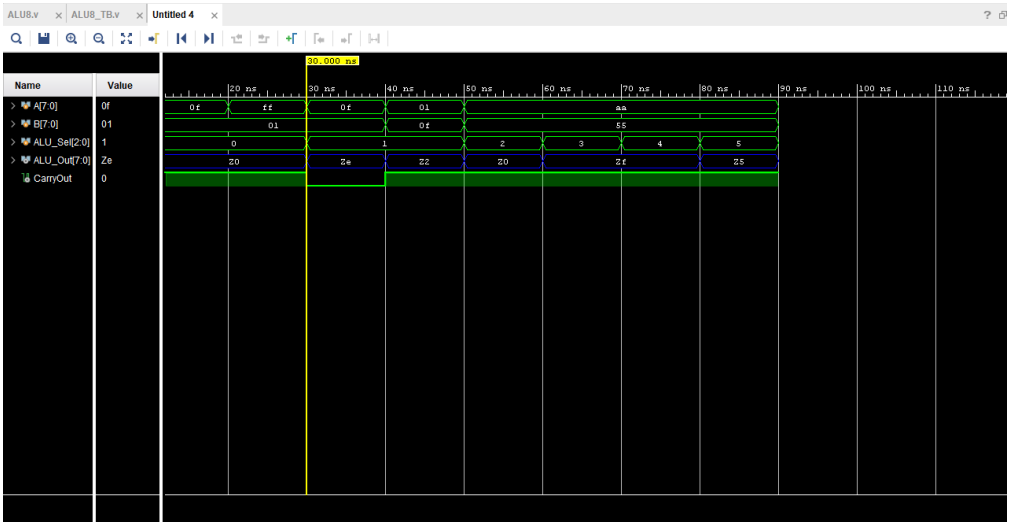

Figure 4.4: Simulation Of 8-BIT

4.3 POWER CONSUMPTION

From the power and timing reports generated during the simulation and synthesis phases of the 4-bit and 8-bit ALU designs on the BASYS 3 FPGA, it can be concluded that both speed and power dissipation have been optimized effectively. This optimization is achieved through careful coding and design, which minimizes the energy consumed during computation while maintaining high operational speed.

The VERILOG HDL code used for designing the 4-bit ALU and 8 bit ALU ensures that the circuit performs efficiently. This is critical for applications like RISC microprocessors, where speed and power efficiency are crucial. The simplicity of the VERILOG code allows for easy adaptation to larger word lengths, such as 16-bit or 32-bit, without requiring major modifications. This flexibility ensures that the design can be scaled to meet the needs of more complex systems. • Speed Optimization: The ALU performs arithmetic and logical operations in a timely manner, with minimized delays between inputs and outputs, thanks to efficient logic implementation. • Power Dissipation Optimization: The design avoids excessive toggling of gates and reduces unnecessary resource usage, resulting in lower power consumption while performing the required ALU operations.

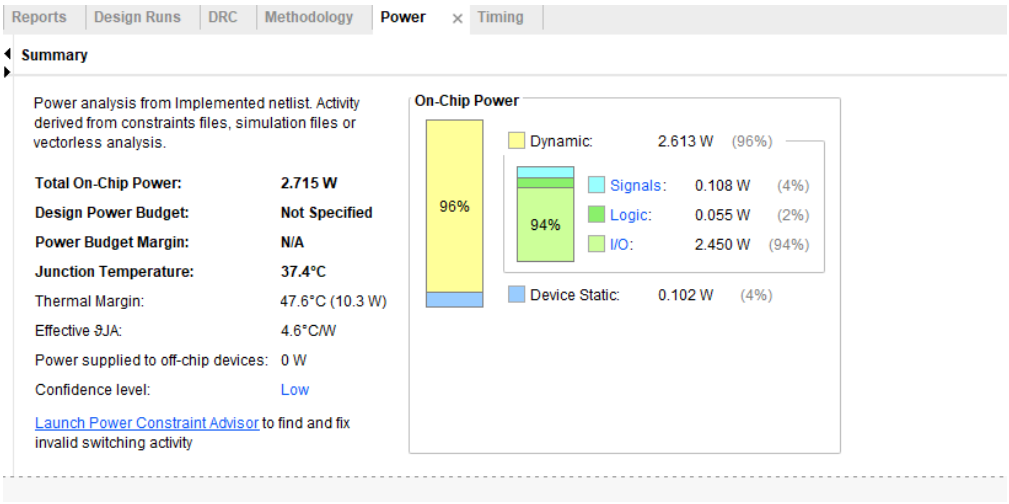

Figure 4.5: Power consumption Of 4-BIT

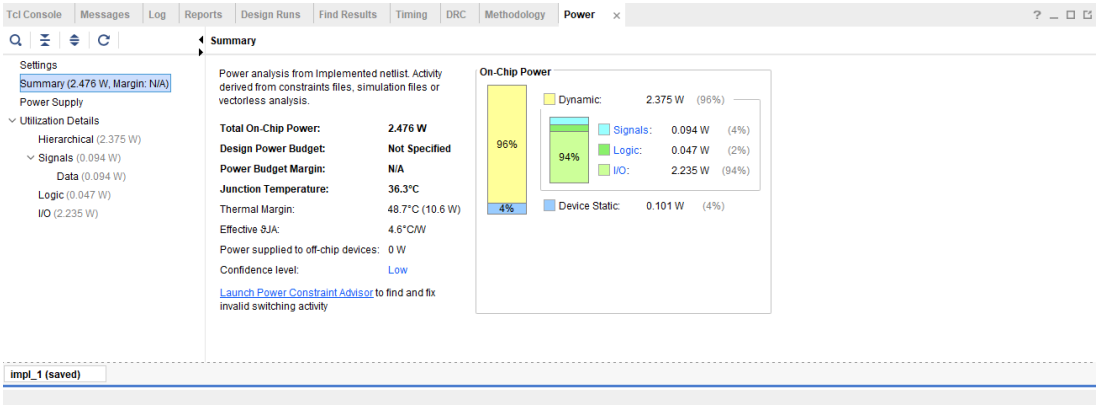

Figure 4.6: Power consumption Of 8-BIT

4.4 IMPLEMENTATION ON BASYS-3

The ALU design is implemented using VERILOG HDL, with the coding, simulation, and synthesis performed in XILINX VIVADO 2016.2. The design is then loaded onto an FPGA using the BASYS3 FPGA kit. The 4-bit ALU takes two 4-bit inputs, A and B, and performs the specified arithmetic or logical operations based on user inputs.

4.4.1 Components of the FPGA:

- Programmable Read-Only Memory: Used for debugging, storing permanent data, and ensuring high-speed operation with a reduced physical size.

- LCD, LEDs, and DIP Switches: These components provide the user interface for input and output. The DIP switches allow the user to set the 4-bit inputs, and the LEDs display the result of the ALU operation.
- JTAG Cable: This cable connects the PC to the FPGA, enabling the transfer of code from the computer to the FPGA for execution.

4.4.2 Operation and Verification:

The 4-bit ALU operates by taking inputs from the DIP switches on the FPGA board. The output of the ALU operation is displayed on the LEDs. The obtained output is then compared with the simulation results to ensure correctness. For the 8-bit ALU, the LEDs are used to display the flag register status, which indicates the condition or result of the ALU operation. Selection lines are provided to control and select the desired operation based on user requirements.

4.4.3 Steps for ALU Implementation:

- 1. Compilation and Simulation: The program is compiled and simulated using XILINX VIVADO, followed by synthesis to prepare the design for hardware implementation.
- 2. Code Loading: The synthesized code is then loaded into the FPGA using the JTAG cable, mapping the input and output pins according to the specified code.
- 3. Operation Testing: Once the design is loaded onto the FPGA, the user provides the necessary input values and selects the desired ALU operation via the selection lines. The result is then displayed on the LEDs.

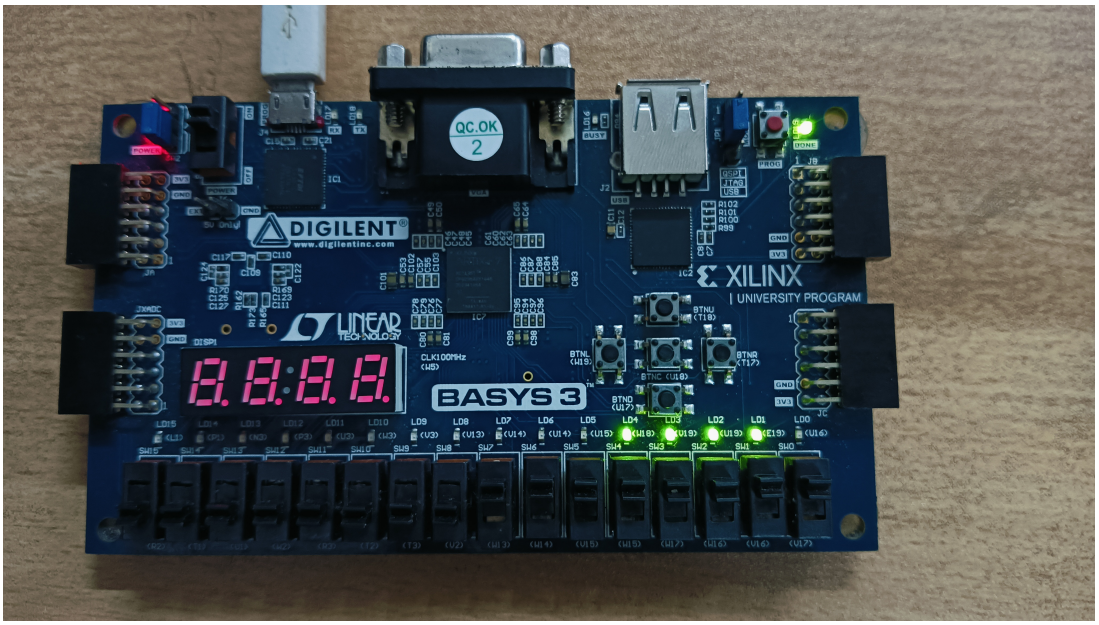

Figure 4.7: AND operation on BASYS 3 FPGA board

The RESET switch on the FPGA allows easy re-uploading of the code if necessary. After loading the design into the FPGA, the hardware operation is verified by comparing the results obtained from the FPGA with those from the simulation, ensuring the design works as intended for both the 4-bit and 8-bit ALU implementations. The hardware realisation was carried out in the FPGA kit. And the obtained output is verified with the help of simulation results.

Chapter 5

FUTURE SCOPE AND CONCLUSION

The future scope of 4-bit and 8-bit ALUs implemented using FPGA revolves around enhancing their efficiency, scalability, and application in various domains. Here are some key areas where advancements and future developments can take place:

1. Enhancement of Efficiency

- **Low Power Design:** Future research in low power design can concentrate on developing power-efficient alus by utilizing low-power logic families, clock gating techniques, and dynamic voltage scaling.
- **Speed Enhancement:**By utilizing parallel processing techniques and pipeline architectures, the execution speed of alus can be significantly enhanced.
- **Reduced Area Utilization:** Further optimization of gate-level design and logic minimization techniques will reduce FPGA resource usage.

2. Expansion to 16-bit, 32-bit, and 64-bit ALUs

- The concepts and architectures of 4-bit and 8-bit ALUs can be scaled up to implement larger ALUs for modern processors and high-performance computing.
- Modular ALU designs can be developed to support both low-bit and high-bit operations on the same FPGA.

3. AI and Machine Learning Applications

- ALUs can be optimized for AI-based applications by integrating neural network accelerators or deep learning hardware components in FPGA-based designs.

- FPGA-based ALUs can assist in performing matrix operations efficiently for machine learning computations.

4. Integration in RISC-V and Custom Processors

- The 4-bit and 8-bit ALU designs can be used in custom RISC processors, especially for embedded systems, IoT, and real-time applications.
- Open-source architectures like RISC-V can be modified to include power-efficient ALU designs.

5. FPGA-Based Cryptography and Security Applications

- Alus can be improved to carry out encryption and decryption tasks for cryptographic purposes.
- Customized fpga alus for security protocols like aes, rsa, and sha can be created.

6. Quantum computing and reversible logic

- By utilizing reversible logic-based alus, power dissipation can be greatly reduced, making it beneficial for quantum computing and energy-efficient applications.
- Future studies may concentrate on developing quantum-inspired alu architectures that harness the power of fpga.

7. FPGA in Space and Defense Applications

- Compact and energy-efficient alus can be utilized in satellites and space exploration missions where minimizing power consumption is of utmost importance.
- Radiation-hardened fpga alus can be created for use in military and aerospace applications.

8. Implementation in IoT and Edge Computing

- 4-bit and 8-bit alus can be tailored for low-power iot devices and smart sensors to efficiently process real-time data.
- FPGA-based ALUs can be incorporated into edge computing platforms to execute lightweight computational tasks without depending on cloud processing.

9. **Fpga-based neuromorphic computing**

- In the future, researchers may develop alus specifically tailored for neuromorphic architectures, emulating the human brain's ability to process information.
- FPGA-based bio-inspired computing models can utilize custom algorithms for cognitive computing tasks.

10. **Automation and Robotics**

- Small-scale alus can be integrated into fpga-driven robotics applications for controlling motion, processing images, and enabling autonomous navigation. Industrial automation systems can greatly benefit from the use of optimized fpga-based alu architectures in real-time decision-making processes.

5.1 CONCLUSION

This paper discusses the design, simulation, and implementation of both 4-bit and 8-bit ALUs using VERILOG HDL on the BASYS 3 FPGA kit. The design was thoroughly verified through the use of simulation waveforms to ensure the correctness of the ALU operations. Additionally, the internal structure of the circuit was analysed using RTL (Register Transfer Level) analysis, providing insights into the data flow and logic operations within the ALU. The code was written using the Xilinx VIVADO tool, which is an essential tool for hardware development and simulation. After synthesizing the code, the ALU design was successfully implemented on the BASYS 3 FPGA, confirming the effectiveness of VERILOG HDL in FPGAbased hardware design. The successful execution of

both the 4-bit and 8-bit ALUs demonstrates the ability of the BASYS 3 FPGA platform to handle various ALU operations efficiently, providing a robust solution for digital logic design.

REFERENCES

[1] Zou, Qin and Sun, Qin and Chen, Long and Nie, Bu and Li, Qingquan. "A Comparative Analysis of LiDAR SLAM-Based Indoor Navigation for Autonomous Vehicles". In: *IEEE Transactions on Intelligent Transportation Systems* 23.7 (2022), pp. 6907–6921. DOI: 10.1109/TITS.2021.3063477.

[2] Roy, Rohit and Tu, You-Peng and Sheu, Long-Jye and Chieng, Wei-Hua and Tang, Li-Chuan and Ismail, Hasan. "Path Planning and Motion Control of Indoor Mobile Robot under Exploration-Based SLAM (e-SLAM)". In: *Sensors* 23.7 (2023). ISSN: 1424-8220. DOI: 10.3390/s23073606. URL: https://www.mdpi.com/1424-8220/23/7/3606.

[3] Shen, Dong and Xu, Yuhang and Huang, Yakun. "Research on 2D-SLAM of Indoor Mobile Robot based on Laser Radar". In: *Proceedings of the 2019 4th International Conference on Automation, Control and Robotics Engineering*. CACRE2019. Shenzhen, China: Association for Computing Machinery, 2019. ISBN: 9781450371865. DOI: 10.1145/3351917.3351966. URL: https://doi.org/10.1145/3351917.3351966.

[4] Mu, Lili and Yao, Pantao and Zheng, Yuchen and Chen, Kai and Wang, Fangfang and Qi, Nana. "Research on SLAM Algorithm of Mobile Robot Based on the Fusion of 2D LiDAR and Depth Camera". In: *IEEE Access* 8 (2020), pp. 157628–157642. DOI: 10.1109/ACCESS.2020.3019659.

[5] Khan, Misha Urooj and Zaidi, Syed Azhar Ali and Ishtiaq, Arslan and Bukhari, Syeda Ume Rubab and Samer, Sana and Farman, Ayesha. "A Comparative Survey of LiDAR-SLAM and LiDAR based Sensor Technologies". In: *2021 Mohammad*

Ali Jinnah University International Conference on Computing (MAJICC). 2021, pp. 1–8. DOI: 10.1109/MAJICC53071.2021.9526266.