M. Tech (VLSI Design and Embedded System) Amit Singh 2023

Vedic Mathematics-Driven Approach to High-Speed and Energy-Efficient ALU Design

A DISSERTATION REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE AWARD OF THE DEGREE

OF

MASTER OF TECHNOLOGY

IN

VLSI DESIGN & EMBEDDED SYSTEM

SUBMITTED BY:
Amit Singh

2K21/VLS/04

UNDER THE SUPERVISION OF

Dr. Deva Nand
ASSISTANT PROFESSOR
&
Mr. Sachin Dhariwal
ASSISTANT PROFESSOR

ELECTRONICS & COMMUNICATION ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

MAY 2023

ELECTRONICS & COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CANDIDATE’S DECLARATION

I, Amit Singh (2K21/VLS/04), student of M.Tech (VLSI Design and Embedded System), hereby declare that the Project Dissertation “**Vedic Mathematics-Driven Approach to High-Speed and Energy-Efficient ALU Design**” which is submitted by me to the Department of Electronics and Communication Engineering, Delhi technological university, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associate ship, fellowship or other similar title or recognition.

Place: Delhi Amit Singh

Date: (2K21/VLS/04)

ELECTRONICS & COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled "**Vedic Mathematics-Driven Approach to High-Speed and Energy-Efficient ALU Design**" which is submitted by **Amit Singh, Roll No. 2K21/VLS/04** to the department of Electronics and Communication Engineering, Delhi Technological University, Delhi in the partial fulfilment of the requirement for the award of the degree of Master of Technology, is record work of the report work carried out by student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi **SUPERVISOR**

Date:

Dr. Deva Nand
Assistant Professor
Department of ECE,DTU

Mr. Sachin Dhariwal
Assistant Professor
Department of ECE,DTU

ACKNOWLEDGEMENT

I would like to express my deep sense of gratitude to my highly respected and esteemed guides, **Dr. Deva Nand** and **Mr. Sachin Dhariwal** for suggesting the topic of my project and for giving me complete freedom and flexibility to work on this topic. They have been very encouraging and motivating and the intensity of encouragement has always increased with time. Without their constant support and guidance, I would not have been able to complete this work. I extend my sincere thanks to all my friends who have patiently helped me directly or indirectly in accomplishing this work successfully.

ABSTRACT

This dissertation presents a Vedic Mathematics-driven approach to designing high-speed and energy-efficient Arithmetic Logic Units (ALUs) using Xilinx Vivado. ALUs are essential components in digital systems, and the demand for improved performance and reduced power consumption continues to grow. Traditional ALU designs often suffer from complex architectures and high power consumption, limiting their efficiency.

The proposed approach leverages the principles of Vedic Mathematics, an ancient Indian system known for its simplicity and efficiency. By applying Vedic Mathematics techniques, the design methodology aims to achieve high-speed computation and reduced power consumption. Complex arithmetic operations are decomposed into simpler computations using Vedic Mathematics sutras (aphorisms), which provide efficient algorithms for addition, subtraction, multiplication, and division—core functions of an ALU. Additionally, the Vedic Mathematics-driven ALU design incorporates optimization techniques, such as clock gating, to further reduce power consumption. Clock gating selectively disables the clock signal to inactive circuit blocks, reducing unnecessary power consumption. By strategically organizing and optimizing arithmetic operations based on Vedic Mathematics principles and implementing clock gating, the proposed design achieves improved energy efficiency while maintaining high-speed performance.

The effectiveness of the approach is demonstrated through comparisons with traditional ALU designs in terms of speed and power consumption. The project was implemented using Xilinx Vivado 2020.2, and Artrix-7 FPGA for synthesis. Experimental results show notable improvements in both speed and energy efficiency compared to traditional designs, validating the effectiveness of the Vedic Mathematics-driven approach combined with clock gating.

CONTENTS

List of Tables

List of Figures

List of Abbreviations

ALU: Arithmetic Logic Unit

UTM: Urdhva Tiryagbhyam Method

Vedic: Vedic Mathematics

LSB: Least Significant Bit

XOR: Exclusive OR

AND: Logical AND

OR: Logical OR

MSB: Most Significant Bit

CPU: Central Processing Unit

HDL: Hardware Description Language

RTL: Register Transfer Level

FSM: Finite State Machine

IC: Integrated Circuit

FPGA: Field-Programmable Gate Array

SOP: Sum of Products

POS: Product of Sums

CAD: Computer-Aided Design

RTL: Register Transfer Level

PLA: Programmable Logic Array

MUX: Multiplexer

LUT: Look-Up Table

I/O: Input/Output

RTL: Register Transfer Level

CHAPTER 1: INTRODUCTION

1.1 Introduction:

In the ever-evolving landscape of computing, the demand for high-performance and energy-efficient systems has become increasingly vital. The Arithmetic Logic Unit (ALU), a important component in modern processors, plays a pivotal role in executing arithmetic and logical operations. To meet the growing demands of computation, researchers have explored various methodologies to enhance ALU performance while minimizing power consumption.

This thesis focuses on designing and implementation of a high-speed and energy-efficient ALU that leverages the principles of Vedic mathematics and clock gating techniques. Vedic mathematics, rooted in ancient Indian mathematical techniques, offers unique strategies for fast computation, while clock gating enables power-saving by selectively disabling clock signals to specific circuit components when they are not in use.The integration of Vedic mathematics in the ALU design introduces a novel approach for achieving high-speed computation. Vedic mathematical techniques, such as the Urdhav Tiryak Sutra, provide efficient algorithms for arithmetic operations, which can be adapted to enhance the ALU's performance. By exploiting the inherent parallelism and optimization possibilities within Vedic mathematics, the ALU can achieve faster computation speeds compared to conventional ALUs.

In addition to speed improvements, this thesis also addresses the critical aspect of power consumption by incorporating clock gating techniques into the ALU design. Clock gating enables the selective activation of clock signals to specific ALU components, effectively reducing power consumption by disabling unnecessary clock cycles. This power-saving mechanism is particularly beneficial in scenarios where certain ALU components are idle or not actively participating in computation.

Moreover, a key objective of this research is to achieve an overall resource utilization that is lower than that of a conventional ALU. By optimizing the ALU's architecture and design, the efficient allocation and utilization of resources can be achieved. This includes minimizing the number of gates, reducing signal propagation delays, and optimizing the use of registers and multiplexers, resulting in a resource-efficient ALU design. By combining the power of Vedic mathematics for fast computation and clock gating techniques for low power consumption, this thesis aims to develop a high-speed and energy-efficient ALU. The proposed ALU design emphasizes overall resource utilization, striving to achieve a design that maximizes performance while minimizing resource requirements.

The primary objective of this work is to advance ALU architectures by undertaking a thorough analysis, optimizing designs, and evaluating performance. The outcomes of this study will provide valuable insights and guidelines for designing ALUs that offer enhanced speed, reduced power consumption, and improved resource utilization. Ultimately, the research conducted in this thesis holds the potential to shape the future of ALU design, enabling more efficient and powerful computing systems.

1.2 Motivation:

The motivation behind this thesis stems from the ever-increasing demand for high-performance and energy-efficient computing systems in today's technology-driven world. As the complexity of computational tasks continues to grow, there is a pressing need to develop innovative solutions that can handle intensive computations while minimizing power consumption.

The Arithmetic Logic Unit , being a critical component inside processors, plays a pivotal role in executing arithmetic and logical operations. However, the performance of traditional ALUs is often limited by factors such as speed, resource utilization, power consumption . Thus, there is a strong motivation to explore novel approaches that can address these limitations and pave the way for more efficient ALU designs. The integration of Vedic mathematics and clock gating techniques in proposed ALU design offers a unique and promising solution. Vedic mathematics, with its ancient algorithms and techniques, provides a rich source of mathematical wisdom that can be harnessed to optimize computation speed. By leveraging the parallelism and optimization possibilities inherent in Vedic mathematics, the ALU can achieve significantly faster computation times compared to conventional ALUs.

Furthermore, incorporating clock gating techniques in ALU design presents an opportunity to tackle the challenge of power consumption. With clock gating, specific components of the ALU can be selectively activated or deactivated, conserving energy by minimizing unnecessary clock cycles. This power-saving mechanism aligns with the growing need for energy-efficient computing systems, enabling longer battery life in portable devices and reduced energy consumption in data centers. Additionally, the optimization of overall resource utilization is a crucial aspect in ALU design. Conventional ALUs often suffer from inefficient allocation and utilization of resources, leading to unnecessary hardware overhead and increased complexity. By focusing on resource optimization in the proposed ALU design, it is possible to achieve a more streamlined architecture with reduced gate counts, optimized signal propagation paths, and efficient usage of registers and multiplexers.

The potential impact of this research goes beyond individual ALUs. By developing a high-speed and energy-efficient ALU with improved resource utilization, various computing applications can benefit. From embedded systems and mobile devices to high-performance computing environments, the proposed ALU design can enhance computational performance, extend battery life, and reduce energy consumption.

In conclusion, the motivation behind this thesis is rooted in the pursuit of advancing ALU architectures to meet the growing demands for high-speed and energy-efficient computing systems. By integrating Vedic mathematics for fast computation, clock gating techniques for power savings, and resource optimization considerations, The aim of this research is to make a valuable contribution to the development of ALUs that offer enhanced performance, reduced power consumption, and improved resource utilization. The outcomes of this research hold the potential to revolutionize the efficiency and capabilities of computing systems, benefiting a wide range of applications and facilitating progress in various domains.

1.3 Objective:

The objectives of this thesis are centered around the development and evaluation of a high-speed and energy-efficient ALU design that incorporates Vedic mathematics and clock gating techniques. These objectives guide the research and serve as milestones to accomplish throughout the study. The key objectives of this thesis are as follows:

To conduct a comprehensive literature review: The first objective is to perform an extensive review of the existing literature on ALU design, Vedic mathematics, and clock gating techniques. This review aims to establish a solid foundation of knowledge and understanding of the historical progression, principles, and challenges associated with ALU architectures and their performance characteristics.

To explore the potential of Vedic mathematics in ALU design: This objective focuses on investigating the utilization of Vedic mathematics principles, particularly the Urdhav Tiryak Sutra algorithms, to enhance the speed and efficiency of arithmetic operations within the ALU. The aim is to adapt and integrate Vedic mathematics techniques into the ALU design to achieve faster computation speeds compared to conventional ALUs.

To incorporate clock gating techniques in the ALU design: This objective entails incorporating clock gating techniques into the ALU to achieve a reduction in power consumption. By selectively activating or deactivating clock signals to specific components based on their usage, power savings can be achieved, thereby enhancing the energy efficiency of the ALU.

To optimize overall resource utilization: This objective emphasizes the optimization of resource utilization within the ALU design. By minimizing the number of gates, optimizing signal propagation paths, and efficiently utilizing registers and multiplexers, the objective is to achieve a streamlined ALU architecture that maximizes performance while minimizing hardware overhead.

To design and implement a novel high-speed and energy-efficient ALU: This objective focuses on the actual design and implementation of the proposed ALU, incorporating the principles of Vedic mathematics and clock gating techniques. The design process involves careful consideration of the ALU components, such as the Vedic multiplier, full adder, and clock gating circuitry, to ensure seamless integration and optimal functionality.

To evaluate the performance of the proposed ALU: This objective involves the evaluation and benchmarking of the developed ALU design. Performance metrics such as computation speed, power consumption, and resource utilization will be measured and compared against conventional ALU designs. The evaluation process aims to validate the effectiveness and efficiency of the approach the proposed ALU in achieving the desired high-speed and energy-efficient characteristics.

To provide insights and guidelines for future ALU designs: The final objective of this thesis is to contribute to the body of knowledge in ALU design and provide insights and guidelines for future research and development in this field. The findings and outcomes of this research can serve as a basis for further advancements in ALU architectures, leading to more efficient and powerful computing systems.

By accomplishing these objectives, this thesis aims to make significant contributions to the field of ALU design, offering a novel approach for achieving high-speed and energy-efficient computation. The fulfilment of these objectives will provide valuable insights and guidance for researchers, engineers, and practitioners working in the area of computer architecture and digital circuit design.

1.4 Organization of the report:

Chapter 1 of the report initiates by presenting an introduction to the research topic, establishing the context for the study. It explains the purpose and significance of developing a speedy and efficient ALU design. The chapter includes a discussion on the motivation behind the research, highlighting the increasing demand for efficient computing systems. The objectives of the thesis are outlined, focusing on the exploration of integrating Vedic mathematics and clock gating techniques to enhance ALU performance. Furthermore, the organization of the report is presented, providing an overview of the subsequent chapters and their content.

Chapter 2 presents a comprehensive literature review that covers the relevant research and knowledge in the field. It examines existing literature on ALU design, Vedic mathematics, and clock gating techniques. The review provides a deep understanding of the historical development, principles, and challenges associated with ALU architectures and their performance characteristics. Additionally, it explores different types of arithmetic logic units (ALUs) and their features, including conventional ALUs, ALUs with clock gating, and ALUs with Vedic multipliers. The chapter critically analyses and synthesizes the existing knowledge to lay the foundation for the subsequent chapters.

Chapter 3 focuses implementation of a novel high-speed and energy-efficient ALU design. It describes the design and operation of the proposed ALU, with an emphasis on integrating Vedic mathematics and clock gating techniques. The chapter discusses the design considerations and implementation details of the Vedic multiplier, full adder, and overall ALU architecture. It explains how these components contribute to achieving high-speed computation and energy efficiency. The design choices made to optimize performance and resource utilization are presented, highlighting the innovation and novelty of the proposed ALU design.

In Chapter 4, the simulation results and analysis of the developed ALU design are presented, offering a comprehensive assessment of its performance in key areas such as computation speed, power consumption, and resource utilization. The chapter discusses the simulation methodology, presents the obtained results, and conducts a comprehensive analysis of the speed improvement, power improvement, and resource utilization achieved by the proposed ALU design. The findings are critically discussed and compared against conventional ALU designs, highlighting the strengths and advantages of the novel ALU architecture.

Chapter 5 offers a summary of the main discoveries and conclusions derived from the research. It revisits the objectives outlined in Chapter 1 and assesses their accomplishment. The chapter discusses the implications and contributions of the research, emphasizing the advancements made in high-speed and energy-efficient ALU design through the integration of Vedic mathematics and clock gating techniques. Additionally, the report outlines the study's limitations and proposes potential directions for future research and improvement in ALU architectures.

The References section lists all the sources cited throughout the report, providing proper citations for further reading and validation of the research.
The Appendices section includes additional materials that support the report, such as the Verilog code for the conventional ALU, clock-gated ALU, Vedic multiplier ALU, and the novel ALU. These appendices provide technical details and implementation specifics for reference and replication purposes.

CHAPTER 2: LITERATURE REVIEW AND ALU TYPES

2.1 Literature Review

Arithmetic Logic Units (ALUs) play a crucial role in modern computing systems, performing various arithmetic and logical operations. To fulfill the increasing demands for high-speed and energy-efficient computation, researchers have explored innovative approaches to ALU design, including the integration of Vedic mathematics and clock gating techniques. In this literature review, we delve into the existing research in this field to gain insights into the historical development, principles, and challenges associated with high-speed and energy-efficient ALU designs.

Vedic mathematics, derived from ancient Indian mathematical texts, offers alternative computational techniques that can potentially enhance the performance of ALUs. One notable algorithm from Vedic mathematics is the Urdhav Tiryak Sutra, which enables faster multiplication operations. N. Gadda et al. (2020) investigated the application of the Urdhav Tiryak Sutra in ALU architectures. Their study demonstrated that the integration of Vedic multiplication algorithms led to improved multiplication performance compared to traditional methods. The integration of Vedic mathematics into ALU architectures brings forth a range of benefits that contribute to faster and more efficient computation. One key advantage lies in the reduced propagation delay achieved through the application of Vedic multiplication algorithms. Traditional multiplication methods often involve a series of iterative steps, leading to longer propagation delays and slower computation. However, the Urdhav Tiryak Sutra, based on Vedic mathematics, presents a more streamlined approach that reduces the number of intermediate steps, resulting in minimized propagation delay.[2]

Additionally, the incorporation of Vedic mathematics fosters increased parallelism within the ALU design. Parallelism refers to the simultaneous execution of multiple operations, and it is a crucial factor in enhancing computation speed. By leveraging the principles of Vedic mathematics, ALUs can exploit parallel processing capabilities more effectively, enabling the execution of multiple arithmetic operations concurrently. This heightened parallelism translates into significant time savings, as the ALU can handle complex calculations in a highly efficient and parallelized manner. These techniques, derived from ancient Indian mathematical texts, provide unique algorithms and methods that optimize computation and reduce the complexity of traditional approaches. By

leveraging these alternative techniques, ALUs can further enhance their efficiency and precision in performing a wide range of arithmetic operations.[7]
In the pursuit of energy-efficient computing systems, minimizing power consumption has become a critical objective. One area of focus in ALU design is the implementation of clock gating techniques, which have been extensively investigated as a means of optimizing power efficiency. Clock gating involves selectively controlling the clock signal to different parts of the circuitry based on their input and output dependencies.[6]

A study conducted by M. Satte et al. (2021) delved into the realm of clock gating schemes specifically designed for ALUs. The researchers aimed to demonstrate the effectiveness of their proposed technique in reducing power consumption while maintaining the ALU's performance. The core idea behind their approach was to minimize unnecessary clock toggling, thereby conserving power without compromising the functionality of the ALU.[4]
The clock gating scheme successfully mitigated dynamic power consumption in the ALU by selectively enabling or disabling the clock signal to specific circuitry.This dynamic power reduction stemmed from the elimination of unnecessary clock transitions, which typically occur when clock signals propagate through unused or idle components. Through careful analysis and design considerations, the researchers were able to identify and exploit opportunities for clock gating within the ALU architecture. The experimental results showcased the significant power savings achieved with the clock gating technique. By reducing the power consumed by idle or non-essential circuitry, the overall power consumption of the ALU was notably reduced. This reduction in power consumption directly translated into improved energy efficiency, contributing to the goal of creating energy-efficient computing systems.[4]

Importantly, the power savings achieved through clock gating were achieved without sacrificing the ALU's performance. The researchers ensured that the proposed clock gating scheme did not introduce any adverse effects on the ALU's functionality, timing, or computation speed. This was achieved through meticulous design and verification processes to guarantee that the enabled clock signals accurately synchronized the ALU's operations and preserved data integrity.[3]

Overall, the study demonstrated the effectiveness of clock gating techniques in reducing power consumption in ALU designs. The proposed clock gating scheme successfully minimized unnecessary clock toggling, resulting in significant power savings without compromising the ALU's performance. These findings highlight the potential of clock gating as a viable approach to enhance the energy efficiency of ALUs and contribute to the advancement of power-efficient computing systems.

Conventional ALU designs have been the subject of extensive research to understand their performance characteristics and trade-offs. A. Alrashdi et al. (2022) conducted a comparative analysis of different ALU architectures, including ripple-carry and carry-lookahead adders. Their study examined metrics such as computation speed, power consumption, and resource utilization. The findings revealed that carry-lookahead adders offered faster computation at the expense of increased power consumption and resource utilization. Such insights are crucial for understanding the design trade-offs in conventional ALUs[1].

2.2 Arithmetic Logic Unit (ALU)

The Arithmetic Logic Unit (ALU) serves as a crucial component within a computer's central processing unit (CPU), undertaking arithmetic and logical operations on binary data. It serves as the computational powerhouse of the CPU, enabling the execution of various mathematical calculations and logical decisions necessary for data processing. The main objective of an ALU is to execute various arithmetic operations. These operations are executed on binary inputs, typically represented as a series of bits (0s and 1s), and produce binary outputs based on the specified operation.

The ALU consists of a combination of digital logic circuits, registers, and multiplexers designed to efficiently carry out these operations. It operates on binary data in parallel, with the word size determining the number of bits it can process simultaneously. Common word sizes for ALUs include 8-bit, 16-bit, 32-bit, and 64-bit, among others. In addition to basic arithmetic and logical operations, ALUs often incorporate additional features to support more complex computations. These features can include shifting operations (bitwise left or right shift), comparison operations (less than, greater than, equal to), and conditional operations (if-then-else), allowing the ALU to handle a wide range of computational tasks.

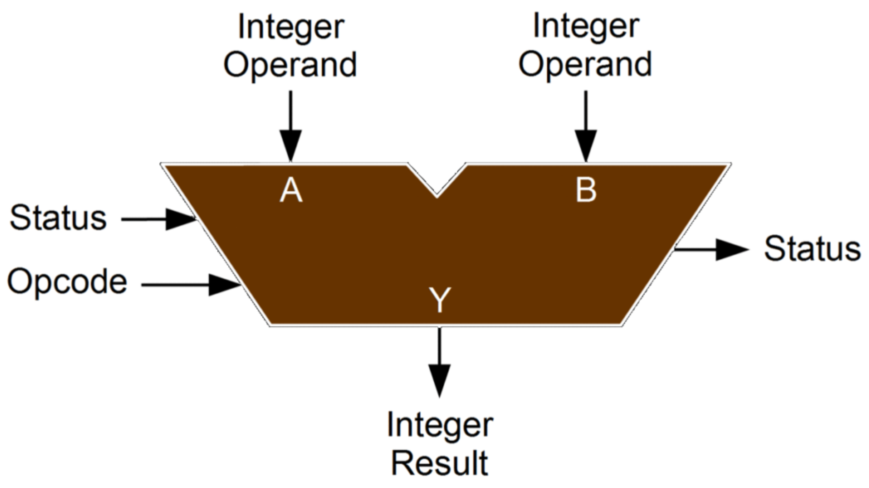


Fig 2.1: Block diagram of ALU [2]

The design of an ALU involves careful consideration of factors such as performance, power consumption, area utilization, and architectural choices. ALUs can be implemented using various design methodologies and architectures, each with its own trade-offs and optimizations. Designers strive to achieve a balance between power efficiency, speed and area utilization to meet the specific requirements of the target computing system.

The speed and efficiency of ALUs directly influence the execution time of computational tasks and the overall responsiveness of the system. Consequently, researchers and engineers continuously explore innovative techniques and optimizations to enhance ALU performance, reduce power consumption, and improve overall efficiency.

In summary, an Arithmetic Logic Unit (ALU) is a fundamental component of a CPU responsible for executing arithmetic and logical operations on binary data. Its design, which includes digital logic circuits and various supporting features, enables efficient and accurate computation. The performance, power consumption, and architectural choices of an ALU significantly impact the overall performance and efficiency of a computer system, making it an area of continuous research and optimization.

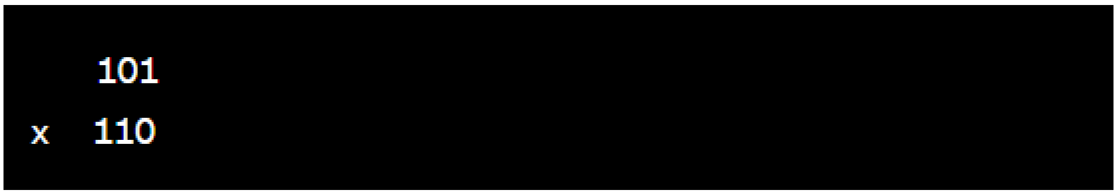
2.3 Urdhav Tiryagbhyam method

The Urdhva Tiryagbhyam method is a prominent technique in Vedic mathematics, an ancient Indian system of mathematics that dates back thousands of years. Vedic mathematics is believed to have originated from the Vedas, the ancient scriptures of India. The system was developed by ancient scholars known as "Rishis" who codified mathematical concepts and techniques in concise and systematic ways.

The Urdhva Tiryagbhyam method is specifically used for efficient multiplication of two-digit numbers mentally. It relies on the principles of place value and symmetry to simplify the calculations. The name "Urdhva Tiryagbhyam" itself translates to "Vertical and Crosswise" or "Vertically and Horizontally" in Sanskrit, referring to the movement of the digits during the multiplication process.

Let's multiply 101 by 110 using the urdhva triyakbhyam method.

Step 1: Write down the two numbers:



```
  101
x 110
```

Fig 2.2: First step of urdhva triyakbhyam multiplication

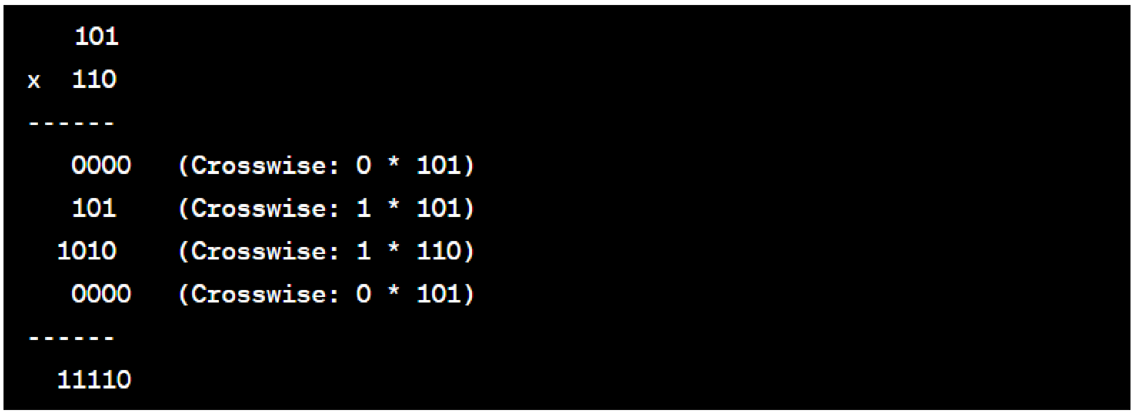
Step 2: Perform the vertical and crosswise calculations.

Multiply the units place (0) of the second number by the first number (101). Write down the result (0000) under the line.

Multiply the tens place (1) of the second number by the first number (101). Write down the result (101) below the line.

Multiply the crosswise numbers (1 from 110 and 1 from 101). Write down the result (1010).

Multiply the crosswise numbers (1 from 110 and 0 from 101). Write down the result (0000).



```
   101
x  110
------
   0000   (Crosswise: 0 * 101)
   101    (Crosswise: 1 * 101)
  1010    (Crosswise: 1 * 110)
   0000   (Crosswise: 0 * 101)
------
  11110
```

Fig 2.3: Second step of urdhva triyakbhyam multiplication

Step 3: Add the results obtained in Step 2.

0000 + 101 + 1010 + 0000 = 1111.

Step 4: Combine the results to get the final answer.

The final answer is 11110 (101 multiplied by 110 using urdhva triyakbhyam).

So, multiplying 101 by 110 using urdhva triyakbhyam gives us the result of 11110.

The Urdhav Tiryak Sutra technique eliminates the need for carrying over digits during the multiplication process, making it faster and more efficient. It breaks down the multiplication into simpler vertical and crosswise operations, allowing for parallel calculations and reducing the overall number of steps required.

By leveraging the parallel nature of the Urdhav triyakbhyam Sutra, the multiplication operation can be executed more quickly compared to traditional methods. This algorithm is particularly beneficial when performing multiplication of large numbers or when designing high-speed computing systems where efficiency and speed are crucial.

Overall, the Urdhav triyakbhyam Sutra provides an alternative approach to multiplication that simplifies the process and enables faster computation. Its parallel nature and elimination of carry-over digits contribute to improved multiplication performance in various computational contexts.

2.4 Clock Gating Technique

Clock gating is a power management and digital circuit design technique employed to decrease power consumption by regulating the clock signal provided to specific circuit

elements. It involves selectively enabling or disabling the clock signal to specific components or sections of a circuit based on certain conditions or events.
In a digital system, the clock signal is often used as a global timing reference, synchronizing the operation of various components within the system. However, not all components need to be active and consuming power at all times. Clock gating provides a means to dynamically control the clock signal, allowing certain circuit elements to be "gated off" when they are not required to perform their tasks. By disabling the clock signal to inactive components, unnecessary switching and power consumption can be minimized.

Clock gating is typically implemented using specialized circuitry called "clock gating cells" or "clock gates." These gates are inserted in the clock path of the circuit and act as switches that either allow or block the propagation of the clock signal to specific destinations. The decision to enable or disable the clock signal to a particular circuit element is based on control signals or conditions determined by the system's logic.
When a clock gating cell receives a control signal indicating that the associated circuit element is inactive or does not require a clock pulse, it blocks the clock signal, preventing it from reaching that element. As a result, the component remains in a "gated off" state, consuming minimal power and reducing unnecessary switching activity. Dynamic power consumption in a digital circuit is primarily attributed to the clock tree, which constitutes over 50% of the overall power consumed. This power can be attributed to three main components:
1. Power consumed by the combinatorial logic: The combinatorial logic elements in the design, which change their values on each clock edge, contribute to power consumption. This is because of the switching activity and the associated dynamic power dissipation.

2. Power consumed by flip-flops: Flip-flops, which are used as sequential elements for storing and propagating data, consume power even when they are not actively transitioning their state. The extensive utilization of flip-flops in digital circuits contributes significantly to the dynamic power consumption, making it a substantial portion of the overall power usage.

3.Power consumed by the clock buffer tree: The clock buffer tree, which is responsible for distributing the clock signal throughout the design, consumes power as well. The power consumption in the clock buffer tree arises from the buffers themselves and the wiring capacitance they drive, which contribute to both static and dynamic power dissipation.

Collectively, these three elements constitute over 50% of the dynamic power consumption in a design, with the clock tree holding a significant position. To optimize power in a design, various techniques like clock gating, power gating, and clock network synthesis are employed. These techniques aim to minimize power consumption in the combinatorial logic, flip-flops, and the clock buffer tree, ultimately resulting in power-efficient designs as a whole.

Latch Free Clock Gating

A clock gating technique known as latch-free clock gating utilizes a basic AND or OR gate, depending on the flip-flop trigger edge, to control the clock signal. However, this style poses limitations where the enable signal's inactivity during a clock pulse or multiple fluctuations can lead to premature termination of the gated clock output or the generation of multiple clock pulses. Considering our design's reliance on single-clock flip-flops, this clock gating style is deemed unsuitable due to these constraints.

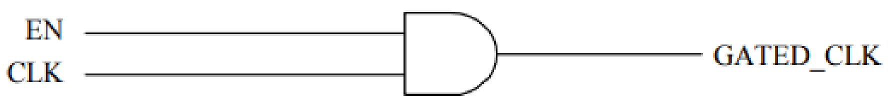


Fig 2.4: Latch free clock gating circuit [3]

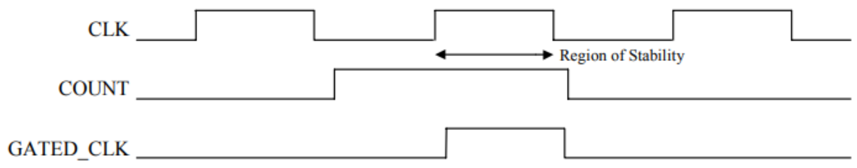


Fig 2.5: Operation of Latch Free Clock Gating Circuit [3]

Latch Based Clock Gating

In the latch-based clock gating style, a level-sensitive latch is integrated into the design to preserve the enable signal from the active edge of the clock until the inactive edge of the clock. By incorporating this latch, the enable signal's state is captured and preserved throughout the entire duration of the clock pulse.

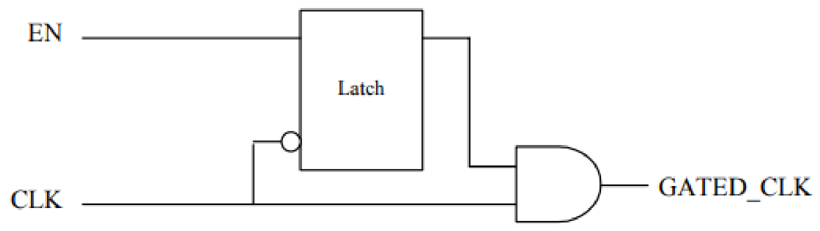


Fig 2.6: Latch based clock gating circuit [3]

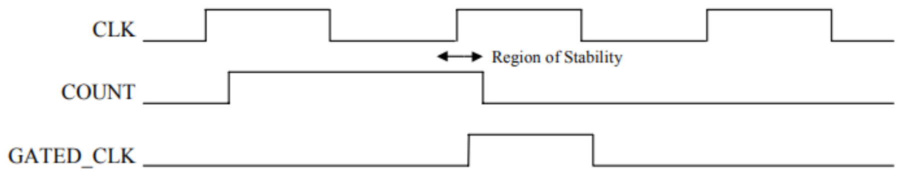


Fig 2.7: Operation of Latch based Clock Gating Circuit [3]

Because of this latch-based mechanism, the stability of the enable signal is required only during the rising edge of the clock, similar to the conventional ungated design approach. The integration of the latch guarantees that subsequent changes during the clock pulse do not impact the enable signal, allowing it to maintain its state until the entire clock pulse is generated.

2.6 Conventional ALU

The traditional ALU comprises two distinct sub-modules known as the " Arithmetic Unit" and the "Logical Unit." These components collectively perform arithmetic and logical operations based on the designated ALU control input.

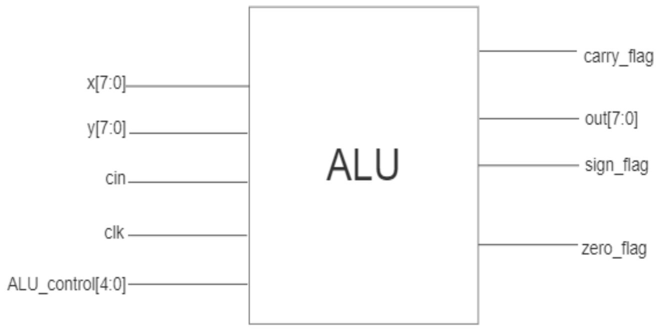


Fig 2.8 Block diagram of conventional ALU

The ALU module has the following inputs:

- clk: Clock signal
- x: 8-bit input operand
- y: 8-bit input operand
- cin: Carry-in signal
- ALU_control: 5-bit control signal which specify which operation to be performed

And the following outputs:

- zero_flag: Output flag indicating if the result is zero
- sign_flag: Output flag indicating if the result is negative
- out: 8-bit output result of ALU operation
- carry flag: Output carry flag (used only for addition operation)

The ALU module consists of two sub-modules:

Arithmetic unit: This module performs arithmetic operations (addition, subtraction, multiplication). It instantiates three sub-modules: AdderRipple (for addition), sub (for subtraction), and mult (for multiplication). The results from these sub-modules are stored in wires: add_result, sub_result, and mul_result. The carryout from the AdderRipple module is also stored in the wire carryout. The ALU_control input determines which result is selected for the output. In the always block, based on the ALU_control value, the appropriate result is assigned to the "out" output and carryout is assigned to the "cout" output.

Logical unit: This module performs logical operations (bitwise operations and logical comparisons). By taking inputs "a" and "b," the ALU executes the designated logical operations according to the value of "ALU_control." The outcome is then stored in the logical_output register. Within the always block, the suitable logical operation corresponding to the ALU_control value is performed, and the resulting value is assigned to "logical_output" output.

The ALU module combines the results from the Arithmetic unit and logical unit sub-modules based on the ALU_control input. Within the first always block, the "out" output is determined by selecting the suitable result based on the value of "ALU_control." Additionally, the carry flag is assigned accordingly. In the second always block, the zero_flag is set if the output is equal to zero, while the sign_flag is set if the output is negative.

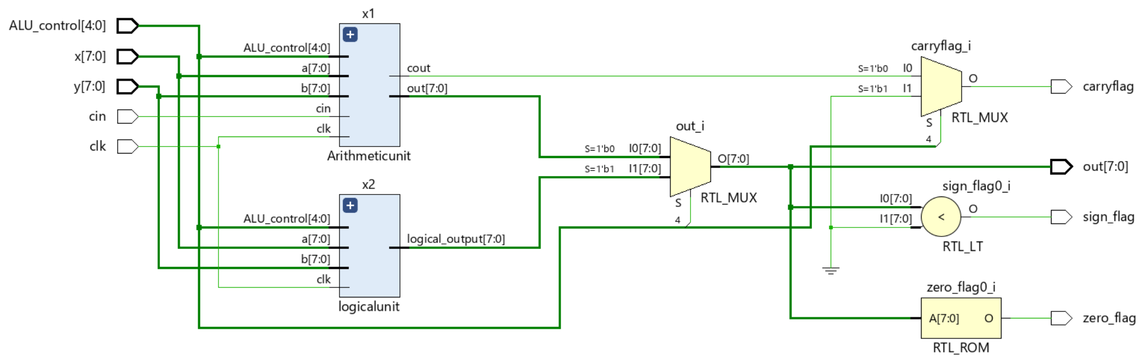


Fig 2.9 RTL Schematic of the ALU

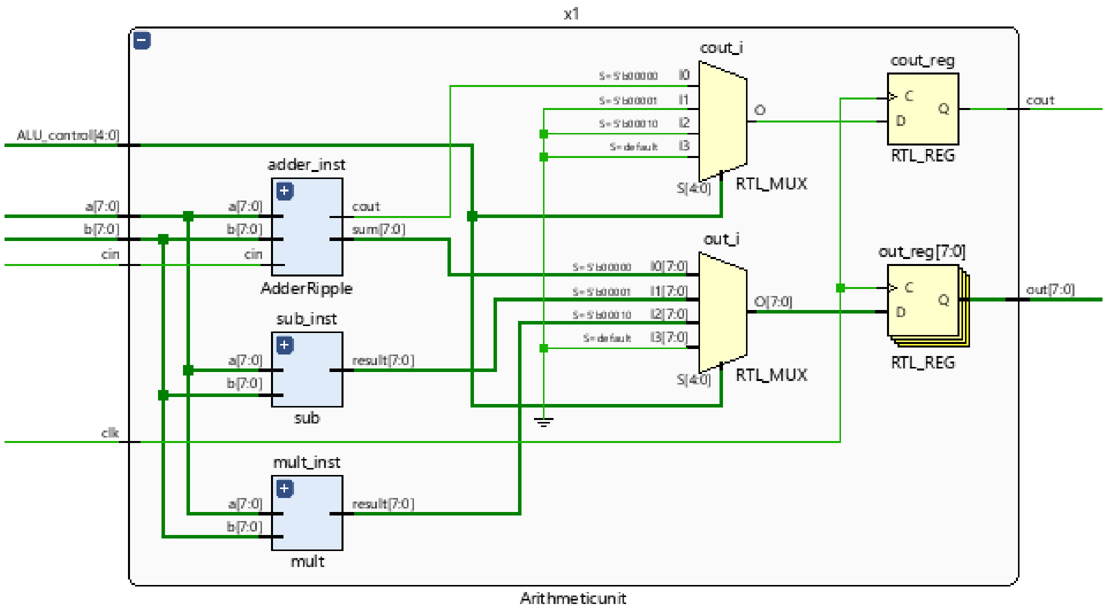


Fig 2.10 RTL Schematic of the Arithmetic unit inside ALU

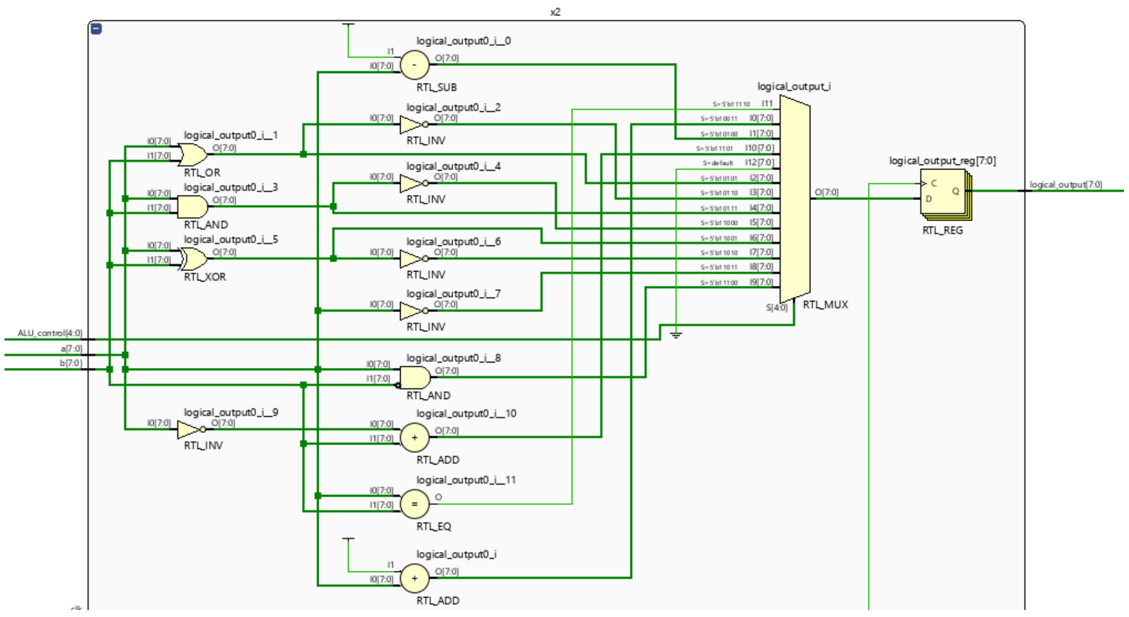


Fig 2.11 RTL Schematic of the Logical unit of ALU

Overall, this ALU module can perform arithmetic operations (addition, subtraction, multiplication) and logical operations (bitwise operations and logical comparisons) based on the ALU_control input. The outputs include the result, flags indicating zero and sign, and carry flag (used only for addition).

ALU Control	Function	Output
00000	Add	A+B
00001	Subtract	A-B
00010	Multiply	A*B
10011	Increment	A+1
10100	Decrement	A-1
10101	Bitwise OR	A|B
10110	Bitwise NOR	~(A|B)
10111	Bitwise AND	A&B
11000	Bitwise NAND	~(A&B)
11001	Bitwise XOR	A^B

11010	Bitwise XNOR	~(A^B)
11011	Bitwise NOT	~A
11100	Bitwise AND with complement	A&(~B)
11101	Complement and add	~A+B
11110	Equality comparison	A==B

Table 2.1 Functions provided by ALU

2.7 ALU with clock gating

This ALU has all the functions which a conventional ALU can provide, but here an extra feature of clock gating is added.

The provided ALU (Arithmetic Logic Unit) module has been devised to execute a range of arithmetic and logical operations, guided by input signals and control signals. It comprises three primary sub-modules: Arithmetic unit, logical unit, and clock gating. The Arithmetic unit module manages arithmetic operations like addition, subtraction, and multiplication. It accepts two 8-bit inputs (a and b), a carry-in signal (cin), and an ALU control signal (ALU_control). Within the module, it employs an AdderRipple module for addition, a sub module for subtraction, and a mult module for multiplication. The output of the chosen operation is stored in the out register, while the carry-out is stored in the cout register. The logical unit module handles logical operations such as bitwise OR, AND, XOR, and complement, along with performing increment, decrement, and comparison operations. It takes two 8-bit inputs (a and b) and an ALU control signal (ALU_control). The ALU_control signal determines the operation to be executed. The output of the selected operation is stored in the logical_output register. The clock gating module facilitates clock gating, which is a power-saving technique. It receives an input clock signal (clk_in) and an enable signal (enable), generating an output clock signal (clk_out) based on the enable signal. When ALU_control is 0, clk_out1 is enabled, activating the arithmetic unit (x1). Conversely, when ALU_control is 1, clk_out2 is enabled, activating the logical unit (x2).

In the main ALU module, the outputs of the Arithmetic unit and logical unit modules are selected based on the ALU_control signal. If ALU_control is 0, the output of the arithmetic unit is selected (out and carryflag). If ALU_control is 1, the output of the logical unit is selected (out). The zero_flag is set if the output is zero, and the sign_flag is set if the output is negative.

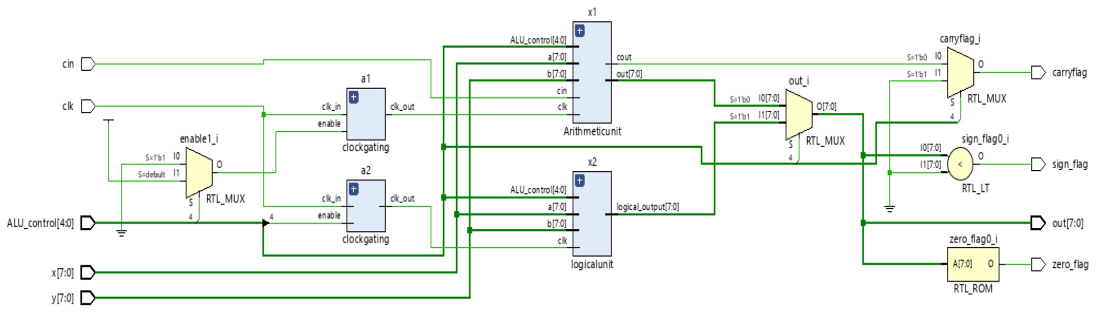


Fig 2.12 RTL Schematic of the clock gating based ALU

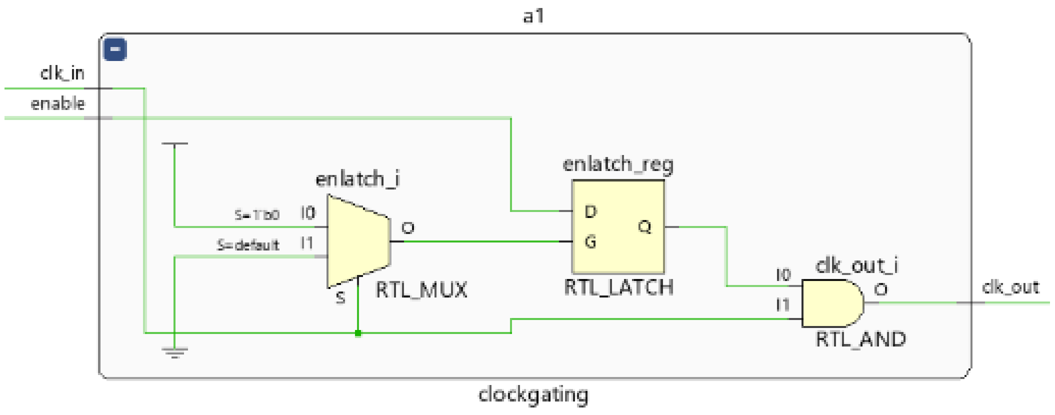


Fig 2.13 RTL Schematic of the clock gating

The clock gating module consists of a flip-flop (enlatch) and a combinational logic block. When input clock signal (clk_in) is low (0), the combinational logic evaluates the enable. If the enable is high (1), indicating that the module should be enabled, the flip-flop (enlatch) is updated with the enable value. The output clock signal (clk_out) is then generated by logically ANDing the enable signal (enlatch) with the input clock signal (clk_in).

In the ALU module, two instances of the clock gating module, a1 and a2, are instantiated. a1 is used to gate the clock signal (clk) going into the Arithmetic unit module (x1), while a2 gates the clock signal going into the logical unit module (x2). The enable signal for a1 is derived from the negation of ALU_control, while the enable signal for a2 is directly ALU_control.

By controlling the clock signals of the modules using clock gating, the ALU can selectively activate and deactivate specific functional units based on the value of ALU_control. This helps reduce power consumption by avoiding unnecessary clock cycles and computations when certain operations are not needed.

2.8 ALU with Vedic Multiplier

The ALU based on the Urdhva Tiryagbhyam method of Vedic multiplication is designed to efficiently perform logical and arithmetic operations using the Urdhva Tiryagbhyam technique. The Urdhva Tiryagbhyam method, derived from Vedic mathematics, offers an alternative approach to traditional multiplication algorithms.
When operating in the multiplication mode, the ALU takes two input operands, represented as binary numbers, and a control signal indicating the desired operation.

The operands are divided into smaller subparts, such as 2-bit or 4-bit segments, depending on the specific implementation.
For each subpart, the Urdhva Tiryagbhyam method is applied. The method involves breaking down the subparts into smaller units, typically 2x2 bits or 4x4 bits, and performing parallel multiplications. This is done by computing the bitwise products of the corresponding bits in the subparts using AND gates.
The partial products obtained from each subpart multiplication are then combined using addition operations. The addition can be performed using specialized adders, such as the ripple-carry adders, or other adder configurations depending on the specific design requirements.

By decomposing the multiplication process into smaller steps and utilizing parallelism, the ALU based on the Urdhva Tiryagbhyam method achieves faster and more efficient multiplication compared to traditional methods. It takes advantage of the parallel nature of the Urdhva Tiryagbhyam technique and optimizes hardware utilization to improve performance.

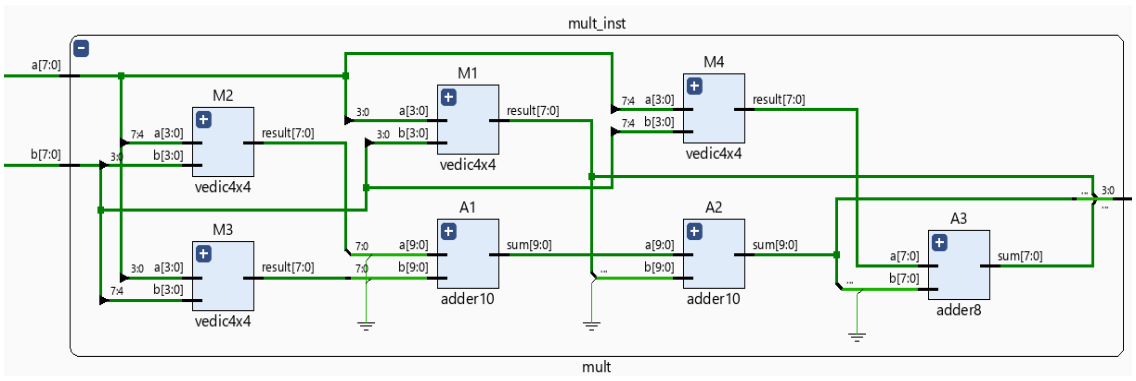


Fig 2.14 RTL Schematic of the Vedic Multiplier

The ALU may include additional features such as carry propagation, sign detection, zero detection, and logical operations like bitwise AND, OR, XOR, etc. These features enable the ALU to do a wide range of arithmetic and logical operations efficiently.

The advantages of using the Urdhva Tiryagbhyam method in an ALU include reduced complexity, faster computation, and improved efficiency. By breaking down the multiplication process into smaller steps, the ALU can exploit parallelism and optimize the use of hardware resources, resulting in improved performance.

Utilizing the principles of ancient Indian mathematics, an ALU incorporating the Urdhva Tiryagbhyam method of Vedic multiplication presents an alternative approach to executing efficient logical and arithmetic operations.

CHAPTER 3: NOVEL HIGH SPEED AND ENERGY EFFICIENT ALU DESIGN

3.1 Multiplier Design

The Vedic multiplication algorithm implementation provides a more detailed and modular approach to multiplication. It decomposes the multiplication process into smaller steps, allowing for better control and understanding of the operations involved. By utilizing the Vedic multiplication technique, it aims to provide improved speed and efficiency compared to conventional multiplication methods.

In contrast, the simple multiplication we used is a more straightforward implementation that directly uses the built-in multiplication operator in Verilog. It offers a simpler and more concise approach, especially when the focus is on functionality rather than the intricate details of the multiplication algorithm.

The Vedic multiplier hierarchical design allows for better organization and modularization of the different stages involved in the Vedic multiplication algorithm. It encapsulates the functionality of each stage within separate submodules, making the design more modular, reusable, and easier to understand and maintain. It also provides flexibility in terms of modifying or optimizing specific stages of the multiplication algorithm independently.

On the other hand, the simple multiplication sacrifices the modularity provided by the hierarchical design. It provides a more compact implementation by directly using the multiplication operator, but it may not be as flexible or easily adaptable to modifications or enhancements in the multiplication algorithm.

Ultimately, the choice between the two approaches depends on the specific requirements of the design. If the goal is to explore and understand the Vedic multiplication algorithm and optimize its performance, the first module with its hierarchical design and detailed implementation would be more suitable. However, if the focus is on simplicity, functionality, and a faster design turnaround, the second module's direct use of the multiplication operator can be a more practical choice.

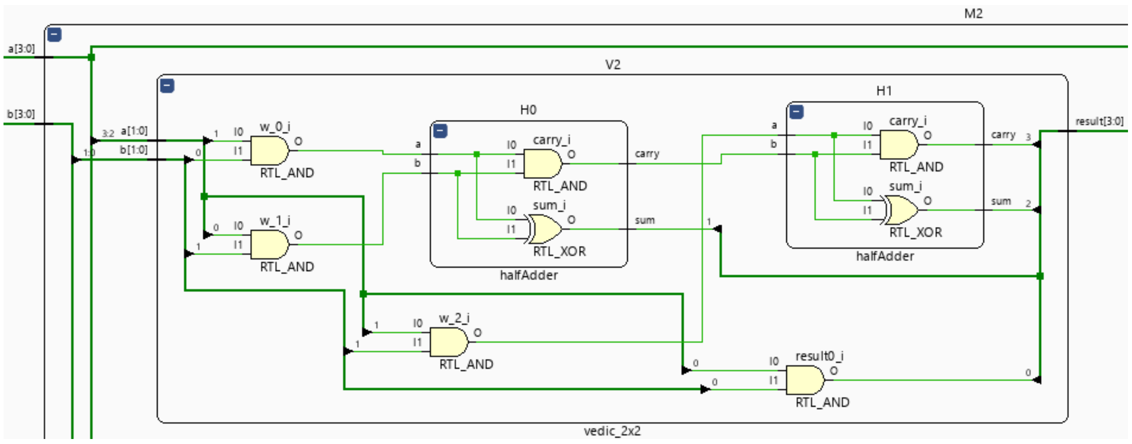


Fig 3.1 RTL Schematic of the Vedic Multiplier 2*2 block

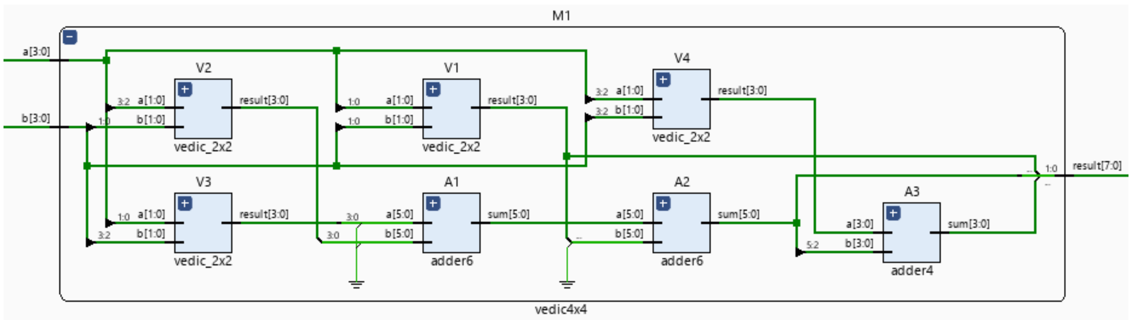


Fig 3.2 RTL Schematic of the Vedic Multiplier 4*4 block

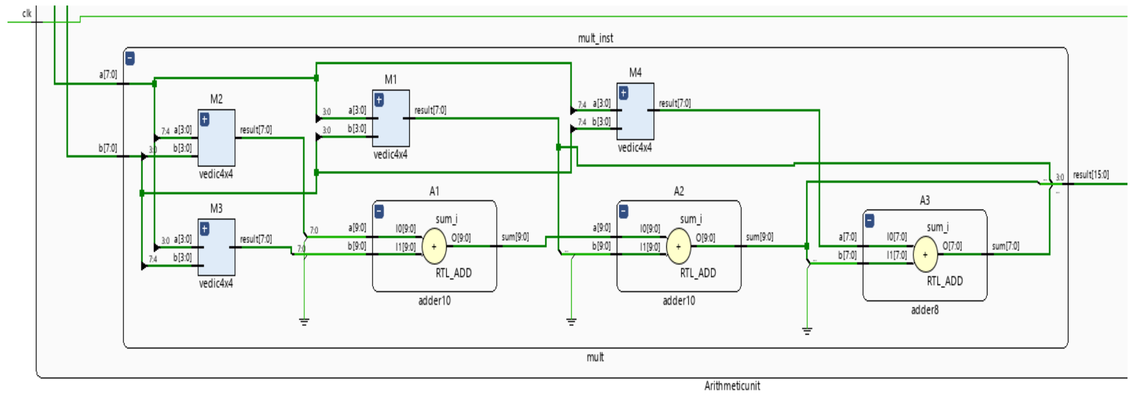


Fig 3.3 RTL Schematic of the Vedic Multiplier with detailed adder block

Let's break down the blocks of the Vedic multiplier in detail:

- **vedic_2x2 module:**

The vedic_2x2 module comprises two half-adders and four AND gates. Each AND gate calculates the bitwise product of the corresponding bits from input signals "a" and "b." Specifically, the four AND gates compute the following products:

a[0] & b[0]: Bitwise product of least significant bit of "a" and least significant bit of "b."

a[1] & b[0]: Bitwise product of the second least significant bit of "a" and least significant bit of "b."

a[0] & b[1]: Bitwise product of least significant bit of "a" and the second least significant bit of "b."

a[1] & b[1]: Bitwise product of second least significant bit of "a" and second least significant bit of "b."

These bitwise products are then fed into two half-adders, which perform the addition of the corresponding pairs of products. The outputs of the two half-adders are combined to generate the final 4-bit result of the multiplication.

- **vedic4x4 module:**

To perform the multiplication, the module utilizes four instances of the vedic_2x2 module. Each vedic_2x2 module is responsible for multiplying two 2-bit parts of the inputs a and b. To maximize efficiency, the module divides the 4-bit inputs into two 2-bit parts, allowing the utilization of the vedic_2x2 multiplier's capabilities at the 2-bit level.

The vedic_2x2 modules calculate the partial products for each pair of 2-bit parts. These partial products are then added together using two 6-bit adders, known as adder6. The adders take the partial products and perform addition to generate a 6-bit sum. The 6-bit sum represents the intermediate result of the multiplication.

- **adder4 module:**

The module presented here is a 4-bit adder that takes two 4-bit inputs, "a" and "b," and generates a 4-bit sum. It utilizes a straightforward bit-wise addition method to calculate the sum of the inputs. The addition process occurs bit by bit, starting from the least significant bit to the most significant bit. At each bit position, an XOR gate (represented by the ^ operator) computes the sum bit, while an AND gate (represented by the & operator) calculates the carry bit.

To begin, the XOR gate adds the LSBs of "a" and "b" to produce the LSB of the sum. Simultaneously, an AND gate computes the carry generated from adding the LSBs. This carry bit propagates to the subsequent bit position.This process is repeated for each subsequent bit position, where the corresponding bits of "a" and "b" are added using XOR gates, and the carries from the previous bit positions are added using AND gates. By iteratively applying this bit-wise addition process, the module determines the sum of the 4-bit inputs "a" and "b." The resulting 4-bit sum is generated as the module's output.
This type of adder is commonly known as a ripple carry adder since the carry bit ripples through each bit position, introducing a delay in sum computation. It serves as a fundamental component for addition operations in digital circuits and finds widespread application.

- **adder6 module:**

The module represents a 6-bit adder, which takes two 6-bit inputs, a and b, and produces a 6-bit sum. The adder follows a similar principle to the adder4 module but operates on 6-bit inputs instead.

The addition process in the adder6 module is performed bit by bit, starting from the LSB to the most significant bit MSB. For each bit position, the module uses an XOR gate to calculate the sum and an AND gate to calculate the carry. By applying this bit-wise addition process iteratively, the module computes the sum of the 6-bit inputs a and b. The resulting 6-bit sum is produced as the output of the module.
Similar to the adder4 module, the adder6 module is also a ripple carry adder, where the carry ripples through each bit position, resulting in a delay in the computation of the sum.

It is a fundamental building block for addition operations in digital circuits, particularly when dealing with larger bit-width inputs.

- **adder8 module:**

The module presented here represents an 8-bit adder, which accepts two 8-bit inputs, "a" and "b," and generates an 8-bit sum. It follows the same principle as the previous adder modules but operates on 8-bit inputs.
Similar to the adder4 and adder6 modules, the addition process in the adder8 module is carried out bit by bit, starting from the LSB to the most significant bit MSB. At each bit position, an XOR gate calculates the sum bit, while an AND gate determines the carry bit.
To begin, the XOR gate combines the LSBs of "a" and "b" to produce the LSB of the sum. Concurrently, an AND gate calculates the carry generated by adding the LSBs. This carry bit is then propagated to the subsequent bit position.
This process is repeated for each subsequent bit position, where the corresponding bits of "a" and "b" are added using XOR gates, and the carries from the previous bit positions are incorporated using AND gates.
By iteratively applying this bit-wise addition process, the module computes the sum of the 8-bit inputs "a" and "b." The resulting 8-bit sum is generated as the module's output.
The adder8 module functions as a ripple carry adder, where the carry bit ripples through each bit position, introducing a delay in the sum computation. It serves as a foundational component for addition operations in digital circuits, particularly when dealing with larger bit-width inputs.

- **adder10 module:**

The module presented here is an adder10, which functions as a 10-bit adder. It takes two 10-bit inputs, "a" and "b," and produces a 10-bit sum. Following the same principle as the previous adder modules, it performs bit-wise addition to calculate the sum.
Similar to the adder4, adder6, and adder8 modules, the addition process in the adder10 module is executed bit by bit, starting from the LSB to the most significant bit MSB. At each bit position, an XOR gate calculates the sum bit, while an AND gate determines the carry bit.

The LSBs of "a" and "b" are added using an XOR gate, yielding the LSB of the sum. Concurrently, an AND gate computes the carry generated by adding the LSBs. This carry bit is then propagated to the next bit position.

This process is repeated for each subsequent bit position, where the corresponding bits of "a" and "b" are added using XOR gates, and the carries from the previous bit positions are incorporated using AND gates.

Furthermore, the mult module combines all the aforementioned submodules to construct the complete Vedic multiplier. By utilizing a combination of the vedic4x4, adder10, adder8, and adder6 modules, it performs multiplication of two 8-bit inputs, "a" and "b." Intermediate results are passed between the submodules to generate the final 8-bit output.

In summary, the adder10 module performs bit-wise addition to compute the sum of 10-bit inputs "a" and "b," while the mult module leverages a combination of submodules to implement the complete Vedic multiplier for the multiplication of two 8-bit inputs.

3.2 Full Adder Design

Ripple carry adder has been used in the traditional ALU to perform binary addition of multiple bits. It is a simple and straightforward implementation of an adder, but it has a drawback in terms of propagation delay.

A ripple carry adder processes the bits of two numbers being added in a sequential manner, starting from the least significant bit (LSB) and propagating the carry from one stage to the next. Each bit position incorporates a full adder that considers the corresponding bits of the input numbers and the carry from the previous stage. The carry output from each stage becomes the carry input for the subsequent stage, creating a ripple effect throughout the adder.

The primary advantage of a ripple carry adder lies in its simplicity and straightforward implementation. It relies on basic logic gates and full adders, making it relatively easy to construct. However, the ripple carry adder suffers from a significant drawback, which is the propagation delay. Since the carry signal needs to propagate through each stage, the overall delay of the adder is determined by the longest path of carry propagation. Consequently, the propagation delay increases proportionally with the number of bits in the adder.

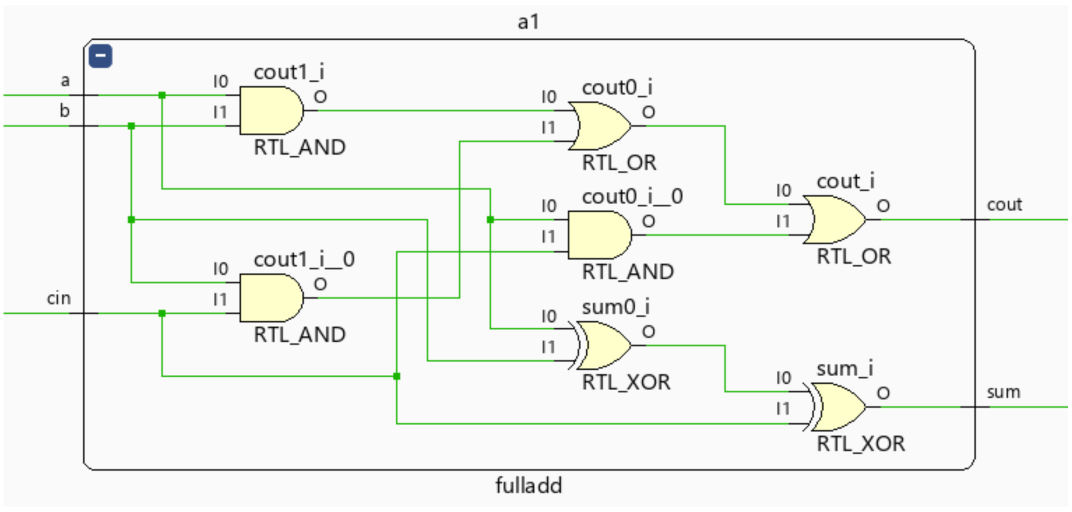


Fig 3.4 RTL Schematic of Full Adder Adder block

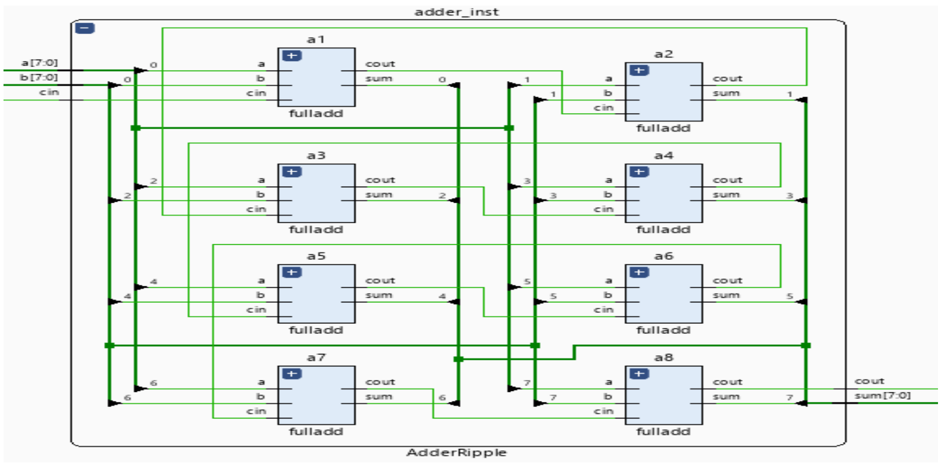


Fig 3.5 RTL Schematic of Ripple Carry Adder block

The module describes a 8-bit ripple carry adder module (Adder Ripple) and a full adder module (fulladd) used within it. Here's a detailed description of each module:

1. **Adder Ripple module:**

- This module represents an 8-bit ripple carry adder. It takes two 8-bit inputs a and b, a carry-in cin, and produces an 8-bit sum sum and a carry-out cout.
- It includes eight instances of the fulladd module (a1 to a8), which perform full addition for each bit position of the inputs.

- The carry-out (cout) generated by each full adder is connected to the carry-in (cin) input of the next full adder, resulting in a ripple effect that causes the carry to propagate from the least significant bit (LSB) to the most significant bit (MSB).

2. **Full adder module:**

- This module represents a full adder. It takes two input bits a and b, a carry-in cin, and produces a sum bit sum and a carry-out bit cout.
- The sum bit sum is computed using an XOR gate (a^b^cin), which performs bitwise addition and XORs the inputs with the carry-in.
- The carry-out bit cout is calculated using a combination of AND and OR gates. It considers all possible combinations of the input bits and the carry-in (cin) to determine if a carry is generated or not.
- The Adder Ripple module utilizes the fulladd module to perform bit-wise addition of the input bits (a and b) and propagate the carry through the cascade of full adders. The output sum represents the 8-bit sum of the inputs, while the cout indicates the carry-out from the most significant bit.

3.3 Arithmetic Unit

The Arithmetic unit, a component of the ALU (Arithmetic Logic Unit), is tasked with executing arithmetic operations, such as addition and subtraction, on binary numbers. It receives two input operands, performs the designated arithmetic operation based on control signals, and generates the resulting output.

The underlying concept behind the Arithmetic unit revolves around comprehending the fundamental principles of binary arithmetic. In digital systems, binary arithmetic adheres to the same principles as decimal arithmetic, albeit with a base of 2. The Arithmetic unit employs logic gates and combinational circuits to manipulate binary numbers, enabling the execution of desired arithmetic operations.

The Arithmetic unit typically consists of several components:

- Adder/Subtractor: The core component of the Arithmetic unit is the adder/subtractor circuit. It performs both addition and subtraction operations. The adder circuit adds two binary numbers and produces the sum, while the subtractor circuit subtracts one binary number from another.

- Carry/Borrow Logic: The carry/borrow logic is responsible for propagating carry or borrow signals generated during addition or subtraction operations. It ensures correct alignment and calculation of carry/borrow bits in multi-bit operations.

- Control Signals: The control signals determine the specific arithmetic operation to be performed. In the given module, the ALU_control signal of width 5 selects the desired operation, such as addition or subtraction.

- Input and Output Registers: The Arithmetic unit typically has input and output registers to latch and store the operands and results. In the given module, the x and y inputs represent the binary operands, while the out output represents the result of the arithmetic operation.

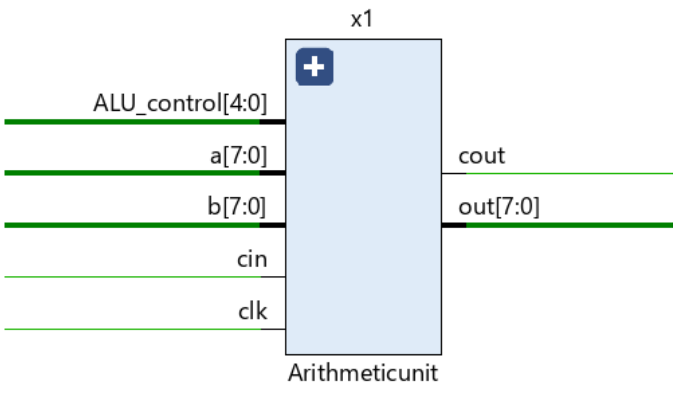


Fig 3.6 Arithmetic Unit

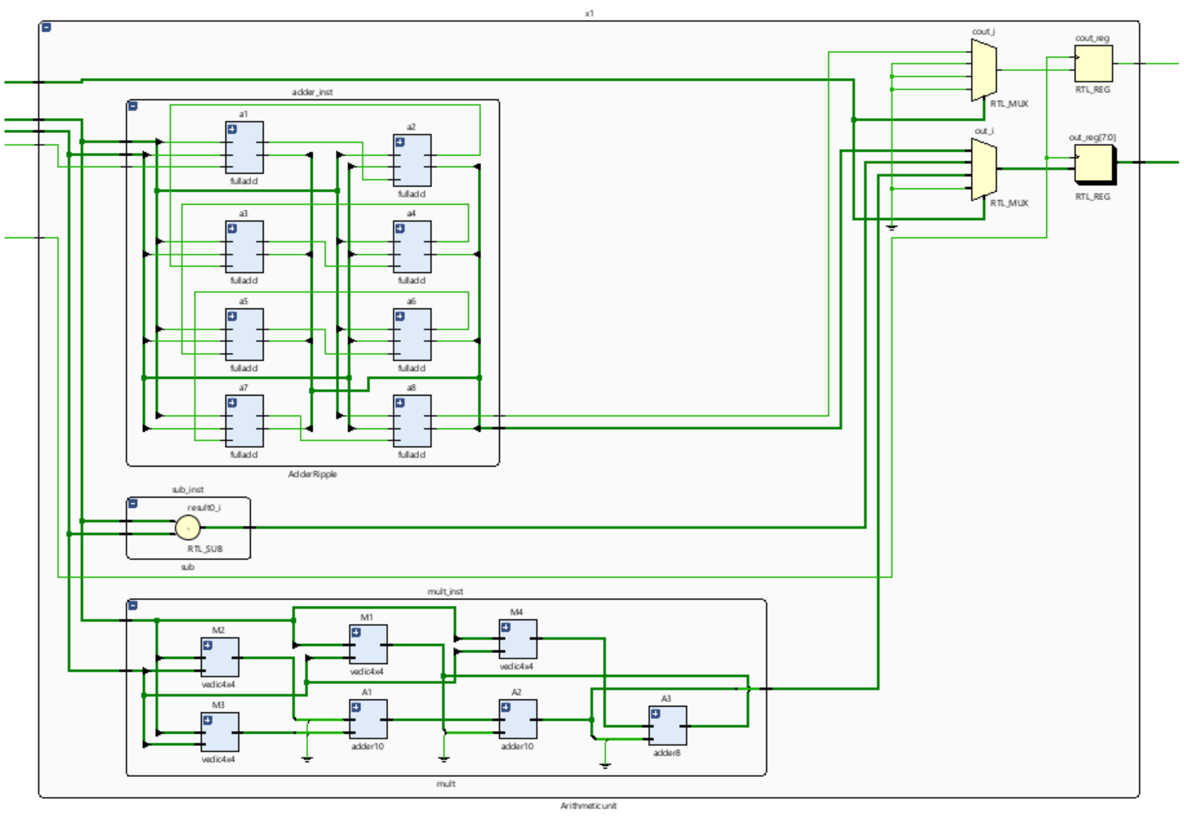


Fig 3.7 RTL Schematic of Arithmetic Unit

The functioning of the Arithmetic unit involves the following steps:

- The input operands x and y are provided to the Arithmetic unit.
- The control signals (ALU_control) specify the desired operation, such as addition or subtraction.
- The adder/subtractor circuit performs the arithmetic operation on the input operands based on the control signals.
- The carry/borrow logic handles any carry or borrow signals generated during the operation and produces the carry flag (carryflag) as the output.
- The result of the operation is stored in the output register (out), which represents the final output of the Arithmetic unit.
- Additional flags, such as the zero flag (zero_flag) and sign flag (sign_flag), may also be generated to indicate special conditions, such as if the result is zero or negative.

By combining these components and following the principles of binary arithmetic, the Arithmetic unit within the ALU enables the execution of arithmetic operations in digital systems. It forms a crucial part of processors and digital circuits, providing the capability to perform mathematical calculations and manipulations on binary data.

3.4 Logical Unit

The logical module, represented by the logical unit module, is responsible for performing logical operations on binary numbers based on the specified control signals. It takes two input operands (a and b), performs the specified logical operation, and produces the logical output.

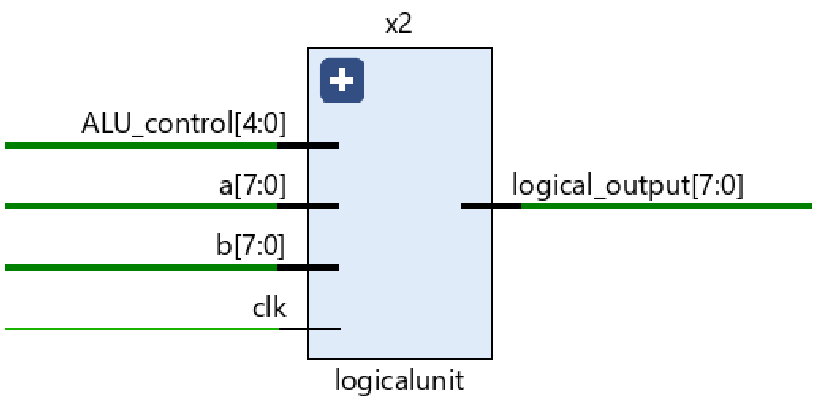


Fig 3.8 Logical Unit

The working of the logical module can be understood as follows:

- The input operands a and b represent binary numbers on which logical operations are performed.
- The control signals (ALU_control) determine the specific logical operation to be executed. The ALU_control signal is of width 5 in this module.
- The logical operations are implemented using a case statement within an always block that triggers on the positive edge of the clk signal.
- Inside the case statement, different cases are defined based on the value of ALU_control. Each case represents a specific logical operation to be performed.

- For each case, the logical output (logical_output) is assigned based on the operation. The logical operations include bitwise operations such as OR (|), AND (&), XOR (^), as well as other operations like complement (~), addition/subtraction with constant values, and comparison (==).
- If none of the defined cases match the ALU_control value, the default case assigns a default value of 8'b0 to the logical_output.
- The logical_output represents the result of the performed logical operation and is stored in the output register.

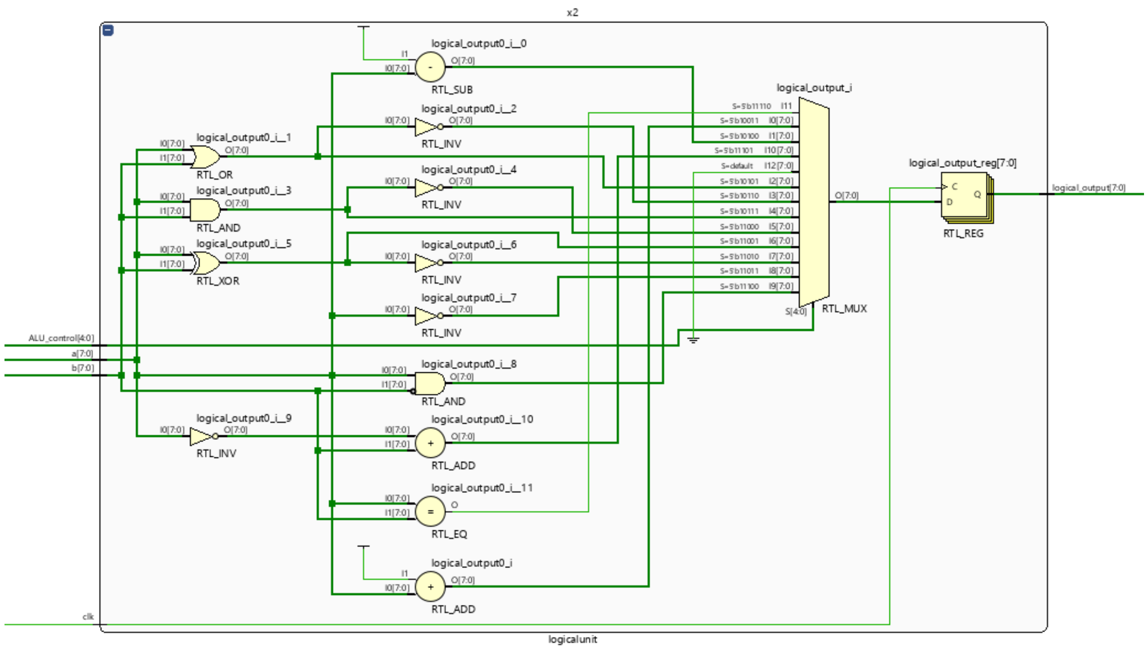


Fig 3.9 RTL Schematic of Logical Unit

By utilizing the case statement and the control signals, the logical module enables the execution of various logical operations on binary data. It provides the capability to perform bitwise manipulations, complement operations, comparisons, and other logical computations in digital systems.

3.5 ALU Design

The ALU described here incorporates clock gating and Vedic Mathematics principles, which offer certain advantages.

Clock gating is a technique used to selectively enable or disable the clock signal to specific parts of a circuit based on certain conditions. In the ALU, clock gating is employed to control the flow of clock signals to different submodules based on the ALU control signals. This technique helps in reducing power consumption by conserving clock energy in the submodules that are not actively involved in the current operation. By gating the clock, unnecessary switching activity and power dissipation can be minimized, resulting in improved energy efficiency.

The ALU incorporates Vedic Mathematics principles to optimize arithmetic operations. The use of a ripple carry adder submodule in the ALU allows for efficient addition of two 8-bit binary numbers.

By combining clock gating and Vedic Mathematics principles, the ALU in the provided code aims to achieve improved power efficiency and computational speed. The selective clock gating helps in reducing power consumption, while the Vedic Mathematics techniques optimize arithmetic operations, leading to faster computations. These design considerations contribute to making the ALU more efficient and advantageous compared to traditional approaches.

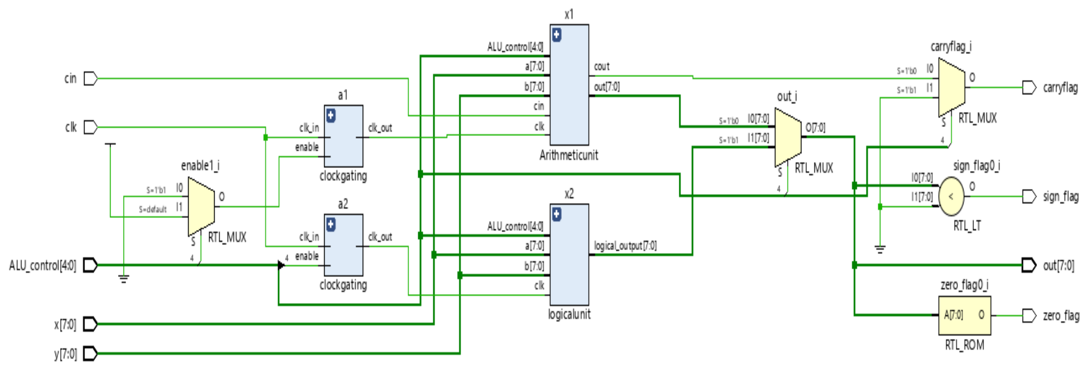


Fig 3.10 RTL Schematic of ALU

The ALU (Arithmetic Logic Unit) described in the given figure performs arithmetic and logical operations on two 8-bit inputs (x and y). Let's break down its working:
The ALU module has the following inputs:

- clk: Clock signal
- x: 8-bit input
- y: 8-bit input
- cin: Carry-in input
- ALU_control: 5-bit control signal
    - ALU_control[4]: Determines the operation to be performed. If ALU_control[4] is 0, the ALU performs Arithmetic operation; if 1, it performs Logical operations.
    - ALU_control[3:0]: Determines the specific arithmetic or logical operation to be performed.

The ALU module has the following outputs:

- zero_flag: Indicates whether the result is zero (1 if zero, 0 otherwise)
- sign_flag: Indicates the sign of the result (1 if negative, 0 otherwise)
- out: 8-bit output that holds the computed result
- carryflag: Indicates the carry-out from the ALU operation (valid only for addition)
-

Here's how the ALU works:

- Clock Gating:
    - Two clock gating modules (clockgating a1 and clockgating a2) are used to control the clock signal based on the ALU_control input.
    - clk_out1 is the clock output for the arithmetic operations, and clk_out2 is the clock output for the logical operations.
    - When ALU_control[4] is 0, enable1 is set to 1, and enable2 is set to 0. Vice versa when ALU_control[4] is 1.
- Arithmetic Unit:
    - The Arithmeticunit module performs arithmetic operations (addition and subtraction) based on the control signal and clock output.

    - It takes the clock output (clk_out1), input values (x and y), carry-in (cin), and ALU_control.
    - The Arithmeticunit module has an output Arithres, which holds the result of the arithmetic operation, and cout2 for the carry-out.
- Logical Unit:
    - The logicalunit module performs logical operations based on the control signal and clock output.
    - It takes the clock output (clk_out2), input values (x and y), and ALU_control.
    - The logicalunit module has an output logical_output, which holds the result of the logical operation.
- Output Assignment:
    - The output out is assigned based on the ALU_control signal.
    - When ALU_control[4] is 0, the output out is assigned with Arithres (result of the arithmetic operation) and carryflag is assigned with cout2 (carry-out from addition).
    - When ALU_control[4] is 1, the output out is assigned with logicalres (result of the logical operation), and carryflag is set to 0 since carry is not applicable for subtraction.
- Zero Flag:
    - The zero_flag is assigned based on the value of out.
    - If out is equal to 0, zero_flag is set to 1, indicating that the result is zero. Otherwise, it is set to 0.
- Sign Flag:
    - The sign_flag is assigned based on the signed value of out.
    - If out is less than 0 (interpreted as a signed value), sign_flag is set to 1, indicating a negative result. Otherwise, it is set to 0.

In summary, the ALU module performs arithmetic and logical operations based on the control signals. It uses clock gating to control the execution of arithmetic and logical operations. The output flags (zero_flag and sign_flag) provide additional information about the result, and the out signal holds the computed result.

CHAPTER 4: SIMULATION RESULT & DISCUSSION

4.1 Simulation Result of ALU

Based on the provided module and testbench code, the simulation results of the ALU module can be observed and analysed. The simulation captures the behaviour of the ALU for various input combinations and ALU control signals.

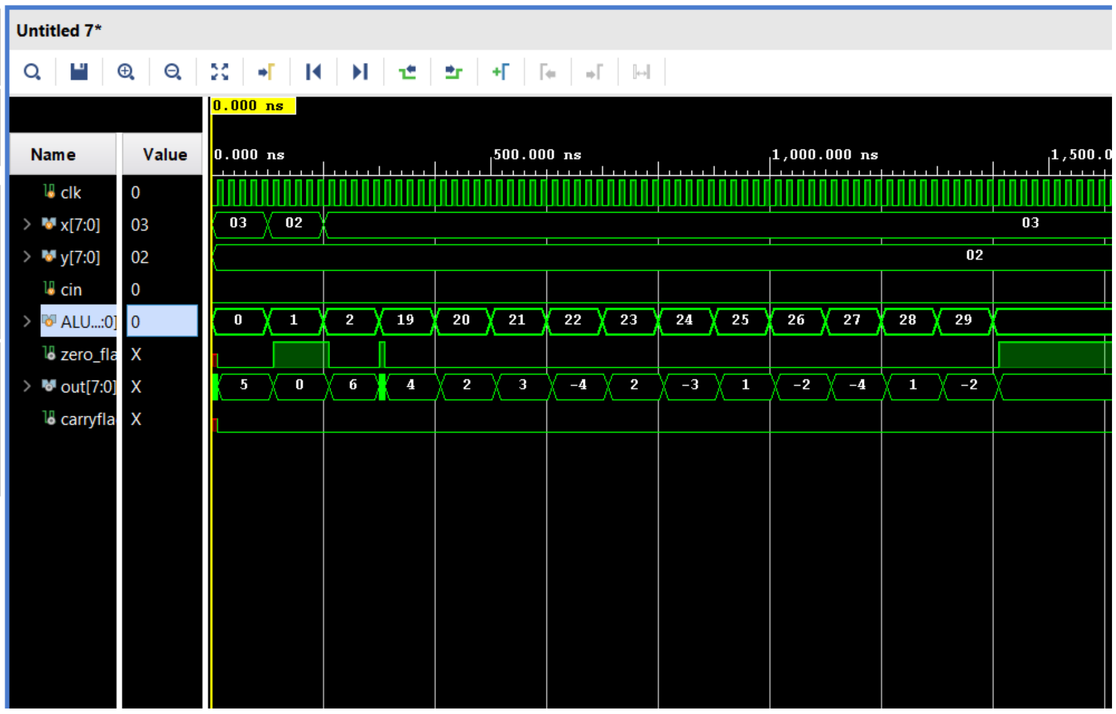


Fig 4.1 Simulation Result of ALU

We have given a clock signal of 20 ns in the testbench.

The zero flag (zero_flag) is correctly set to 1 when the output (out) of the ALU is zero, and 0 otherwise.

The simulation shows that the ALU module performs various arithmetic and logical operations based on the input signals x, y, cin, and ALU_control. The output signals out, zero_flag, and carryflag are updated accordingly.

During the simulation, we observe that the ALU correctly performs addition, subtraction, and multiplication operations based on the ALU_control values. The out signal reflects the expected arithmetic results, while the carry flag indicates the carry-out from addition or the lack there of for subtraction.

Overall, the simulation results indicate that the ALU module operates as expected, performing arithmetic and logical operations accurately and producing the correct output values and flag signals based on the given inputs and control signals.
From the simulation we can see that we are getting results as expected following output as shown in below table:

Input x	Input y	ALU function	ALU control signal	Output
3	2	x+y	00000	5
2	2	x-y	00001	0
3	2	x*y	00010	6
3	2	x+1	10011	4
3	2	x-1	10100	2
3	2	x|y	10101	3
3	2	~(x|y)	10110	-4
3	2	x&y	10111	2
3	2	~(x&y)	11000	-3
3	2	(x^y)	11001	1
3	2	~(x^y)	11010	-2
3	2	~x	11011	-4
3	2	x&~y	11100	1
3	2	~x+y	11101	-2
3	2	x==y	11110	0

Table 4.1 Output of Simulation

4.2 Power Improvement Analysis

The power consumption of various Arithmetic Logic Unit (ALU) designs was examined in this study. The initial three designs, including the Conventional ALU [2], ALU with Clock Gating [6], and ALU with Vedic Multiplier [2], were implemented based on the design proposed by the authors in their research paper. Conversely, the fourth design, ALU with Clock Gating and Vedic Multiplier, introduced in this study, represents a novel research contribution.

Design	**Power(W)**
Conventional ALU	5.540
ALU with Clock Gating	5.334
ALU with Vedic Multiplier	5.548
ALU with Clock gating and Vedic Multiplier	5.445

Table 4.2 Power consumption of different ALUs

Based on the power consumption data provided in the table, we can discuss following points:

- Conventional ALU: This design consumes the highest power, with a power consumption of 5.540W. It represents the baseline power consumption of the ALU without any optimization techniques.
- ALU with Clock Gating: The ALU design with clock gating shows a reduced power consumption of 5.334W compared to the conventional ALU. Clock gating is an effective power-saving technique that selectively enables or disables clock signals to unused or idle components, reducing unnecessary power consumption.
- ALU with Vedic Multiplier: The ALU design incorporating a Vedic multiplier exhibits a slightly higher power consumption of 5.548W compared to the conventional ALU. The Vedic multiplier is a multiplication algorithm based on ancient Indian mathematics, known for its speed and efficiency. While it improves performance, it may result in slightly higher power consumption due to the complexity of the multiplier circuit.
- ALU with Clock Gating and Vedic Multiplier: The ALU design combining clock gating and the Vedic multiplier demonstrates a reduced power consumption of

5.445W compared to both the conventional ALU and the ALU with only the Vedic multiplier. This indicates that the benefits of clock gating in reducing unnecessary power outweigh the slight increase in power due to the Vedic multiplier.

4.3 Speed Improvement Analysis

The Delay Analysis of various Arithmetic Logic Unit (ALU) designs was examined in this study. The initial three designs, including the Conventional ALU [2], ALU with Clock Gating [6], and ALU with Vedic Multiplier [2], were implemented based on the design proposed by the authors in their research paper. Conversely, the fourth design, ALU with Clock Gating and Vedic Multiplier, introduced in this study, represents a novel research contribution.

Design	**Delay(ns)**
Conventional ALU	7.226
ALU with Clock Gating	7.353
ALU with Vedic Multiplier	7.007
ALU with Clock gating and Vedic Multiplier	7.221

Table 4.3 Delay comparison of different ALUs

Based on the delay data provided in the table, we can discuss the following points:

- Conventional ALU: The conventional ALU has the highest delay of 7.226 ns. This delay represents the baseline performance of the ALU without any optimization techniques. It indicates the time required for the ALU to complete its operations.
- ALU with Clock Gating: The ALU design with clock gating shows a slightly increased delay of 7.353 ns compared to the conventional ALU. Clock gating introduces additional circuitry and logic to control the clock signals, which can introduce some overhead and result in a slightly longer delay.

- ALU with Vedic Multiplier: The ALU design incorporating a Vedic multiplier demonstrates a reduced delay of 7.007 ns compared to the conventional ALU. The Vedic multiplier algorithm is known for its speed and efficiency, which contributes to the improved performance and reduced delay in the ALU.
- ALU with Clock Gating and Vedic Multiplier: The ALU design combining clock gating and the Vedic multiplier exhibits a delay of 7.221 ns. It shows a slightly increased delay compared to the ALU with only the Vedic multiplier but still performs better than the conventional ALU.

We can observe that the inclusion of the Vedic multiplier tends to improve the performance by reducing the delay, while the introduction of clock gating may introduce some additional delay due to the added circuitry.

4.4 Synthesis Results

The synthesis schematic of the ALU with Clock gating and Vedic Multiplier combines two key optimization techniques to improve power efficiency and performance: clock gating and the Vedic multiplier algorithm. Let's discuss each of these components and their impact on the synthesis schematic:

Clock Gating: Clock gating is a power optimization technique that aims to reduce power consumption by selectively disabling clock signals to specific circuit elements when they are not in use. In the ALU design, clock gating is applied to certain modules or sub-modules to minimize unnecessary switching activity and reduce power consumption. The synthesis schematic incorporates additional logic gates and control signals to enable or disable the clock signals based on specific conditions or input patterns.

Vedic Multiplier: The Vedic multiplier is a high-speed multiplication algorithm inspired by ancient Indian mathematics. It offers a more efficient and faster multiplication operation compared to conventional multiplication techniques such as the shift-and-add algorithm. The Vedic multiplier module in the synthesis schematic includes specialized circuits and computational stages that perform the multiplication using Vedic mathematics principles. These stages are optimized for high-speed and low-power operation, resulting in improved performance and reduced power consumption during multiplication operations.

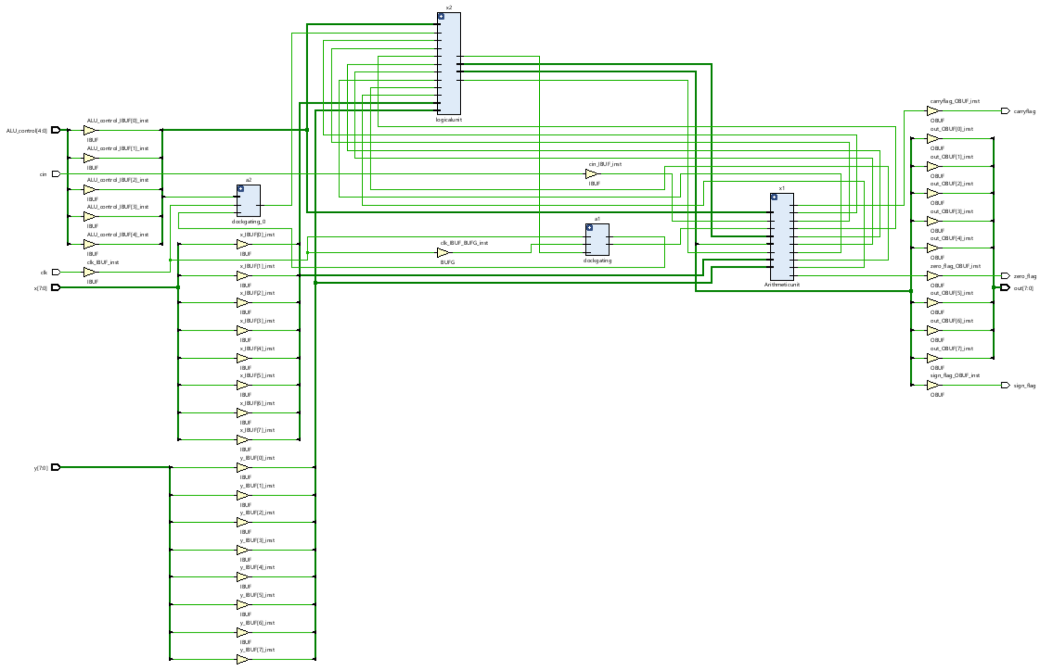


Fig 4.2 Synthesis Schematic of ALU

The synthesis process combines these two optimization techniques into a single ALU design. It involves converting the RTL (Register Transfer Level) description of the ALU into a gate-level netlist. During synthesis, the synthesis tool analyses the RTL description and applies various optimization algorithms to generate an optimized gate-level representation of the design.

The synthesis schematic of the ALU with Clock gating and Vedic Multiplier includes modules such as clock gating logic, Vedic multiplier units, arithmetic and logical computation units, control logic, and interconnects. These modules are interconnected based on the design specifications and the synthesis tool's optimizations. The clock gating logic would be strategically placed to selectively enable or disable clock signals to minimize power consumption. The Vedic multiplier modules would be designed to efficiently perform multiplication operations, leveraging the principles of Vedic mathematics.

Overall, the synthesis schematic of the ALU with Clock gating and Vedic Multiplier combines the benefits of clock gating and the Vedic multiplier algorithm to achieve a power-efficient and high-performance ALU design. Here we have used Artrix 7 family of FPGA for implementation. The specific details and intricacies of the schematic would depend on the design specifications, synthesis tool used, and the targeted technology or hardware platform.

CHAPTER 5: CONCLUSION AND FUTURE WORK

5.1 Summary of Findings

The findings from the provided data reveal interesting insights about the different ALU designs in terms of delay and power consumption.

When comparing the delay of the designs, it is observed that the Conventional ALU has a delay of 7.226 ns. The introduction of clock gating in the ALU design slightly increases the delay, as the ALU with Clock Gating exhibits a delay of 7.353 ns. However, the implementation of the Vedic Multiplier significantly improves the delay, as the ALU with Vedic Multiplier demonstrates a lower delay of 7.007 ns. Combining clock gating and the Vedic Multiplier in the ALU design results in a delay of 7.221 ns for the ALU with Clock gating and Vedic Multiplier.

In terms of power consumption, ALU with vedic multiplier consumes the highest power at 5.548 W. On the other hand, the ALU with Clock Gating offers more efficient power utilization, as it consumes the lowest power at 5.334 W. The inclusion of the Vedic Multiplier slightly increases the power consumption, with the ALU with Vedic Multiplier consuming 5.548 W. By combining clock gating and the Vedic Multiplier, the ALU with Clock gating and Vedic Multiplier strikes a balance between power consumption and performance, consuming 5.445 W.

Considering both the delay and power consumption, it can be concluded that ALU with Clock gating and Vedic Multiplier is a favourable choice. It exhibits a relatively lower delay compared to the Conventional ALU while consuming less power. However, if faster operation is a priority, the ALU with Vedic Multiplier proves to be advantageous due to its optimized multiplication algorithm, resulting in lower delay.

Additionally, the findings highlight the impact of different design techniques on the performance and efficiency of the ALU. The inclusion of clock gating in the ALU design aims to reduce power consumption by selectively enabling clock signals to specific modules or blocks. While it does result in a slight increase in delay, the overall power reduction is evident, as seen in the lower power consumption of the ALU with Clock Gating compared to the Conventional ALU.

On the other hand, the integration of the Vedic Multiplier introduces a specialized multiplication algorithm inspired by ancient Vedic mathematics. This algorithm optimizes the multiplication operation, leading to improved performance and reduced delay in the ALU design. Although the Vedic Multiplier adds a slight increase in power consumption compared to the Conventional ALU, the trade-off between performance and power efficiency is generally favourable.

The combination of clock gating and the Vedic Multiplier in the ALU design aims to leverage the benefits of both techniques. It seeks to achieve a balance between reduced power consumption through clock gating and enhanced performance through the optimized multiplication algorithm. As observed, the ALU with Clock gating and Vedic Multiplier achieves a lower delay compared to the Conventional ALU, showcasing the effectiveness of the integrated design approach.

Overall, the findings emphasize the importance of considering different design techniques and their implications when developing ALU architectures. By incorporating techniques such as clock gating and specialized algorithms like the Vedic Multiplier, designers can enhance the performance, power efficiency, and overall functionality of the ALU. The choice of specific design techniques should be based on the priorities and requirements of the target application, ensuring a well-optimized and balanced design solution.

5.2 Future Work

Based on the findings from the current study, there are several potential avenues for future work and further improvements in the design of the ALU:

- **Optimization of power consumption:** While clock gating and the Vedic Multiplier have shown positive effects on power reduction individually, further exploration can be done to optimize power consumption even more. This can involve investigating advanced clock gating techniques, such as fine-grained gating or dynamic gating, to achieve more efficient power utilization.
- **Exploration of alternative multiplication algorithms:** The Vedic Multiplier has demonstrated advantages in terms of performance and delay reduction. However, other multiplication algorithms, such as Booth's algorithm or Wallace

Tree Multiplier, could be explored and compared to determine their suitability for the ALU design. This exploration may uncover new approaches that offer a better balance between performance, power, and area efficiency.

- **Integration of advanced optimization techniques:** Additional optimization techniques can be investigated to further improve the overall performance of the ALU. These may include architectural modifications, algorithmic enhancements, or the utilization of advanced synthesis and optimization tools. By employing these techniques, it may be possible to achieve even lower delays and power consumption without sacrificing functionality.
- **Evaluation of different design trade-offs:** Future work can involve conducting a comprehensive analysis of various trade-offs in the ALU design, such as area versus power, performance versus power, or area versus performance. By exploring different design points within these trade-offs, designers can gain a better understanding of the design space and make informed decisions based on the specific requirements of their target applications.
- **Implementation on different FPGA or ASIC platforms:** The current study focused on a specific platform, but future work could involve implementing and evaluating the ALU design on different FPGA or ASIC architectures. This would help assess the design's scalability, portability, and performance across a broader range of hardware platforms.
- **Exploration of other ALU functionalities:** While the current study focused on arithmetic and logical operations, future work can expand the scope to include additional ALU functionalities, such as bitwise operations, shift operations, or more advanced mathematical functions. This would contribute to a more comprehensive and versatile ALU design capable of supporting a wider range of applications.

By pursuing these avenues for future work, researchers and designers can continue to advance the field of ALU design, aiming for improved performance, power efficiency, and versatility in various computing systems and applications.

REFERENCES

1. A. Alrashdi and M. I. Khan, “Design and Comparative Analysis of High Speed and Low Power ALU Using RCA and Sklansky Adders for High-Performance Systems”, Eng. Technol. Appl. Sci. Res., vol. 12, no. 2, pp. 8426–8430, Apr. 2022.
2. N. Gadda and U. Eranna, "64-bit ALU Design using Vedic Mathematics," 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), Vellore, India, 2020, pp. 1-4, doi: 10.1109/ic-ETITE47903.2020.122.
3. N. Khanna and D. K. Mishra, "Clock Gated 16-Bits ALU Design & Implementation on FPGA," 2018 4th International Conference for Convergence in Technology (I2CT), Mangalore, India, 2018, pp. 1-5, doi: 10.1109/I2CT42659.2018.9057910.
4. M. Vatte, "Power optimization in configurable ALU using blend of techniques," 2021 2nd International Conference for Emerging Technology (INCET), Belagavi, India, 2021, pp. 1-5, doi: 10.1109/INCET51464.2021.9456346.
5. G. Verma, "Design and Analysis of ALU for Low Power IOT Centric Processor Architectures," 2020 Global Conference on Wireless and Optical Technologies (GCWOT), Malaga, Spain, 2020, pp. 1-5, doi: 10.1109/GCWOT49901.2020.9391609.
6. R. Kulkarni and S. Y. Kulkarni, "Power analysis and comparison of clock gated techniques implemented on a 16-bit ALU," International Conference on Circuits, Communication, Control and Computing, Bangalore, India, 2014, pp. 416-420, doi: 10.1109/CIMCA.2014.7057835.
7. G. Rawat, K. Rathore, S. Goyal, S. Kala and P. Mittal, "Design and analysis of ALU: Vedic mathematics approach," International Conference on Computing, Communication & Automation, Greater Noida, India, 2015, pp. 1372-1376, doi: 10.1109/CCAA.2015.7148593.
8. S. Lad and V. S. Bendre, "Design and Comparison of Multiplier using Vedic Sutras," 2019 5th International Conference On Computing, Communication, Control And Automation (ICCUBEA), Pune, India, 2019, pp. 1-5, doi: 10.1109/ICCUBEA47591.2019.9128517.

9. D. Lachireddy and S.R. Ramesh, "Power and Delay Efficient ALU Using Vedic Multiplier," in T. Sengodan, M. Murugappan, and S. Misra (Eds.), Advances in Electrical and Computer Technologies, Lecture Notes in Electrical Engineering, vol. 672, Singapore: Springer, 2020, pp. 761-769, doi: 10.1007/978-981-15-5558-9_61
10. A. Aadarsh, A. Kumar, A. Yadav and P. C. Joshi, "Design and Power Analysis of 32-Bit Pipelined Processor," 2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), Greater Noida, India, 2021, pp. 604-608, doi: 10.1109/ICACITE51222.2021.9404622.