DESIGN AND SIMULATION OF 32-BIT ARITHMETIC LOGIC UNIT DEVELOPMENT USING VERILOG

[1]M.VINEETHA, [2]DR.CH.VENUGOPALREDDY, [3]T. KALA VYSHNAVI, [4]P.V. ANEESHA, [5]J. HARI PRIYA

[2]Professor & HOD, ECE Dept, RISE Krishna Sai Prakasam Group of Institutions, Valluru, AP

[1,3,4,5,]Students, ECE Dept, RISE Krishna Sai Prakasam Group of Institutions, Valluru, AP

[1]mattavineetha3107@gmail.com,[2]phdvenu@gmail.com,[3]kalavyshnavi@gmail.com,[4]honeyaneesha@gmail.com, [5]jadahariipriya8191@mail.com

ABSTRACT

The goal of this project is to design and develop a 32-bit Arithmetic Logic Unit (ALU) using Verilog HDL. Vivado Design Suite is used for simulation and synthesis. The ALU is architected using a modular method and comprises of three major functional units: an Arithmetic Unit, a Logic Unit, and a Shift Unit. The arithmetic unit facilitates effective 32-bit numerical computing by supporting fundamental operations including addition, subtraction, increment, and decrement. The logic unit conducts basic bitwise operations including AND, OR, XOR, and NOT, which are crucial to digital processing and control logic. Bit manipulation and arithmetic scaling are made easier by the shift unit's left and right shift operations.

All functional units are integrated using multiplexers and governed by a common set of control signals, allowing seamless selection and execution of required operations within a single ALU framework. The design is scalable, fully synthesizable, and ready for hardware implementation. Functional verification and synthesis results obtained using Vivado validate the correctness and efficiency of the proposed ALU architecture. A dependable and adaptable 32-bit ALU design appropriate for FPGA-based digital systems and processor applications is demonstrated in this project.

Keywords: 32-bit ALU, Verilog HDL, Arithmetic Unit, Logic Unit, Shift Unit, Digital System Design, FPGA, ASIC

I INTRODUCTION

The Arithmetic Logic Unit (ALU) is the fundamental computational component of processors, microcontrollers, and application-specific integrated circuits in contemporary digital systems. The arithmetic, logical, and shift operations that underpin all data processing activities are carried out by the ALU. With the increasing demand for high-performance, low-power, and scalable digital systems, efficient ALU design has become a significant area of research and development in the field of VLSI and digital system design.

The rapid growth of embedded systems, FPGA-based applications, and custom processor architectures has created a need for flexible and modular ALU designs that can be easily customized and optimized for different applications. A 32-bit ALU is particularly important because it aligns with the word length used in many modern processors and digital platforms, enabling efficient handling of large data sets and complex computations.

Verilog Hardware Description Language (HDL) has emerged as one of the most widely used languages for modelling, designing, and simulating digital systems at various abstraction levels. It allows designers to describe hardware behaviour and structure in a precise and synthesizable manner. Verilog HDL allows designers to effectively simulate, synthesis, and execute complicated digital circuits on FPGA platforms when paired with sophisticated design tools like the Vivado Design Suite. The design and implementation of a 32-bit Arithmetic Logic Unit using Verilog HDL is the main goal of this project. The suggested ALU architecture adheres to a modular design methodology, which divides the entire system into smaller, clearly defined functional units. Specifically, the ALU consists of three primary components: an Arithmetic Unit, a Logic Unit, and a Shift Unit. Each unit is designed independently and later integrated using multiplexers controlled by select signals.

The Arithmetic Unit supports essential operations such as addition, subtraction, increment, and decrement, which are fundamental for numerical computation in digital systems. Bitwise logical operations like AND, OR, XOR, and NOT, which are frequently utilized in control logic, masking operations, and decision-making processes, are carried out by the Logic Unit. The Shift Unit enables left and right shift operations, which are critical for arithmetic scaling, bit manipulation, and efficient multiplication or division by powers of

two.

The design attains flexibility, reusability, and scalability by combining these functional units into a single ALU framework. The entire ALU is fully synthesizable and suitable for FPGA and ASIC implementation. The suggested design's accuracy, performance, and viability are verified through functional simulation and synthesis using the Vivado Design Suite.

The design attains flexibility, reusability, and scalability by combining various functional units into a single ALU framework. The entire ALU is fully synthesizable and suitable for FPGA and ASIC implementation. The suggested design's accuracy, performance, and viability are verified by functional simulation and synthesis using the Vivado Design Suite. This project serves as a strong foundation for understanding processor data path components and provides practical exposure to Verilog-based digital design.

II LITERATURE SURVEY

[1] Mitchell (1962) proposed the first logarithmic multiplication algorithm, replacing conventional multiplication with addition using logarithmic approximation. This approach significantly reduced hardware complexity and computation time. However, approximation errors were introduced, especially for certain operand combinations. Despite its limitations, this work laid the foundation for approximate arithmetic designs.

[2] Hennessy and Patterson (1990) discussed the fundamental role of the Arithmetic Logic Unit in processor data path design. Their work emphasized how efficient ALU architecture directly impacts overall processor performance, power consumption, and instruction throughput.

[3] Parhami (2000) presented comprehensive techniques for computer arithmetic, including ALU design strategies for addition, subtraction, and logical operations. The study highlighted trade-offs between speed, area, and power in digital arithmetic circuits.

[4] Weste and Harris (2005) analysed CMOS VLSI design methodologies and demonstrated optimized ALU implementations using modular and hierarchical design approaches. Their work emphasized synthesizable HDL coding practices for ASIC and FPGA platforms.

[5] Koren (2002) explored computer arithmetic algorithms with a focus on error detection and correction in ALU operations. The work provided insights into reliable arithmetic unit design for safety-critical applications.

[6] Brown and Rose (2007) investigated FPGA-based ALU implementations and showed that modular ALU architectures improve scalability and ease of verification. Their work demonstrated efficient synthesis using modern FPGA tools.

[7] Sklansky (1960) introduced parallel prefix addition techniques that influenced the design of fast adders within ALUs. These techniques significantly reduced carry propagation delay compared to ripple carry adders.

[8] Brent and Kung (1982) proposed tree-based adder architectures that balance speed and hardware complexity. Their work remains influential in high-performance ALU arithmetic unit design.

[9] Rabaey et al. (2003) discussed low-power digital integrated circuit design and showed how optimized ALU architectures contribute to reduce dynamic and static power consumption in processors.

[10] Sentovich et al. (1992) introduced logic synthesis techniques that enabled efficient mapping of ALU designs onto FPGA fabrics. Their work emphasized the importance of synthesizable HDL descriptions.

III EXISTING SYSTEM

The current system uses Ripple Carry-based arithmetic circuits to implement a 32-bit Arithmetic Logic Unit (ALU). The ALU is designed using three major functional blocks: the Arithmetic Unit, Logical Unit, and Shift Unit, whose outputs are selected using a 16:1 multiplexer controlled by select signals.

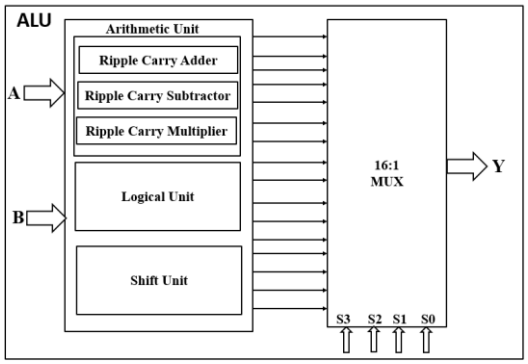


FIG 1: BLOCK DAIGRAM OF ALU USING RIPPLE CARRY ADDER

Ripple Carry-Based Arithmetic Unit

Ripple Carry logic, the most straightforward and widely used arithmetic architecture, is used to build the Arithmetic Unit in the current system.

a) Ripple Carry Adder

• Adds inputs A and B by 32 bits.
• Every full adder produces a carry that needs to move up to the following bit.
• From LSB to MSB, carry propagation happens bit by bit.
b) Subtractor for Ripple Carry
• The two's complement method is used to perform subtraction.
• The same ripple carry adder construction is used to complement and add input B with input A.
• The ripple mechanism is still used in borrow/carry propagation.
c) Multiplier for Ripple Carry
• Ripple carry adders and partial product generation are used to perform multiplication.
• Ripple carry logic is used in every partial product addition.

The Logical Unit

32-bit inputs A and B are subjected to bitwise operations by the Logical Unit, including:
• AND
• OR
• XOR
• NOT
Characteristics:
• These operations do not require carry propagation.
• Compared to arithmetic operations, execution is quicker.
• There isn't much hardware complexity.

Unit of Shift

The Shift Unit supports:
• The logical left shift
• Logical Right Shift Features:
• Used for bit manipulation, scaling, and alignment.
• Arithmetic carry logic is not used to accomplish shifting; instead, wiring is used.
• There is very little delay.

16:1 MUX multiplexer

• One operation result is chosen by a 16:1 multiplexer from: o Arithmetic unit outputs
Logical unit outputs
o Shift unit outputs
• Select lines S0, S1, S2, and S3 control it.
Function:
• Guarantees that the ALU output receives just one operation output.
• Increases selection latency, particularly when slow arithmetic blocks are used.

Output Block

• The final 32-bit ALU output is the multiplexer's chosen output.

· The following factors affect output delay: o Arithmetic carry propagation delay
o Delay in multiplexer selection

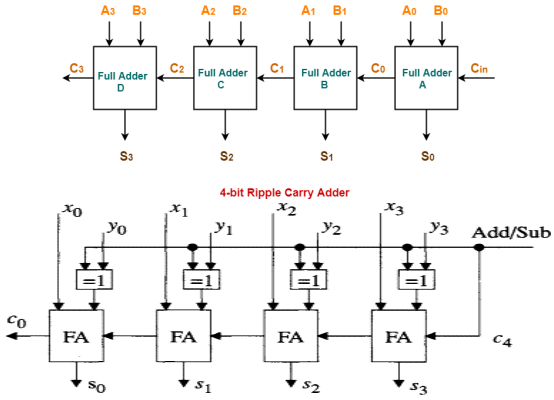


FIG 2: RIPPLE CARRY ADDER/SUBTRACTOR

IV PROPOSED SYSTEM

By substituting a Carry Save Adder (CSA) for the Ripple Carry Adder (RCA) in the arithmetic unit, including the multiplier block, the suggested ALU architecture addresses the performance constraints of the current ripple carry-based design. The Arithmetic Unit, Logical Unit, and Shift Unit make up the overall modular ALU structure, and a 16:1 multiplexer with four selection lines (S3–S0) controls operation selection.

Adders are crucial components of computer and arithmetic circuits that can be used in signaling techniques. Adders are typically found in a variety of microprocessor processors and digital signal processing building components. Although there are many various kinds of adders, such as the carry skip adder, CLA adder, RCA or ripple carry adder, and many more, the carry save adder is a low-cost, high-speed adder with little latency. Multi-operand addition is commonly used in array processing, division, and multiplication. Several strong adders are required to add multiple numbers instead of just two. A high-speed multi-operand type adder that is ideal for adding multiple operands together is called a carry-save adder, or CSA. As a result, it avoids speed-up computation & time-consuming carry propagation. This article discusses an overview of a carry save adder or CSA.

Carry save Adder: What is it?

A carry-save adder, often known as a CSA, is a kind of digital adder that is primarily used to effectively calculate the sum of three or more binary values. Because a binary multiplier adds the two binary integers above after multiplication, a CSA is typically utilized within a binary multiplier. This technique allows for the implementation of a

large adder, which is significantly faster than the typical addition of numbers.

Block Diagram of Carry save Adder

The Carry Save adder circuit diagram is illustrated below. This sort of adder differs greatly from others in that it stores the carry and adds to the sum of the subsequent stage using another fuller adder (FA) rather than transmitting the middle carries toward the subsequent stages. The method of adding binary bits involves preserving the carriers and sum bits in the first stage before moving on to the second. The stored carry and sum bits are added at this stage, which functions similarly to a ripple carry adder, or RCA. The operands utilized in this adder are three like X, Y & Z where 'Z' is a four-bit input carry. In this case, for each bit of X, Y, and Z,

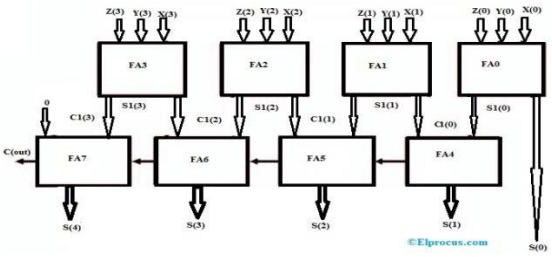


FIG 3: BLOCK DIAGRAM OF CARRY SAVE ADDER

The sum and carry bits are generated for each FA. Here, the carry bits are simply kept and added to the next sum term using a ripple carry adder rather than being sent to the subsequent FA.

Block Diagram of Alu Using Carry Save Adder

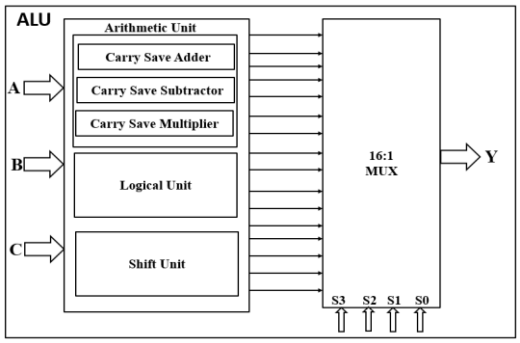


FIG 4: BLOCK DIAGRAM OF ALU USING CARRY SAVE ADDER

Carry save Adder for Arithmetic Units

Carry Save Adders are used by the Arithmetic Unit in the suggested scheme for:

• Extension

• Subtraction

Multiplication

a) Carry save Adder (CSA)

• CSA generates sum and carry outputs simultaneously to accomplish addition.

• The carry does not instantly spread to the next higher bit, in contrast to RCA.

• As a result, the critical path delay is greatly decreased.

b) Carry save Subtractor

• The two's complement representation is used to implement subtraction.

• Compared to ripple carry subtraction, CSA reduces latency by processing sum and carry bits individually.

The Logical Unit

• Carries out bitwise operations, including AND, OR, XOR, NAND, NOR, NOT, and XNOR.

• Carry propagation is not a part of logical processes.

• Functions autonomously and effectively in both the suggested and current systems.

Unit of Shift

Logical left and right shift operations are supported.

• Used for bit manipulation and arithmetic scaling.

• Because shifting is done via wiring, there is very little delay.

Multiplexer-Based Operation Selection

A 16:1 multiplexer is coupled to the outputs of the Arithmetic, Logical, and Shift Units. • The intended operation is controlled by selection lines S0, S1, S2, and S3. • The ALU output receives the chosen operation result from the multiplexer.

V RESULTS

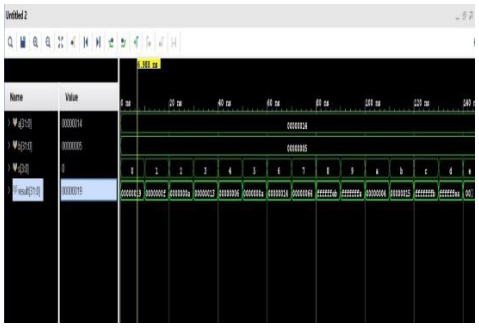


FIG 5: Simulation Output

```
# run 1000ns
-------------------------------------------------------------
 S     Operation          A     B     Result
-------------------------------------------------------------
0000  A + B              20          5          25
0001  A - B              20          5          15
0010  A >> 1             20          5          10
0011  A - 1              20          5          19
0100  B + 1              20          5           6
0101  B << 1             20          5          10
0110  Transfer A         20          5          20
0111  A * B              20          5         100
1000  NOT A              20          5   fffffffeb
1001  NOT B              20          5   fffffffa
1010  A AND B            20          5           4
1011  A OR  B            20          5          21
1100  NAND               20          5   fffffffb
1101  NOR                20          5   ffffffea
1110  XOR                20          5          17
1111  XNOR               20          5   ffffffee
-------------------------------------------------------------
```

FIG 6: Operation of ALU

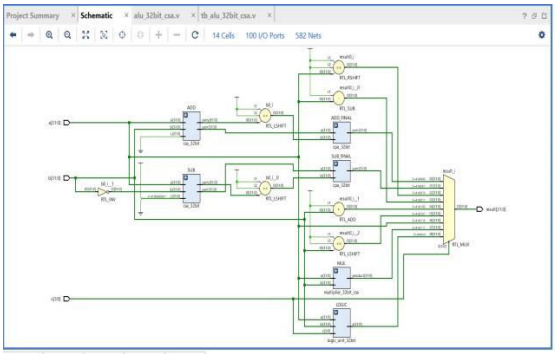


FIG 7: Schematic Diagram

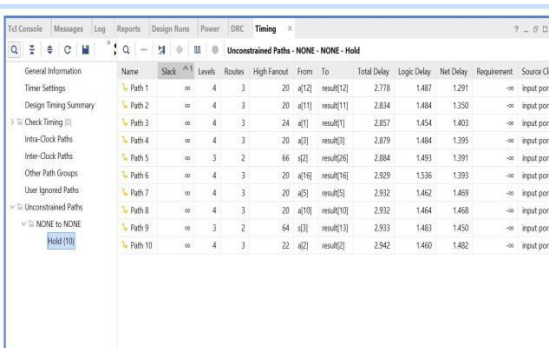
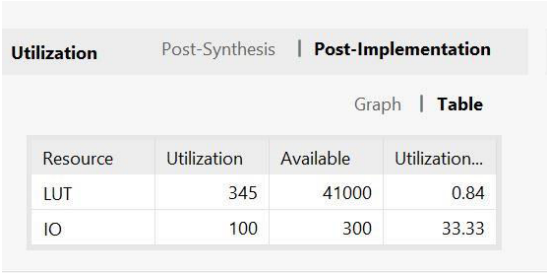


FIG 10: Estimation of Delay



Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization...
LUT	345	41000	0.84
IO	100	300	33.33

FIG 8: Area Utilization

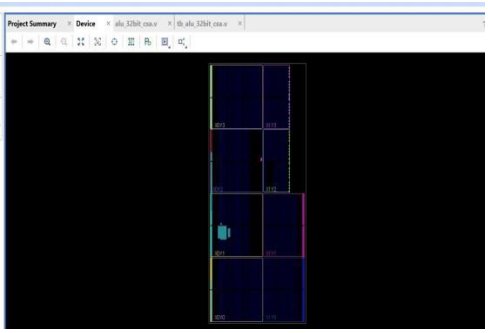


FIG 11: Diagram of Internal Schematic

FIG 9: Power Consumption

Table-1 Comparison table

Parameter	Existing Method (RCA)	Proposed Method (CSA)
Area(LUT)	1433	345
Power(Watt)	73.926	23.295
Dealy(ns)	3.8	2.5

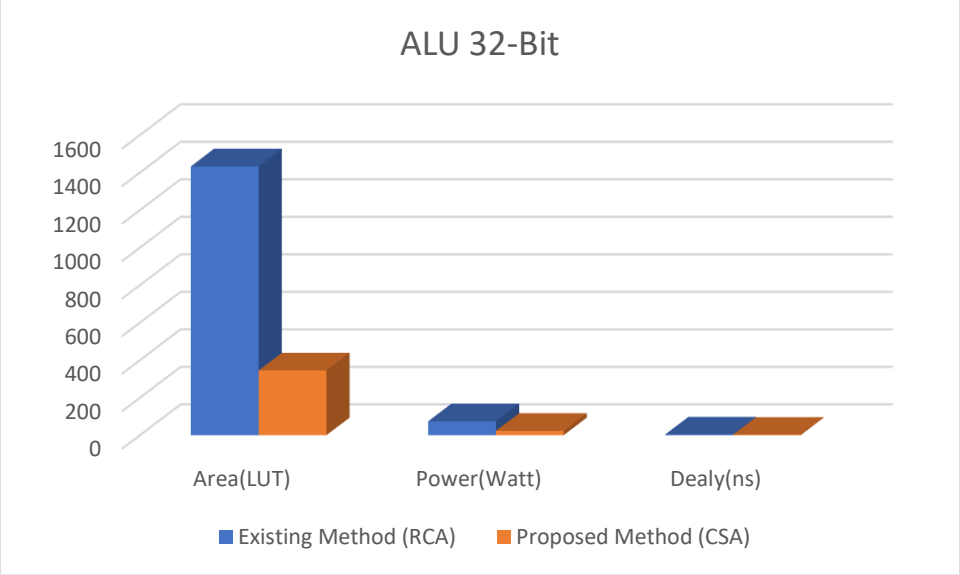


FIG 12: Comparison Table in 3D View

VI CONCLUSION

In this project, a 32-bit Arithmetic Logic Unit (ALU) is successfully designed and implemented using Verilog HDL, with simulation and synthesis carried out in the Vivado Design Suite. An Arithmetic Unit, Logical Unit, and Shift Unit make up the ALU architecture's modular design, which improves clarity, scalability, and verifiability.

Ripple Carry Adders (RCA) were used to implement the arithmetic and multiplier blocks in the current system, which increased propagation latency, increased power consumption, and decreased performance, particularly for multiplication operations. The suggested method substitutes a Carry Save Adder (CSA) in the arithmetic unit, including the multiplier block, for RCA in order to get around these restrictions.

REFERENCES

[1] J. L. Mitchell, "Computer multiplication and division using binary logarithms," IRE Transactions on Electronic Computers, 1962.

[2] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann, 1990.

[3] M. D. Ercegovac and T. Lang, Digital Arithmetic, Morgan Kaufmann, 2004.

[4] N. H. E. Weste and D. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, Pearson, 2005.

[5] K. Parhami, Computer Arithmetic: Algorithms and Hardware Designs, Oxford University Press, 2000.

[6] J. Rabaey, A. Chandrakasan, and B. Nikolic, Digital Integrated Circuits, Prentice Hall, 2003.

[7] S. Brown and Z. Vranesic, Fundamentals of Digital Logic with Verilog Design, McGraw-Hill, 2007.

[8] Xilinx Inc., Vivado Design Suite User Guide, 2018.

[9] S. Knowles, "A family of adders," Proceedings of the 15th IEEE Symposium on Computer Arithmetic, pp. 30–38, 2001.

[10] R. Zimmermann, "Efficient VLSI implementation of modulo ($2^n \pm 1$) addition and multiplication," Proceedings of the 14th IEEE Symposium on Computer Arithmetic, 1999.

[11] P. K. Meher, "Arithmetic circuit design using carry save techniques," IEEE Transactions on Circuits and Systems, vol. 58, no. 9, pp. 2152–2164, 2011.

[12] A. Dandapat, S. Ghosal, and P. Sarkar, "Low-power and high-speed arithmetic unit design using carry save adder," International Journal of Electronics, vol. 97, no. 4, pp. 429–441, 2010.

[13] M. Mano and M. D. Ciletti, Digital Design: With an Introduction to the Verilog HDL, Pearson Education, 2013.

[14] B. Parhami and C.-Y. Hung, "Energy-efficient arithmetic using carry-save techniques," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 24, no. 6, pp. 2113–2122, 2016.

[15] S. Kalaiselvi and S. Ramachandran, "Design and implementation of high-speed ALU using Verilog HDL," International Journal of Computer Applications, vol. 97, no. 5, pp. 14–19, 2014.

[16] K. Y. Cho and S. M. Kang, "High-performance multiplier design using carry save adder," IEEE

Transactions on Computers, vol. 45, no. 10, pp. 1151–1157, 1996.
[17] A. Booth, “A signed binary multiplication technique,” Quarterly Journal of Mechanics and Applied Mathematics, vol. 4, no. 2, pp. 236–240, 1951.
[18] S. E. A. Reddy and K. K. Rao, “FPGA implementation of arithmetic logic unit using optimized adder structures,” International Journal of VLSI Design & Communication Systems, vol. 8, no. 3, pp. 25–34, 2017.
[19] A. Sharma and R. Verma, “Performance analysis of ALU using different adder architectures,” IEEE International Conference on Advances in Computing, 2018.
[20] Xilinx Inc., UG901: Vivado Design Suite User Guide – Synthesis, 2020.