Industrial Engineering Journal
ISSN: 0970-2555
Volume : 53, Issue 6, June : 2024

DESIGN AND IMPLEMENTATION OF PARALLEL PREFIX ADDER USING KOGGE-STONE AND SPANNING TREE ADDER

T. Santosh[1], D. Sathwika[2], B. Venkata Kusuma[3], CH. Gowthami [4], B. Poojitha[5]
UG Students[2,3,4,5], Department of Electronics and Communication Engineering, Vignan's Institute of Engineering for Women
[1]Assistant professor**,** Department of Electronics and Communication Engineering, Vignan's Institute of Engineering for Women

ABSTRACT
Adders serve as fundamental components in arithmetic calculations, with parallel-prefix adders recognized for their superior performance in Very Large Scale Integration designs. Specially, Kogge-Stone and Spanning Tree adders are scrutinized for their distinct advantages among parallel-prefix adders, prioritizing faster execution times. Kogge-Stone adders achieve parallelism by efficiently distributing carry signals, while Spanning Tree adders optimize connections to minimize delay. Both approaches are well-suited for high-performance applications like Arithmetic Logic Units (ALUs). In this , implemented both Kogge-Stone and Spanning Tree approaches. Particularly, the Spanning Tree approach showcases efficient addition of 16-bit binary numbers. Through this endeavor, we aim to offer valuable insights into the performance of these approaches via a comparative analysis of Kogge-Stone and Spanning Tree implementations using Verilog HDL mainly focusing on efficiency.
Keywords: Kogge-stone adder, Spanning tree adder, efficiency.

INTRODUCTION
A parallel prefix adder is a super-fast way to add binary numbers (numbers consisting only of 0s and 1s) used in computer chips.Spanning tree is one of the paralle prefix adder.A spanning tree adder is a high-speed digital circuit used for efficient addition of binary numbers in computer hardware. It operates by breaking down input numbers into bits and arranging them in a tree-like structure. This structure facilitates parallel addition, allowing multiple addition operations to occur simultaneously at different levels of the tree. By exploiting parallelism, spanning tree adders greatly accelerate addition processes compared to sequential methods. Clever carry propagation techniques optimize the flow of carry bits through the tree, minimizing delays. As additions progress up the tree, partial sums are combined to generate the final result at the root. Spanning tree adders are crucial components in arithmetic logic units (ALUs) of processors and other digital systems, enabling fast and accurate arithmetic operations. They play a vital role in high-performance computing applications where rapid computation is essential for processing large volumes of data. The spanning tree adder offers several advantages that make it a preferred choice for high-speed addition operations in digital circuits. Firstly, its inherent parallelism allows for simultaneous computation of multiple additions, resulting in significantly faster processing compared to sequential methods. This parallel processing capability enables spanning tree adders to handle large numbers and complex arithmetic operations efficiently, making them ideal for high-performance computing tasks. Additionally, spanning tree adders employ clever carry propagation techniques that optimize the flow of carry bits throughout the tree structure, minimizing propagation delays and improving overall performance. This efficient carry handling ensures accurate addition results while maintaining high speed.

LITERATURE SURVEY
[1] Giorgo Dimitra kopoulos, Kleanthis Papachatzopoulos and Vassilis Paliouras Proposed "Sum Propagate Adders" 2021. From this paper they can say that, Sum propagate adder can improve the speed and reduce the power consumption and it has efficient pipelining structure. This work takes a novel approach by exploring the concept of performing addition by directly propagating the sum bits of previous bit positions rather than carries. By transforming binary carry-propagate addition into an

equivalent sum-propagate addition, a new design space emerges, encompassing ripple-sum to sum-lookahead adders. While sum-propagate and carry-propagate adders possess asymptotically similar area and delay complexity carry-propagate adders currently exhibit superior characteristics when implemented in established technologies. Overall, the paper presents a well-researched and well executed study but it has the limitations that Sum propagate adder is slower than Carry Propagate adder.

[2] B. Lee and V. Oklobdzija in their paper titled "Improved CLA Scheme with Optimized Delay" proposed approach aimed at enhancing the performance of Carry Look-Ahead Adders (CLA). While CLA structures are acknowledged for their speed in addition operations, they are criticized for offering only marginal speed improvements relative to the hardware investment. The authors argue that this inefficiency stems from the suboptimal implementations of common Carry Look-Ahead adder designs. Their analysis suggests that Carry Look-Ahead adders exhibit higher delay compared to Kogge-Stone adders.

[3] Aiswarya Simson; Deepak S "Design and Implementation of High-speed Hybrid Carry Select Adder" 2021. The authors proposed a novel approach. Improve performance i.e less delay and reduce power consumption. The Carry Select Adder (CSLA) stands out for its ability to accelerate various arithmetic functions. To further enhance its speed, a novel approach incorporates multiple high-speed adder logics within a standard Carry Select adder framework. Look Ahead Adders are strategically employed in the initial stages of the modified adder to capitalize on their computational efficiency, especially for fewer bits. The implementation involves a 64-bit hybrid Carry Select Adder on the Xilinx Spartan 6 FPGA development board. The modified hybrid carry select adder not only boosts speed but also demonstrates improved energy efficiency compared to other carry select adders. Overall, the paper presents a well executed study but it has the limitations that carry select adders structure slighter increase in size.

[4] P. Balasubramanian, C. Dang, D. L. Maskell, and K. Prasad, "Approximate ripple carry and carry look ahead adder" 11October 2017. It reduces power-delay compared to Other Adder. Approximate ripple carry adders (RCAs) and carry lookahead adders (CLAs) are presented which are compared with accurate RCAs and CLAs for performing a 32-bit addition. The accurate and approximate RCAs and CLAs are implemented using a 32/28nm CMOS process. The simulation results show that approximate RCAs report reductions in the power-delay product ranging from 19.5% to 82% than the accurate Ripple carry adder for approximation sizes varying from 4- to 20-bits, it is observed that approximate CLAs achieve a 46.5% reduction in PDP compared to the approximate RCAs. Hence, approximate carry lookahead adders are preferable over approximate Ripple carry adder for the low power implementation of approximate computer arithmetic. Overall, the paper presents a well-researched and well executed study but it has the limitations that the approximate Ripple carry adders and carry lookahead adder are influenced by application requirements, between accuracy and efficiency.

[5] A. Tyagi, proposed "A reduced-area scheme for carry-select adders," IEEE Transaction Computing in 1993 . This paper proposes a new reduced-area carry-select scheme where the second copy of the carry-chain is substituted by an 'or' gate per bit position with Area Efficiency-By minimizing the number of transistors or gates. Replacing the ripple-carry blocks with parallel-prefix blocks results in a select-prefix adder, which has a slightly better area and time than a parallel-prefix adder. Overall, the paper presents a well-researched and well executed study but it has the limitations that Complexity-Implementing reduced area schemes may introduce additional complexity to the design process, delay increases when the area is decreased.

EXISTING METHOD

The schematic illustrates the Kogge-Stone adder, a parallel-prefix adder. In the preprocessing stage, propagate (Pi) and generate (Gi) bits are generated using the equations:

$$Pi = Ai \text{ XOR } Bi$$
$$Gi = Ai \text{ AND } Bi$$

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 53, Issue 6, June : 2024

The carry generation block consists of black and gray cells. Black cells compute one set of generate and propagate signals (G, P) from two sets (Gi, Pi) and (Gj, Pj) using the equations:

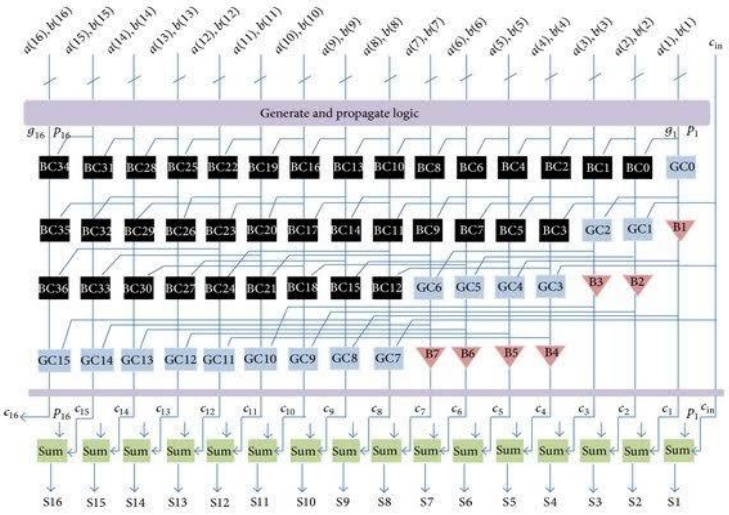
$$G = Gi \text{ OR } (Pi \text{ AND } Gj)$$

$$P = Pi \text{ AND } Pj.$$

Gray cells also compute a set of generate signals (G) using similar equations. In the post-processing stage, the sum (Si) is calculated using the equation:

$$Si = Pi \text{ XOR } Ci\text{-}1$$

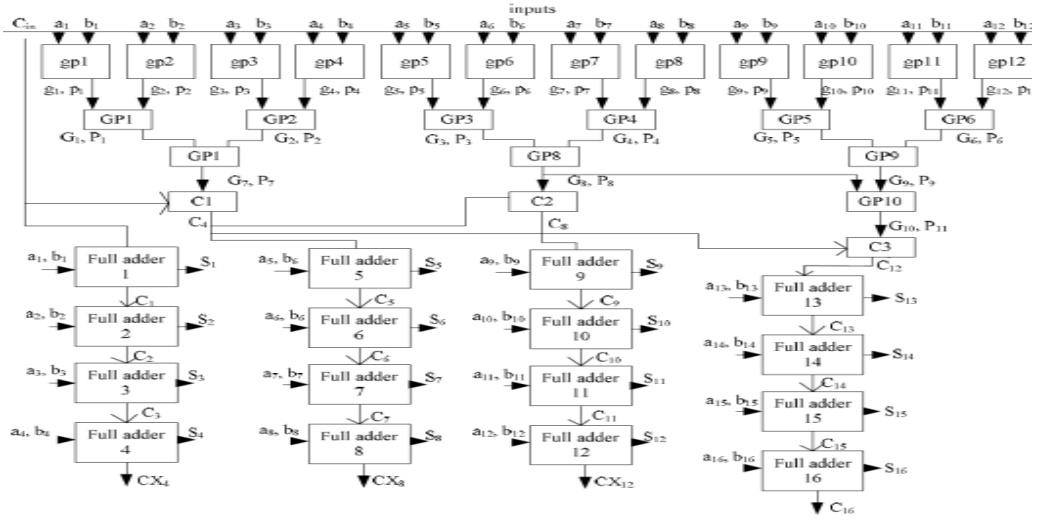
where Ci-1 is the previous carry. Despite its high-speed operation, the Kogge-Stone adder can be complex to implement due to its extensive structure and interconnections, often consuming more chip area compared to a spanning tree adder. This increased area overhead may pose limitations, particularly in applications with constrained chip area.



Figure(1): Schematic diagram of 16-bit Kogge-Stone adder

PROPOSED METHOD

Analog to digital converter (ADC) is the most frequently used analog and mixed-signal circuit for analog to digital data conversion in microprocessors and microcontrollers, DSP architectures, communication devices and consumer electronics applications. It is a strap between analog and digital processing techniques to process the real-world analog signals.



Figure(2): Schematic diagram of 16-bit Spanning Tree adder

- **Input Lines (a and b):**
  These represent the two 16-bit binary numbers (a and b) that will be added together.

- **Propagate (p) and Generate (g):**
  p stands for propagate, which corresponds to the XOR operation of the variables a and b. The XOR (exclusive OR) operation returns true if and only if the operands differ (one is true and the other is false). g stands for generate, which corresponds to the AND operation of the variables a and b. The AND operation returns true only if both operands are true. So, when we say p = a XOR b, it means that the value of p will be true if a and b have different values, and false if they have the same value. Similarly, g = a and b indicates that the value of g will be true only if both a and b are true.

- **P and G:**

The propagator P is determined by the logical AND operation between p1 and p0. Specifically, P will yield a true result only when both p1 and p0 are true. If either p1 or p0 is false, the value of P will be false.

Formula to compute P=p1 AND p0

The Generator G is determined by two conditions: g1: A straightforward OR operation with g1. This means that if g1 is true, G is true regardless of other conditions. g0 and p1: An AND operation between g0 and p1. For G to be true based on this condition, both g0 and p1 must be true.

Formula to compute G=g1 OR (g0 AND p1)

In summary, P represents a logical relationship where both p1 and p0 must be true, while G has two conditions that can make it true: either g1 alone being true or both g0 and p1 being true.

- **Carry (C) Computation:**

The carry computation stage calculates the carry (C) for each bit position based on the signals from Generate (G), Propagate (P), and an input carry (cin). This stage plays a crucial role in determining the carry-out value from each bit position in arithmetic operations. The formula used to compute the carry is:

C=G OR (P AND cin)

Where, G (Generate): The G signal indicates whether there is a generate condition. If G is true, it suggests that the bit position can potentially generate a carry on its own, P (Propagate): The P signal indicates the propagate condition. If P is true, it suggests that the bit position can allow a carry to propagate from a previous bit position, cin (Input Carry): This is the carry-in signal from the previous bit position or an external source. The carry-out C will be true (1) if: There is a generate condition (G is true). There is a propagate condition (P is true) and there is an input carry (cin is true).In summary, the carry computation stage evaluates the conditions for both generation and propagation to determine the carry-out value for each bit, ensuring accurate arithmetic calculations.

- **Full adder:**

A full adder is a digital circuit that performs addition of three input bits: A, B, and an input carry (Cin). It produces two outputs: a sum (S) and an output carry (Cout). The full adder is a fundamental component in arithmetic logic circuits, used extensively in digital systems like CPUs and ALUs (Arithmetic Logic Units).

Sum =A XOR B XOR Cin
Cout= (A AND B) OR Cin (A XOR B)

DESIGN STRUCTURE OF THE SPANNING TREE ADDER

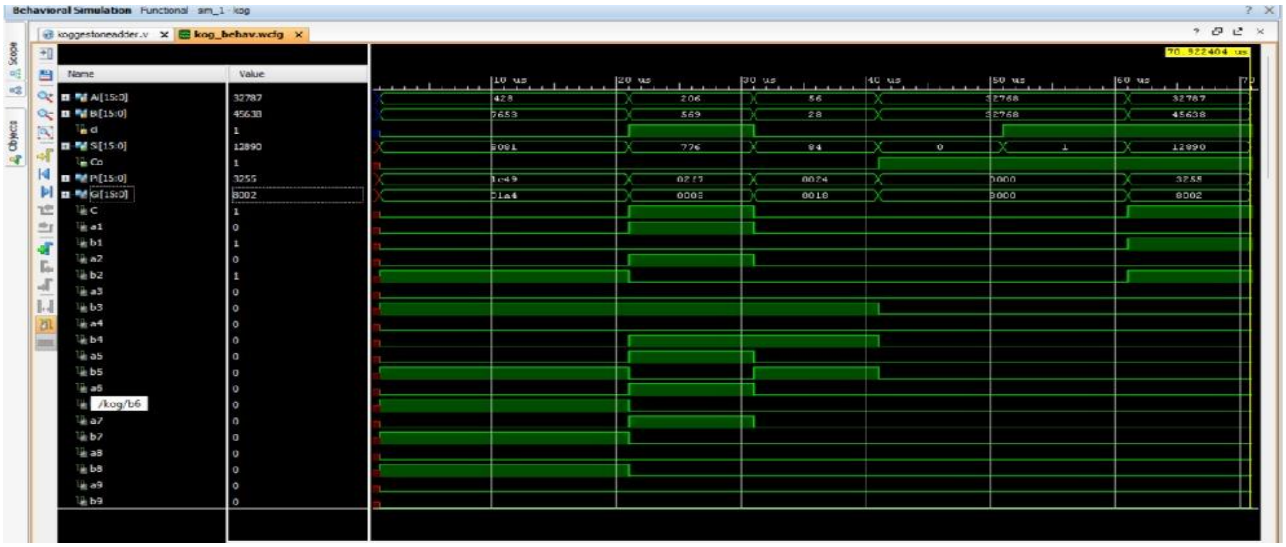
The spanning tree adder (STA) employs a hierarchical tree-like structure to perform addition operations efficiently. The STA begins by decomposing the binary numbers to be added into individual bits. Each bit of the input numbers is processed independently within the adder. The decomposed bits

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 53, Issue 6, June : 2024

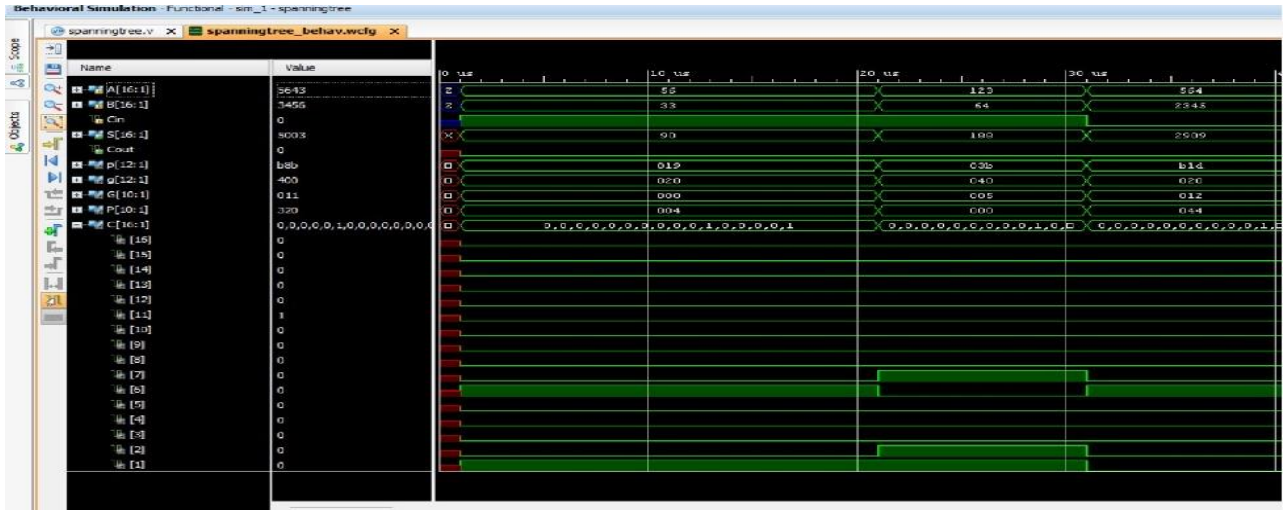
are then organized into a tree structure, often resembling a binary tree. This tree structure forms the backbone of the STA's design. The tree is divided into multiple levels, with each level representing a stage of the addition process. The number of levels in the tree corresponds to the number of bits in the input numbers. At each level of the tree, addition operations are performed in parallel. This means that multiple bits are added simultaneously, exploiting parallelism to enhance speed and efficiency. The STA employs techniques to optimize carry propagation throughout the tree structure. These techniques minimize carry propagation delays, ensuring that the addition process proceeds swiftly and accurately. As addition progresses through the levels of the tree, partial sums are generated at each stage. These partial sums represent intermediate results of the addition process. The partial sums generated at each level are combined as they ascend through the tree structure. This combination process continues until the final sum is obtained at the root of the tree. The final sum obtained at the root of the tree represents the result of adding the input numbers together. This result is the output of the spanning tree adder.

RESULT ANALYSIS

The schematic diagrams of the adders discussed above are simulated in Xilinx Vivado using Verilog HDL.The output wave forms of the16-bit Kogge-Stone adder are shown in Figure (3) and the output wave forms of 16-bit spanning tree adder are shown in Figure (4). The power, area and delay of the different adders are compared for best utilization. The comparative analysis of two 16 bit parallel prefix adders is shown in terms of Total on-chip power, Delay and Number of Look Up Tables (LUT's) as shown in table



Figure(3): Output Wave forms of 16-Bit Kogge-Stone Adder



Figure(4): Output Wave forms of 16-Bit Spanning Tree Adder

TYPE OF PARALLEL PREFIX ADDER	TOTAL ON CHIP POWER	DELAY in ns	NUMBER OF LUTS
16 bit Kogge-Stone adder	11.142Watts	18.326ns	24
16 bit Spanning tree adder	11.594 Watts	5.294ns	23

Table 1: Comparison of two 16-bit parallel prefix adders

CONCLUSION

In this paper, by evaluating both the Kogge-Stone adder and the Spanning Tree adder architectures. The 16-bit Kogge-Stone adder demonstrated an 18ns delay and utilized 24 Look-Up Tables (LUTs). However, the higher LUT count suggests that this design may occupy more FPGA resources. The 16-bit Spanning Tree adder presented a faster delay of 5ns with only 23 LUTs. This design achieves its efficiency by using a spanning tree topology. On comparing the two, the Spanning Tree adder offers both quicker performance and better LUT efficiency than the Kogge-Stone adder. In conclusion, the selection between these two adders should be based on the specific requirements of the project. Careful consideration of these factors will enable designers to choose the most suitable adder for their application.

REFERENCES

1. Dimitrakopoulos, G., Papachatzopoulos, K., & Paliouras, V. (01 July-Sep 2021). Sum propagate adders. IEEE Transactions on Emerging Topics in Computing, 9(3), 1479–1488.
2. B. Lee and V. Oklobdzija, “Improved CLA scheme with optimized delay,” J. VLSI Sign Process. Syst. Signal, Image Video Technol, no. 3, pp. 265–274, 1991.
3. M. Lehman and N. Burla, “Skip techniques for high-speed carry-propagation in binary arithmetic units,” IRE Trans. on Electron. Comput., vol. EC-10, no. 4, pp. 691–698, Dec. 1961.
4. V. Kantabutra, “Designing optimum one-level carry-skip adders,” IEEE Trans. Comput., vol. 42, no. 6, pp. 759–764, Jun. 1993.
5. J. L. S. Weinberger, “A logic for high-speed addition,” in Proc. Nat. Bureau Stand. Circ. 591, 1958, pp. 3–12.
6. R. E. Ladner and M. J. Fisher, “Parallel prefix computation,” J. ACM, vol. 27, no. 4, pp. 831–838, Oct. 1980.
7. G. Dimitrakopoulos and D. Nikolos, “High-speed parallel-prefix VLSI ling adders,” IEEE Trans. Comput., vol. 54, no. 2, pp. 225–231, Feb. 2005.
8. T. Han and D. Carlson, “Fast area-efficient VLSI Adders,” in Proc. IEEE Symp. Comput. Arithmetic, 1987, pp. 49–56
9. V. G. Oklobdzija, “High-speed VLSI arithmetic units: Adders and multi_x0002_pliers,” in Design of High-Performance Microprocessor Circuits.New York, NY, USA: IEEE Press, 2001, pp. 181–204.