Ripple-Carry Vs Carry-Lookahead: A Study of Delay–Area–Power Trade-Offs

Huawei Liu *

College of Letters and Science, University of California, Santa Barbara, Goleta, CA, 93117, United States

* Corresponding Author Email: hliu623@ucsb.edu

Abstract. Since adders are located on the critical paths of arithmetic units, the choice between Ripple-Carry Adders (RCA) and Carry-Lookahead Adders (CLA) has first-order impacts on clock frequency, silicon area, and energy. This paper takes a look back at RCA/CLA and all its variants to see what has happened recently (2021–2025). The research summarizes results from publicly available, peer-reviewed research on EDP, area, power, and improved CLA structures (e.g., hierarchical and non-uniform grouping), hybrid organizations (e.g., carry-select adders using 4-bit RC/CLA generators), approximate and asynchronous CLAs for event-driven or error-tolerant workloads, and final considerations (e.g., fan-in limits and routing congestion) as they pertain to implementation. Through all investigations, CLAs have been shown to reduce critical paths and enable more stringent timing. For smaller word sizes or less stringent clock requirements, RCAs typically outperform in terms of area and dynamic power. Word size, activity factors, cell-library constraints, and flow settings determine the apparent break-even threshold. The paper's last remarks include some helpful hints for selection as well as some outstanding questions, with an emphasis on the potential for non-uniform grouping and comparisons with parallel-prefix adders.

Keywords: Ripple-Carry Adder; Carry-Lookahead Adder; Delay-Area-Power Trade-off; Energy-Delay Product; VLSI Adders.

1. Introduction

From the integer ALU to the last accumulation stage of multiplier and multiply-accumulator, arrange for a spreading of adder circuits along a critical path of an arithmetic unit. Microarchitecture decisions directly affect clock frequency, size, and energy efficiency. To illustrate, an adder exposes a commonly known trade-off: the Carry-Lookahead Adder (CLA) selects the carry signal through generation and propagation logic, whereas the Ripple-Carry adders (RCA) route the carry signal linearly over n-stage full-adders, creating a small and simple-to-route structure. Although more gates are needed and fan-in/routing pressure rises, CLA decreases logic depth (roughly logarithmic to word length). Contemporary VLSI designs often embed CLAs in high-performance datapaths or in the last reduction stage of tree multipliers, while RCAs remain attractive in low-power or small-area contexts where timing is relaxed [1, 2].

Both sides of this trade-off have been optimized in recent studies. In 2022, Balasubramanian and Mastorakis implemented a fast and energy-efficient CLA architecture using well-thought-out prefetch logic [3], while in 2024, Balasubramanian and Maskell proposed a non-uniform grouping strategy that reduces latency without appreciably increasing space requirements [4]. Asynchronous CLA variants can outperform synchronous pipeline systems and optimize speed and energy in event-driven scenarios, as demonstrated by Balasubramanian and Liu [5]. Wide-word adders are increasingly using hybrid architectures. In order to balance latency and area/power, Hasan et al. built a carry selection framework using a 4-bit RC/CLA carry generator [6]. According to Akbar, Wang, and Bermak, speed and fault tolerance in the RCA domain can be increased at a fair cost by appropriately parallelizing the structure and error detection logic [7]. In their study of approximate/adaptive CLA forms for fault-tolerant workloads, Joshi and Mane demonstrated how the Pareto frontier is shifted by precision relaxations [8]. In conclusion, these studies demonstrate that the degree of CLA time advantage and the possibility of reducing energy consumption or the energy-delay product (EDP) are greatly

influenced by bit width, activity factor, routing constraints, and implementation flow. The benefits of CLA in terms of time performance are also supported by these studies.

However, most of the comparisons in the published literature are conducted in a number of different settings, such as with different foundry libraries or process nodes, tool versions, time constraints, and even evaluation levels (behavioral versus post-layout simulation). Determining when basic RCA is still Pareto-optimal and when the reduced carry depth of the CLA exceeds the additional logic and routing costs is difficult due to this variability. A comprehensive overview of existing, comparable findings is highly beneficial during the initial phases of training and in the context of architectural decision-making. This overview synthesizes findings by systematically organizing and comparing results from studies characterized by diverse processes and limitations.

This article functions primarily as a literature review rather than an empirical investigation. This study analyzes peer-reviewed, openly accessible publications on RCA, CLA, and associated variants released from 2021 to 2025. The paper contains a summary of the basics and terms; a classification of the most recent improvements in refined CLAs, hybrid structures, and asynchronous/approximate designs; a comparison of reported time, area, and power trends to show a break-even range that depends on frequency and word size; and the extraction of useful selection criteria and open questions for future research. The next part of the article is set up like this: Section 2 talks about the background and common architectures. Section 3 talks about different application scenarios, and Section 4 talks about problems and future possibilities. Section 5 wraps things up and goes over the main points.

2. Mainstream Architectures

2.1. Ripple-Carry Adder (RCA)

The ripple-carry adder is a simple way to do things, and it is still a stable base for many designs that have to deal with cost or performance limits. A ripple-carry adder connects a number of full adders (FAs) so that the carry out of stage i goes to the carry in of stage $i+1$. The generation term is $g_i=a_i\,b_i$, and the propagation term is $p_i=a_i \oplus b_i$. The recursive equation $c_{i+1}=g_i \vee (p_i \wedge c_i)$ gives us the sum $s_i=p_i \oplus c_i$. The logic depth rises about linearly with the word size n because each stage depends on the previous carry. RCA is not a good choice for demanding clock targets because it has a linear dependence when n is large. Still, its local wiring, low gate count, and predictable placement give it constant benefits in terms of silicon area and often in terms of dynamic performance, especially when the clock period is relaxed. These traits are why RCAs are still the preferred choice for microcontrollers, near-threshold designs, and small data path segments where other units make up the critical path. Because there is no global lookahead, the routing network follows the order of the bits and doesn't often travel very far. This helps lessen the amount of traffic and interference on the clock tree. Designers often use this simplicity by putting carry chain resources on FPGAs or choosing high-density standard cell libraries on ASICs. However, if the fan-out grows, you may need to make the chain bigger or add repeaters to close the timing. These answers are easy to understand and follow. Current research endeavors in RCA predominantly concentrate on resilience and moderate parallelization to mitigate worst-case delays with minimal overhead [7].

2.2. Carry-Lookahead Adder (CLA)

Carry-lookahead adders make logic less complicated by calculating carries directly from the generation and propagation signals instead of waiting for their delay. You can write the carry outputs in closed form within a 4-bit block. The logical combination $g_3 \vee p_3 g_2 \vee p_3 p_2 g_1 \vee p_3 p_2 p_1 g_0 \vee p_3 p_2 p_1 p_0 c_0$ gives us the block carry c_4. The block also allows for group propagation $P_g=p_3 p_2 p_1 p_0$ and group generation $G_g=g_3 \vee p_3 g_2 \vee p_3 p_2 g_1 \vee p_3 p_2 p_1 g_0$, which makes hierarchical lookahead functions possible. Wide adders, like 16 or 32 bits, put together several 4-bit blocks and

add one or two levels of the group lookahead function. This makes the longest chain grow at $O(\log n)$ instead of $O(n)$.

More inputs and longer global connections can make carry lookahead adders (CLA) work less well. When using AND/OR trees for group and propagation generators, designers often break down expressions and limit the number of inputs to four because of some restrictions. This does add more logic stages, but it makes assigning tasks easier. The global routing properties of group and propagation generators can also make connections longer and add more buffers, especially when the data word sizes are bigger. Even with these problems, carry-lookahead adders are still the preferred choice for high-performance data paths and the last steps of tree multipliers. They allow for tighter timing without using a deep pipelining structure [1, 2].

2.3. Different Types and Current Trends

2.3.1 Uneven grouping in carry-lookahead blocks

A current research focus is on refining the organization of lookahead blocks to enhance the adjustment of sensitivity to delays across various bit positions. The research conducted by Balasubramanian and Maskell demonstrates that uneven grouping—where deeper lookahead resources are assigned to more time-sensitive subdomains while maintaining smaller sizes for other domains—can mitigate delay and energy consumption in comparison to even grouping within the same domain [4]. This method works well when there aren't many fan-in limits or routing budgets because it puts lookahead resources where they will do the best.

2.3.2 CLA organizations that are efficient and care about energy

Balasubramanian and Mastorakis introduce design methodologies intended to factorize lookahead equations and constrain the fan-in within the foundational AND/OR trees [3]. This makes it possible to build a fast and energy-efficient carry lookahead adder (CLA). One important thing is to find the best logical depth and input load so that the structure can use the fastest cells without making extra copies of them. The research findings demonstrate that a meticulously engineered CLA can attain optimal timing for intricate prefix adders while ensuring a modest escalation in area and power consumption.

2.3.3 An asynchronous carry-lookahead adder for workloads that are event-driven

To reduce the maximum delay, synchronized carry-lookahead adders (CLAs) are used. But a lot of applications have activities that are not regular or depend on the data. Balasubramanian and Liu suggest using asynchronous CLAs, which keep the benefit of lookahead but use local handshakes and completion detection instead of global clocking [5]. In event-driven situations like this, energy use is kept to a minimum because state changes only happen when work is being done. The average latency can show the actual signal delay instead of just the theoretical maximum. But this makes the interface logic and verification more complicated, so these designs are better for specialized accelerators than for general-purpose processor cores.

2.3.4 Hybrid carry-select systems that integrate RC/CLA generators are the subject of this study

Hybrid systems combine the best features of different architectures. Hasan et al. build a carry-select adder (CSLA) in which a small 4-bit RC/CLA generator makes the carry for each block [6]. This architecture uses guesses about carry values and precomputed sums to shorten the effective carry path and avoid the cost of a large global lookahead network. It is clear that hybrid systems work well, especially for longer words and faster clock speeds. This is because pure ripple-carry adders can't meet the timing requirements and pure carry-lookahead adders need a lot of wires.

2.3.5 Approximate/Adaptive CLAs for Fault-Tolerant tasks

Approximate or adaptive carry-lookahead adders (CLAs) trade accuracy for energy savings when working with multimedia, inference, or sensor workloads that can't handle arithmetic errors very well.

Joshi and Mane examine methods that facilitate carry computation at non-essential bit positions or transition between various precision modes during runtime to alter the Pareto frontier in the delay-power domain [8]. The benefits gained are substantial, particularly when error budgets are clearly delineated. But these kinds of designs don't work for control logic or data paths that need to be very precise.

In short, RCA has little logic and wiring that grows linearly with word size and causes delays. CLA, on the other hand, lowers the theoretical transfer depth by making and sending logic, but this increases the overhead for fan-in and routing. Current research contributions enhance this equilibrium via uneven grouping, energy-efficient logic structuring, asynchronous operation, and hybrid CSLA forms. Also, approximate and adaptive variants create a new design space for applications that can handle errors. These changes show that the choice between RCA and CLA depends on more than just the word size and clock target. It also depends on activity factors, fan-in limits, and routing budgets, which are all talked about in the technical literature.

3. Areas Where it Can be Used

3.1. General Data Paths: Cpus, Microcontrollers, and Address Generation

In regular processors, adders are in the arithmetic logic unit (ALU), the branch/comparison logic, and the address generation units. The clock frequency and word size are the main things that determine whether to use a ripple-carry adder (RCA) or a carry-lookahead adder (CLA). The CLA's lower logical depth makes it easier to close timing on the ALU's critical path and finish the last steps of multiplication operations in CPU cores with deep pipelines that need to support high clock frequencies. At 32 bits and up, the extra area and routing overhead of the CLA is usually fine, especially when speed is the most important thing. Microcontrollers and embedded CPUs, on the other hand, usually run at moderate clock speeds and focus on low leakage currents and small layouts. Ripple-carry adders (RCAs) are appealing in this context because their limited wiring and few gates cut down on both dynamic power and silicon area. Similar things can be said about address generation: when caches or memory interfaces need more generous timing margins, RCAs can save power, but when the L1 cache needs tighter timing cycles, designers have to use carry-lookahead adders (CLAs) or hybrid circuits.

Another useful thing to think about is how predictable the design is. Because RCA timing scales linearly with word size, it is relatively easy to make timing corrections in later stages (for example, by only increasing the size of the chain). The timing of the CLA, on the other hand, depends on how many lookahead signals are in a group, how close they are to each other, and how many signals can be in a group at once. So, the completion of it is more affected by decisions about floor plans. Because of this, teams sometimes use an RCA for ALU slices that aren't as important and save the CLA for the add/sub block that sets the core clock. This is how a good balance between energy efficiency and headroom is sought.

3.2. Signal Processing and AI Accelerators: Multipliers, MAC Arrays, and Mixed Solutions

Adders are very important in tree multipliers and multiply-accumulate (MAC) data paths for digital signal processing (DSP) blocks and AI accelerators. The carry-save form is used to first reduce the partial products. The final carry-propagate adder (CPA) decides how long the stage will take. For medium-sized words, a hierarchical carry-lookahead adder (CLA) is usually better because it cuts down on the maximum delay without adding the extra work that fully parallel prefix trees do. As array sizes grow or frequency targets rise, hybrid structures like carry-select adders (CSLA) with efficient RC/CLA carry generators become appealing. They lower the effective carry path through speculation while keeping block-level costs in check. CLA-based CPAs are often used in accumulators when throughput-oriented MAC arrays work at fixed, moderately high frequencies.

The load from work and activity factors is very important. MAC arrays that work with dense tensors switch very quickly. In these situations, the lower capacitance and local wiring of RCAs can

reduce dynamic power, as long as the timing is kept the same. On the other hand, structures that don't require unnecessary switching are better for sparse or intermittent workloads. Asynchronous or gated-clock CLA variants use event-driven behavior to make the system more energy-efficient while keeping the lookahead speed. Fault-tolerant inference pipelines broaden the design spectrum by enabling approximate/adaptive CLAs to reduce precision at certain bit positions, thereby shifting the Pareto frontier towards lower power and latency, provided that application-level quality metrics stay within acceptable thresholds.

3.3. Implementation Platforms and Physical Considerations (ASIC vs. FPGA)

The features of platforms play a big role in the choice between RCA and CLA. When it comes to ASICs, CLAs are usually the preferred choice, especially when timing budgets are tight. Still, their fan-in and global connections mean that there are more buffers and a longer overall wire length. If a library limits the gate fan-in to four inputs, the lookahead equations need to be changed. This could lower the theoretical depth gains if the grouping isn't done right. So, floor planning and overloading are very important for CLA efficiency. Designers often use uneven grouping to put the deeper lookahead only where delays are really important. When operating points are near the threshold or energy minimization is desired, it is possible to see less energy used per operation because RCAs are simple and have limited capacitance, especially for word lengths that are small to medium.

Field-programmable gate arrays (FPGAs) have carried chain primitives that are specially optimized for fast and space-efficient ripple-carry addition. On the other hand, carry-lookahead adders (CLAs), which are based on wide AND/OR trees and long connections, usually have a less useful mapping because they can't make the best use of the hard carry chain. This is why many FPGA designs use ripple-carry adders (RCAs) or RC-based carry-select adders (CSLA hybrids), even though their ASIC counterparts would use CLAs. In the end, the break-even behavior on both platforms depends on three physical things: (i) the ratio of the target clock period to the word size, (ii) the activity profile of the operands, and (iii) the routing budgets of the technology that was chosen. With a relaxed target clock period or high activity, RCAs usually use less power and space. At higher frequencies or longer word lengths, CLAs and their improved versions give enough time margin.

The ripple carry adder (RCA) is best used in low-power microcontroller units (MCUs), FPGA mappings, ASICs that are close to the threshold, and non-critical data path slices. On the other hand, carry lookahead adders (CLA) and their improvements are the most important parts of arithmetic logic units (ALUs), carry-propagate adder (CPA) stages in high-speed multipliers/MACs, and designs that need to work with very fast clock speeds. To find a balance between area and power use and frequency, hybrid and approximate methods are used. The characteristics of the platform (ASIC vs. FPGA) often have a bigger effect than pure logic theory would say.

4. Suggestions and Discussions

4.1. Problems

A major problem with comparing ripple-carry and carry-lookahead families is that the technical literature doesn't always use the same methods. A lot of research findings come from different versions of tools, technology nodes, and standard cell libraries. Some studies only look at behavioral simulations, while others also look at analyses after the layout. This variety makes it hard to say that the speed-up of lookahead logic is only due to differences in how corners are chosen, how maps are made, or how hard the back-end works. To make a meaningful consolidation of previous work, it is necessary to fully normalize or at least clearly disclose PVT corners, timing specifications, I/O loads, and placement seed values.

Cell libraries make it even harder to design a CLA (carry lookahead adder). Wide AND/OR trees are a natural result of lookahead equations, but most libraries only allow four inputs. So, designers have to factor expressions and balance trees, which can either make the logical path longer or make the area bigger if done wrong. The performance that results depend a lot on algebra, synthesis, and

placement heuristics. This is why some studies show that CLA implementation leads to significant improvements, while others show that the improvements get smaller as the word size increases.

Another thing that makes things uncertain is routing and buffering. Signals that are sent and made in groups can travel a long way. This often means that the extra cost of CLA is mostly due to wire RC and buffer insertion beyond 32 bits. On the other hand, RCA wiring is only local and usually scales in a way that is easy to predict. Energy comparisons are also very sensitive to how you think about activity factors. Some studies employ a 50% random toggle model, while other methodologies rely on application traces characterized by markedly diminished switching. Because energy-delay product scales with both delay and toggle-dependent power depending on switching, the results may be different for high and low workloads. Lastly, the paper needs to think about the effects of the platform. Field-programmable gate arrays (FPGAs) prefer very clear ripple structures with dedicated carry chains, while application-specific integrated circuits (ASICs) prefer lookahead with strict clock specifications. General conclusions across various platforms can be deceptive if the underlying mapping substrate is not clearly defined. Not enough research has been done on marginal factors like operating points close to the threshold, process variation, and aging, even though they could make RCA's energy advantage bigger or limit CLA's time windows.

4.2. Perspectives and Practical Advice

In conclusion, the evidence indicates that context-dependent decision-making is superior to a universal solution. Ripple-carry adders are great for data paths at the microcontroller level, address generation, and non-critical ALU slices where the clock speed is lower and the main goal is to save silicon space or improve performance. Because of their strictly local wiring and few gates, their timing is predictable and scales linearly with word size. When frequency requirements are stricter or wide carry propagate stages determine the pipeline rate, like in high-performance ALUs and the last stage of tree multipliers and MAC arrays, hierarchical carry lookahead structures are better. When global lookahead is expensive and pure ripple addition can't keep up with timing, hybrid structures like carry-select adders with small RC/CLA generators can be a good compromise. Approximate or adaptive CLAs are only good for programs that can handle mistakes and have clear quality budgets.

Engineers who choose carry lookahead adders (CLAs) can use a number of different methods to make them work. One good way to do this is to only give deeper lookahead to the bit ranges that are most time-sensitive, which are usually the upper segments, and keep the other ranges lean. It is best to factorize AND/OR trees to take into account fan-in limits and map them to balanced stages instead of very wide gates. When placing things, it's important to put group drivers and consumers in the same place and to plan for buffers. In addition to delay, area, and power, the line length and number of buffers should be watched to make the real costs of lookahead clear.

More comparable reporting would help the community. Future research should employ a comparison matrix that incorporates a standard cell library, a PVT corner, uniform I/O utilization, and a summary of target periods. It should also be made public what the worst and total negative slack values are, timing violations should be noted, and the results should be averaged over several place-and-route seeds. There should be at least two activity profiles (dense and low-toggle) that show power use. The PPA figures should also include routing proxies like total wire length and inserted buffers. Publishing scripts and configuration files would make it even easier to reproduce.

Different research methods look like they could be useful. The incorporation of routing-aware CLA synthesis may integrate placement cues and connection models into the factorization of lookahead equations. Automatic, non-uniform grouping, managed by search or learning techniques, could optimize group boundaries for a particular layout and a defined fan-in budget. Selection models that consider workload and DVFS (dynamic voltage and frequency scaling) could convert activity statistics and voltage/frequency pairs into a basic choice between RCA, CLA, and CSLA. For bursty, event-driven accelerators, researchers should look into asynchronous or gated-clock hybrids. They should also look into how well they work across corner points, including when they are near the threshold and as they get older. Lastly, future work should look at how well a carefully designed CLA

compares to parallel prefix adders to see when it is still the preferred choice, even when taking wiring costs into account.

In real life, a simple rule of thumb can help you make decisions early on. RCA usually uses the least amount of area and power when the target period is much longer than the FO4-scaled (fan-out-of-4 delay) depth of a ripple chain for the desired word length. For tighter frequency targets or longer word lengths, hierarchical CLA with uneven grouping is the safer choice. CSLA-style hybrids fill the gap between the two. This viewpoint brings together the different results from the literature and gives useful advice on how to choose adders for real-world designs.

In designs where safety or reliability are very important, classical ripple chains fail at the first failure stage. But a self-repairing carry-lookahead adder with hotstandby, fault localization, and partial reconfiguration keeps throughput under failures at 64-bit scale [9]. Even though the gain isn't "free" because the area increases and the control logic gets more complicated, a self-repairing CLA is still the preferred choice when availability and worst-case delay are the most important factors. If you need ASIL/DO-class reliability or graceful degradation, plan for the area overhead and pick CLA with repair over RCA with detection-only [9].

The energy-delay Pareto curve of an RCA can be quantified at the device/cell level by employing hybrid full-adder (HFA) cells, such as XOR-XNOR-based or transmission-gate variations, which result in iso-energy shorter delay or lower energy [10]. Thus, the RCA-CLA break-even threshold can be adjusted by simply changing the libraries used, without requiring any modifications to the design itself. Rule of thumb: check the target library for HFA options and reevaluate EDP before claiming victory for CLA at a certain word size/frequency; if realistic cell choices are taken into consideration, the outcome can be reversed [10].

5. Conclusion

This study analyzes the design parameters of ripple-carry adders (RCAs) and carry-lookahead adders (CLAs), consolidating contemporary research findings from 2021 to 2025 regarding timing, area, performance, and the energy-delay product (EDP). The main point of this is to look at the basic trade-off between RCAs, which make logic and wiring less complicated but make the carry path longer with word size, and CLAs, which make logic depth less complicated through generate/propagate (lookahead) calculations but require more fan-in and global routing. The literature is categorized into advanced CLA organizations (featuring non-uniform grouping), hybrid carry-select configurations with compact RC/CLA generators, asynchronous variants for event-driven environments, and approximate/adaptive methodologies for fault-tolerant applications. The results show that CLAs are better when short cycle times are needed. RCAs are very useful in places where there isn't much space or energy use and the time frames are flexible, especially when there isn't a lot of data. Hybrid approaches connect these two areas, and approximate/adaptive designs push the Pareto frontier further when clear quality goals are set.

Another conclusion has to do with the platform and the physical setting. The benefits of carry-lookahead adders get better as frequency targets go up, but fan-in limits and the high cost of long group signal lines make them less useful. Dedicated carry chains in FPGAs make the balance shift a lot more toward ripple structures or hybrid structures that use RC elements. In both instances, the perceived trade-off is contingent upon variables such as word size, target period, operand activity, and routing budget; no singular architecture unequivocally prevails.

The current study possesses methodological deficiencies that hinder direct comparisons among the examined studies. The intrinsic logical advantages of the two families can be influenced by variations in technology nodes, libraries, tool versions, constraint sets, and even the evaluation level (behavioral versus post-layout). To enhance the depth of knowledge, forthcoming research should emphasize the establishment of standardized reporting mechanisms—encompassing a unified library and corner, uniform I/O loads, diverse place-and-route seeds, clear WNS/TNS (worst/total negative slack)

indicators, and power management across a minimum of two activity profiles—alongside the public dissemination of scripts and configuration files.

There are many promising areas of research that are opening up for future growth. A routable CLA synthesis that improves factorization and placement could help cut down on the number of buffers and the length of the cable. Automatic, search- or learning-driven, non-uniform grouping could create efficient hierarchies for a specific floorplan and budget for fan-in. Selection models that combine workload and DVFS capabilities could use activity statistics and operating points to choose between RCA, CLA, and carry-select hybrid architectures. Asynchronous or gated-clock CLAs warrant additional examination as burst accelerators, necessitating a systematic characterization of their robustness in various domains, including near-threshold operation, variation, and aging. Finally, objective comparisons with parallel prefix adders will show when a well-designed carry-lookahead adder is still the preferred choice after taking into account the costs of interconnects.

In conclusion, the selection among RCA, CLA, and hybrid architectures is fundamentally contingent upon particular contexts. This report gives you the most up-to-date information and clear suggestions: RCA is best for words that are small to medium in length and have slow clock speeds. Hierarchical CLA is better for longer word lengths or tighter timing. In the transition range, hybrid architectures should be thought about. By using these rules along with clear and comparable evaluation methods, developers can choose the right adder architectures that meet timing needs without wasting space or energy.

References

[1] Jiménez A, & Muñoz A. VLSI implementation and performance comparison of multiplier topologies for fixed- and floating-point numbers. Applied Sciences, 15(9), 4621, 2025.

[2] Thamizharasan V, & Parthipan V. Design of efficient binary multiplier architecture using hybrid compressor with FPGA implementation. Scientific Reports, 14, 8492, 2024.

[3] Mastorakis N E et al. High-speed and energy-efficient carry look-ahead adder. Journal of Low Power Electronics and Applications, 12(3), 46, 2022.

[4] Balasubramanian, P et al. A new carry look-ahead adder architecture optimized for speed and energy. Electronics, 13(18), 3668, 2024.

[5] Liu W et al. High-speed and energy-efficient asynchronous carry look-ahead adder. PLOS ONE, 18(10), e0289569, 2023.

[6] Hasan M, Chowdhury S, Faruqe O, et al. Wide word-length carry-select adder design using ripple-carry and carry look-ahead method-based hybrid 4-bit carry generator. Engineering Reports, 6, e12721, 2024.

[7] Akbar M A, Wang B, & Bermak A. A high-speed parallel architecture for ripple carry adder with fault detection and localization. Electronics, 10(15), 1791, 2021.

[8] Joshi V, & Mane P. Novel approximate adaptive carry lookahead adder for error-resilient applications with generic method for error analysis. Scientific Reports, 15, 19215, 2025.

[9] Akbar M A, Wan, B, & Bermak A. Self-Repairing Carry-Lookahead Adder With Hot-Standby Topology Using Fault-Localization and Partial Reconfiguration. IEEE Open Journal of Circuits and Systems, 3, 50–58, 2022.

[10] Giustolisi G, Palumbo G, & Spagnuolo G. Hybrid Full Adders: Optimized Design, Critical Review and Automated Selection. Electronics, 11(19), 3220, 2022.