8-bit Full Adder Design Using NAND-Based Carry Look-Ahead Architecture

First A. Author, *John Gellerup; sqv6@txstate.edu*, Second B. Author, *Brandon Markham; uzy7@txstate.edu,* and Third C. Author, *Cassidy Miskovitz;* fae17@txstate.edu

***Abstract*— This paper presents the design and physical layout of an 8-bit full adder using a Carry Look-Ahead Adder (CLA) architecture constructed entirely from NAND gates. The design emphasizes the advantages of the CLA architecture over the traditional Ripple Carry Adder (RCA), focusing on propagation delay and overall performance as reported in existing literature. LTSpice is used for schematic-level verification, and Microwind is employed for layout implementation and validation. While no direct simulation-based comparison is made, the paper draws on established references to highlight the CLA's parallel carry computation and its impact on speed. The results illustrate the trade-offs and benefits of a NAND-only CLA implementation in digital VLSI design.**

I. INTRODUCTION

As digital systems scale, optimizing arithmetic logic becomes increasingly critical to achieving high performance. Among these arithmetic units, binary adders are foundational components in applications ranging from embedded systems to signal processing. The Ripple Carry Adder (RCA) remains a common design due to its simplicity and area efficiency, but it suffers from timing limitations caused by sequential carry propagation. This paper presents the Carry Look-Ahead Adder (CLA) as a more performance-oriented alternative, offering faster computation through parallel carry generation. The CLA is implemented exclusively using NAND gates, utilizing their universality in digital logic. Prior research highlights the CLA's advantages in reducing propagation delay and enhancing scalability compared to the RCA [1]. Design and layout are completed using LTSpice and Microwind, with results emphasizing the feasibility of a NAND-based CLA for VLSI implementation.

II. OVERVIEW AND DESIGN BACKGROUND

In digital logic, a 1-bit full adder is the foundational element for multi-bit addition. It adds three binary inputs (two operands and a carry-in) and outputs a sum and a carry-out. The logical structure of a full adder can be understood through its gate-level representation, where the sum is generated by a pair of XOR gates. The carry-out is generated by a combination of AND and OR gates, illustrated in Figure [2].

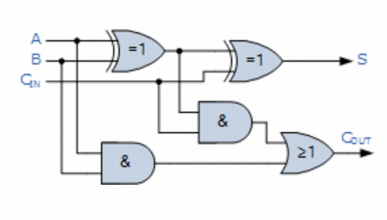


Fig. 1. Logic gate diagram of a 1-bit full adder showing inputs A, B, Cin, and outputs Sum and Cout.

Cascading multiple full adders forms a complete multi-bit adder. The Ripple Carry Adder (RCA) chains these adders sequentially, with each carry-out feeding the next stage's carry-in. While the RCA is easy to implement and area-efficient, its major drawback is that the carry signal must propagate through each stage sequentially. This leads to a cumulative delay that grows linearly with the number of bits, as shown in the block diagram of an 8-bit RCA in Figure 3 [3].

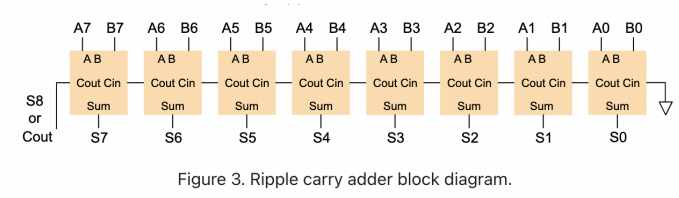


Figure 3. Ripple carry adder block diagram.

Fig. 2. Block diagram of an 8-bit Ripple Carry Adder where each full adder's carry-out feeds into the next stage.

The Carry Skip Adder (CSA) improves upon the delay limitations of the Ripple Carry Adder by reducing the dependency on sequential carry propagation. Instead of waiting for the carry-in to resolve, each stage of the CSA computes two possible sums in parallel, one assuming a carry-in of 0 and another assuming a carry-in of 1. Once the actual carry-in is known, a multiplexer selects the correct result. This parallelism allows the CSA to achieve faster performance than the RCA, particularly for larger bit widths.

The Carry Look-Ahead Adder (CLA), by contrast, introduces propagate and generate logic to compute all carries in parallel. These signals allow the adder to anticipate carry values without waiting for them to ripple through each stage. Figure 3 shows a conceptual CLA architecture with grouped blocks, generate/propagate units, and parallel carry logic [4].

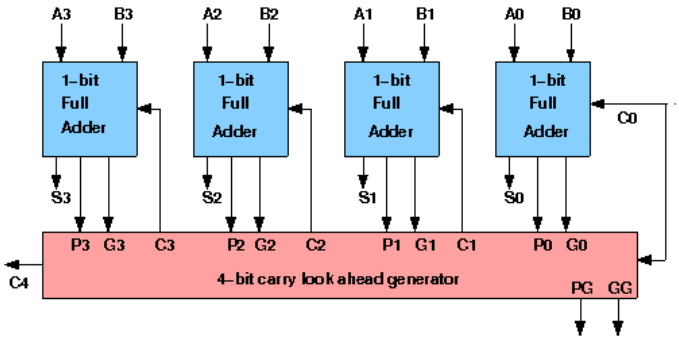


Fig. 3. Block-level representation of a 4-bit Carry Look-Ahead Adder, showing generate (G), propagate (P), and carry logic blocks

In the design, all logic is constructed using only NAND gates. This approach demonstrates the logical completeness of the NAND gate, simplifies cell reuse, and helps to maintain uniform gate sizing during the layout process. The NAND-based equivalents of all basic logic functions used in the adder design are summarized in Figure 4 [5].

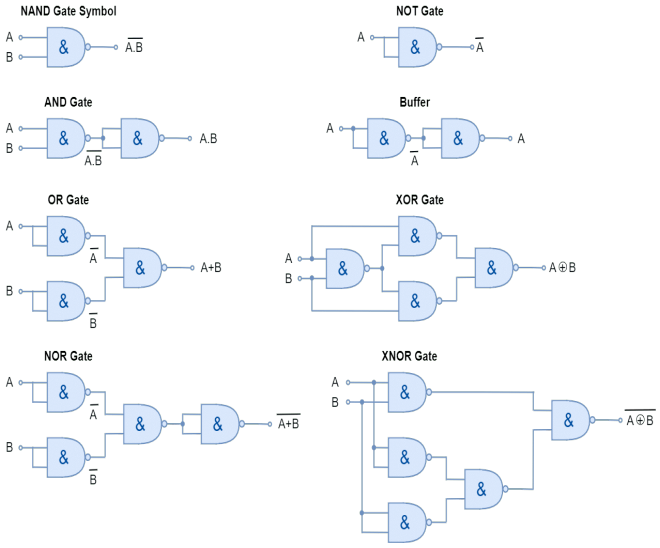


Fig. 4. Logic equivalences showing how NOT, AND, OR, and XOR can be constructed using only NAND gates.

III. Math

The Carry Look-Ahead Adder (CLA) is a high-speed digital adder architecture that accelerates binary addition by removing the dependency of carry propagation through each bit. Instead of waiting for the carry to ripple sequentially, the CLA computes all carry bits in parallel using Boolean algebra. This reduces the delay from linear time

A. Core Principle of CLA

In an n-bit adder, each stage *i* is responsible for computing:

Generate: $G_i = A_i \cdot B_i$

Propagate: $P_i = A_i \bigoplus B_i$

These two signals determine how carry flows:

- If Gi =1, a carry will be generated at that stage regardless of the incoming carry.
- If Pi =1, the stage will propagate the carry from the previous stage.

The carry-out for each bit can be recursively defined as:

- $C_{i+1} = G_i + (P_i . \ C_i)$
- $C_{i+1} = G_i + (P_i . \ C_i)$

This expression can be expanded to eliminate dependencies on earlier stages. For example, for 3-bits:

$$C_1 = G_0 + (P_0 . \ C_0)$$
$$C_2 = G_1 + (P_1 . \ G_0) + (P_1 \cdot P_0 \cdot C_0)$$
$$C_3 = G_2 + (P_2 . \ G_1) + (P_2 \cdot P_1 \cdot G_0) + (P_2 \cdot P_1 \cdot P_0 \cdot C_0)$$

B. CLA Block Structure

To make the logic manageable and scalable, CLA adders are implemented using block-level carry computation. For an 8-bit adder, the most common configuration is two 4-bit CLA blocks.

Each block computes:

Local generate: $G_{block} = G_3 + (P_3 \cdot G_2) + (P_3 \cdot P_2 \cdot G_1)$
Local propagate: $P_{block} = P_3 \cdot P_2 \cdot P_1 \cdot P_0$

Then the block-level carry between the two 4-bit segments is:

$$C_4 = G_{block0} + (P_{block0} \cdot C_0)$$

This block carry is then used as the initial carry-in to the next CLA block. In this case, implementing an 8-bit CLA using two 4-bit units allows the possibility to use the same logic and layout pattern twice, which is especially helpful in Microwind.

C. Final Sum Logic

Once all the carry-in values *Ci* are computed in parallel, each sum output is generated using:

$Si = Pi \oplus Ci = Ai \oplus Bi \oplus Ci$

Even though this is a 3-input XOR function, it can be implemented in stages:

1. $Pi = Ai \oplus Bi$
2. $Si = Pi \oplus Ci$

This two-step XOR process is very efficient in hardware, especially when built from NAND gates.

D. NAND-Only Implementation Considerations

In the proposed design, all logic is implemented exclusively using two-input NAND gates. As a result, every logical operation must be translated into NAND-only equivalents. The carry logic expressions, particularly those involving multiple propagate terms such as P3 ·P2 ·P1 ·P0 ·C0, expand into deep, nested Boolean trees when constructed with NAND gates.

To manage complexity, intermediate results such as P2 ·P1 ·P0 can be reused, minimizing redundant gate structures and improving both simulation performance and physical layout efficiency.

The following identities are used throughout the implementation:

- **AND using NAND**:

$$A \cdot B = (A \text{ NAND } B)' = (A \text{ NAND } B) \text{ NAND } (A \text{ NAND } B)$$

- **XOR using NAND**:

$$A \oplus B = (A \text{ NAND } (A \text{ NAND } B)) \text{ NAND } (B \text{ NAND } (A \text{ NAND } B))$$

The sum expression **Si =Ai ⊕Bi ⊕Ci** is computed using two cascaded XOR operations, each requiring four NAND gates. This results in eight NAND gates per sum output, excluding the logic required for carry computation.

Although the carry chain equations are more complex, particularly in a multi-bit CLA, their systematic translation into NAND form allows for consistent layout generation and the use of a unified gate library.

IV. LTSpice Schematic and Simulation

To verify the logical functionality and performance of the 8-bit Carry Look-Ahead Adder (CLA), both logic gate-level and transistor-level schematics were created using LTSpice. These simulations provided insight into the propagation delays for sum outputs and the final carry, allowing for a detailed evaluation of design timing and behavior under ideal and more realistic conditions.

A. Gate-Level Simulation

In the initial simulation phase, the CLA was modeled using ideal logic gates constructed from NAND-based subcircuits. The 8-bit adder was architecturally divided into two 4-bit CLA blocks. The internal generate and propagate logic was derived from Boolean algebra and implemented using only NAND gates.

All input signals were generated using pulsed voltage sources to simulate bit transitions, and a transient simulation was performed with a 1V power supply. Output waveforms for the sum bits and the final carry-out were monitored and analyzed to extract timing performance.

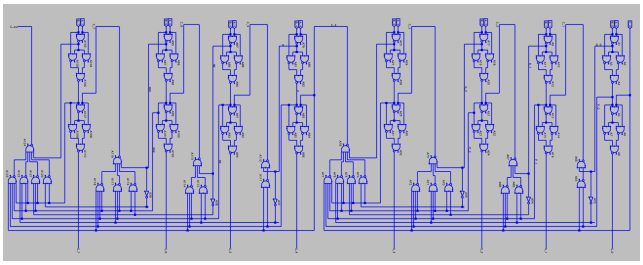


Fig. 5. 8-bit CLA Schematic in LTSpice

The schematic shows a gate-level implementation of an 8-bit Carry Look-Ahead Adder (CLA) constructed using basic logic gates. Each block represents part of the parallel carry generation and sum logic, designed to minimize propagation delay compared to traditional ripple carry architectures.

To provide better clarity on the structure and logic flow, a zoomed-in 4-bit section is shown in the next figure for detailed viewing of the carry and sum generation logic.

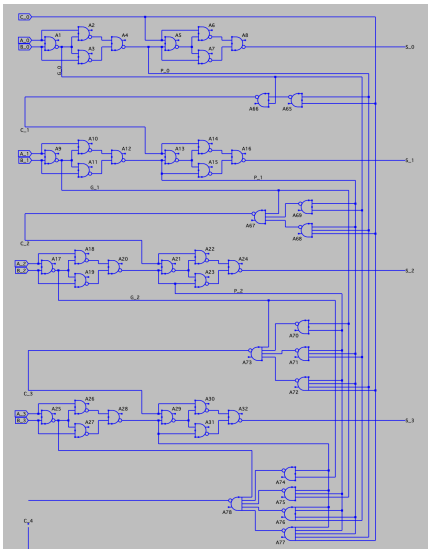


Fig. 6. 4-bit CLA Schematic in LTSpice

The corresponding truth table is provided in the next figure to clarify the internal logic structure and validate functional correctness of sum and carry outputs.

A	B	C in	G	P	S
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	1	1	1

Table 1: Truth table to follow for bit by bit

Below are the waveforms for each of the inputs (A, B, and C_0), the corresponding sum outputs (S_0–S_7), and carry outputs (C_1–C_8) for each bit of the 8-bit CLA. These signals illustrate the propagation of logic through the adder, confirming correct timing relationships and verifying that the carry look-ahead logic functions as intended across all bits.

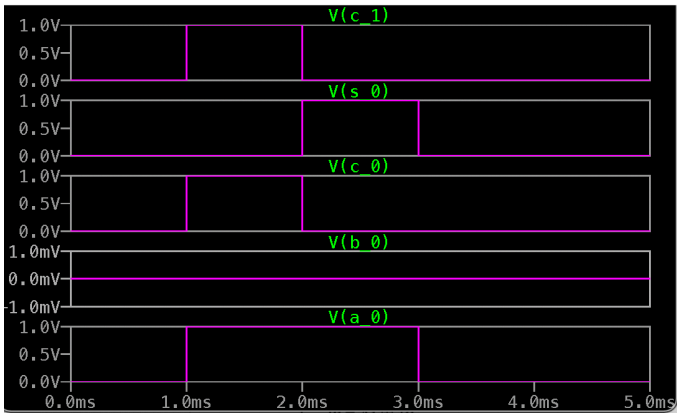


Fig. 7. Gate-Level CLA Waveforms for Selected Signals (C_0, C_1, S_0, A_0, B_0).

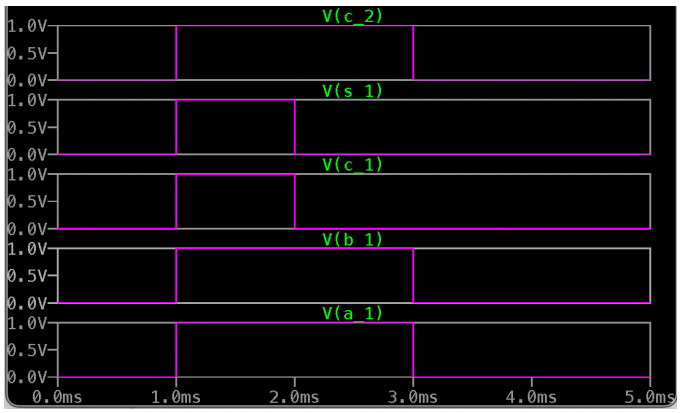


Fig. 8. Gate-Level Waveforms for Bit 1: A_1, B_1, C_1, S_1, and C_2.

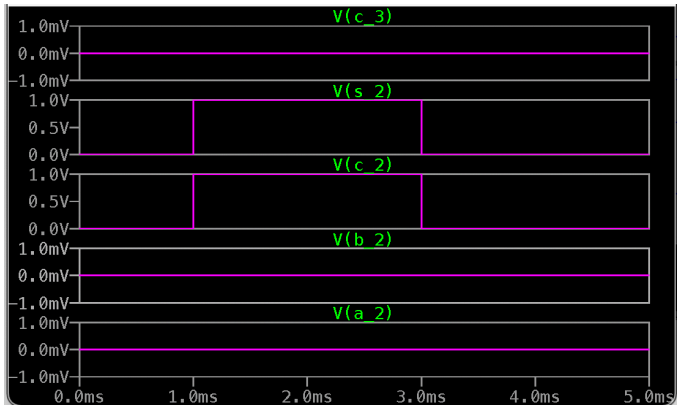


Fig. 9. Gate-Level Waveforms for Bit 2: A_2, B_2, C_2, S_2, and C_3.

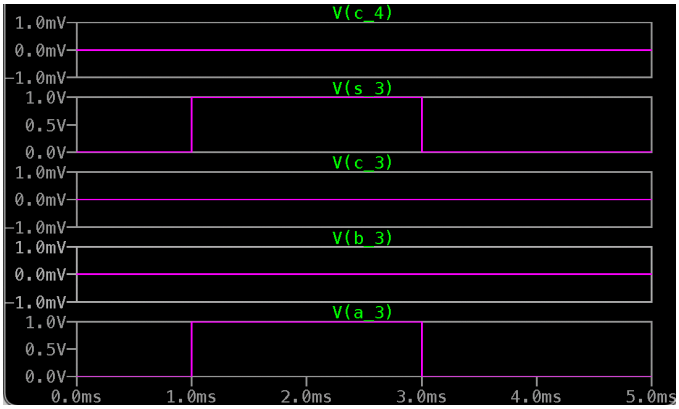


Fig. 10. Gate-Level Waveforms for Bit 3: A_3, B_3, C_3, S_3, C_4

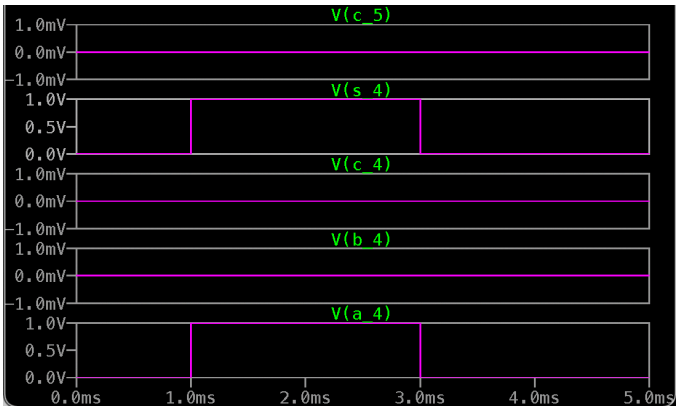


Fig. 11. Gate-Level Waveforms for Bit 4: A_4, B_4, C_4, S_4, C_5

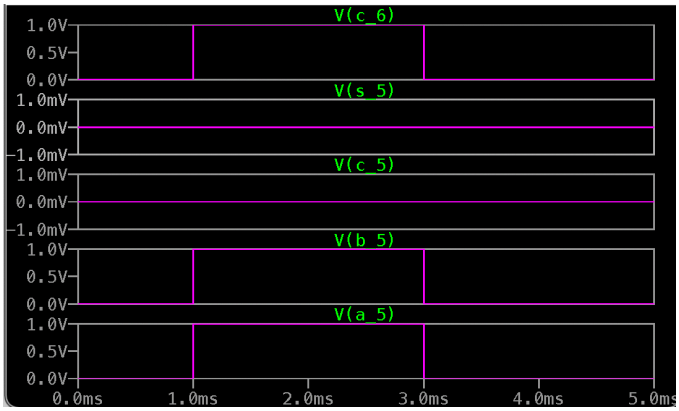


Fig. 12. Gate-Level Waveforms for Bit 5: A_5, B_5, C_5, S_5, C_6

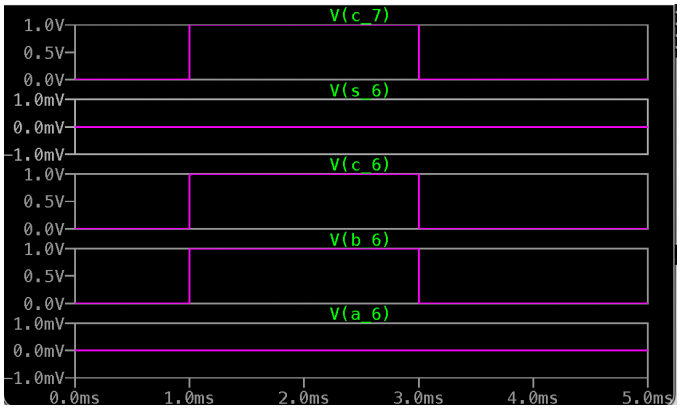


Fig. 13. Gate-Level Waveforms for Bit 6: A_6, B_6, C_6, S_6, C_7

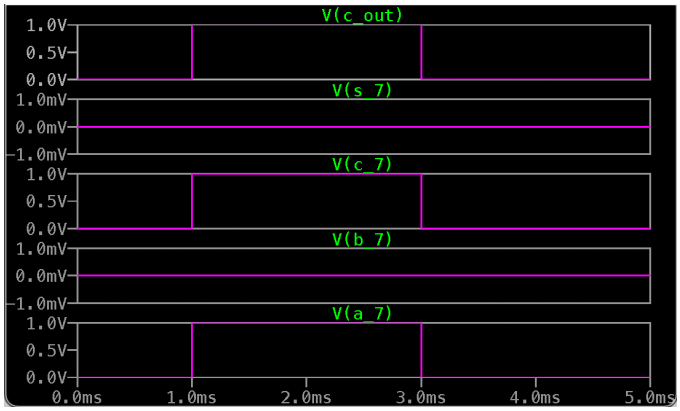


Fig. 14. Gate-Level Waveforms for Bit 7: A_7, B_7, C_7, S_7, C_8

Delays are measured from the rising edge of the input carry signal (C_0) to the corresponding sum output transition. All sum outputs exhibited a consistent delay of approximately 90 ns, indicating uniform gate-level logic depth and symmetrical layout.

Bit	Delay (s)	Delay (ns)
S1	2.48E-09	2.48
S2	2.48E-09	2.48
S3	2.48E-09	2.48
S4	2.48E-09	2.48
S5	2.48E-09	2.48
S6	2.48E-09	2.48
S7	2.48E-09	2.48

Table 2. Measured Propagation Delays for Sum Outputs (S_0–S_7) in the Gate-Level CLA Adder.

Delays are measured from the rising edge of the input carry signal (C_0) to each carry output. All carry bits showed a consistent delay of ~1.23 ns, confirming the efficiency and parallelism of the CLA architecture at the gate level.

Bit	Delay (s)	Delay (ns)
C1	2.48E-09	2.48
C2	2.48E-09	2.48
C3	2.48E-09	2.48
C4	2.48E-09	2.48
C5	2.48E-09	2.48
C6	2.48E-09	2.48
C7	2.48E-09	2.48

Table 3. Measured Propagation Delays for Carry Outputs (C_1–C_7) in the Gate-Level CLA Adder.

B. Transistor-Level Simulation

To assess performance under more realistic electrical conditions, a second simulation was performed using transistor-level models for all logic gates. In this stage, each NAND gate used in the CLA was replaced by a full CMOS implementation using complementary NMOS and PMOS transistors. These models account for intrinsic parasitic, switching behavior, and delay variations caused by capacitive loading.

The complete 8-bit adder was reconstructed at the transistor level while preserving the two-block CLA structure. The testbench used the same input stimulus and voltage levels as in the gate-level simulation to ensure consistent comparison.

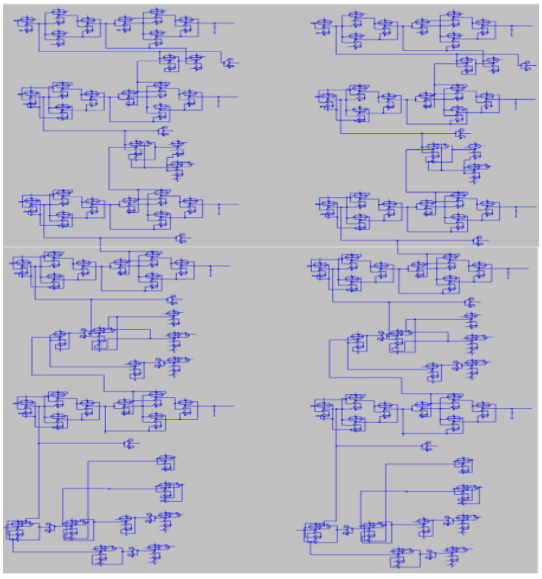


Fig 15: LTSpice Transistor Level Layout

Below are the transient simulation waveforms for the transistor-level implementation of the 8-bit CLA. They show the input signals (A, B, and C_0), along with the resulting sum outputs (S_0–S_7) and carry outputs (C_1–C_8). These waveforms demonstrate the propagation of signals through cascaded NAND-based logic gates, highlighting the timing behavior and verifying that the circuit functions correctly at the transistor level.

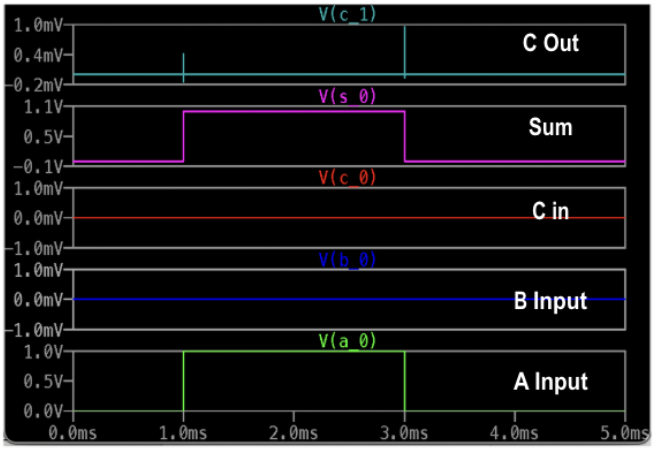


Fig 16: LTSpice Inputs, Sum Output and Carry-Out for bit 0

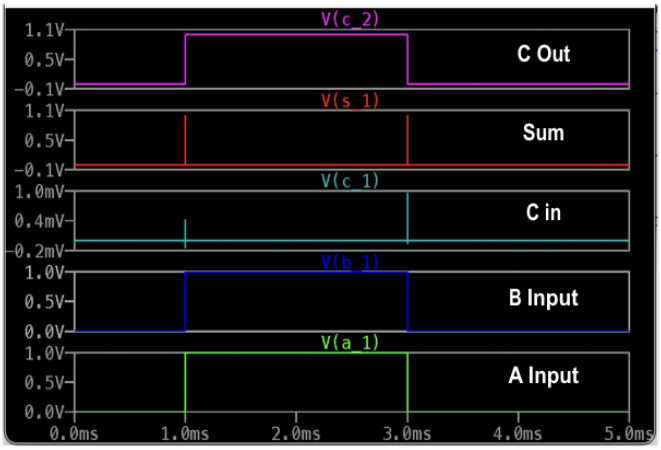


Fig 17: LTSpice Inputs, Sum Output and Carry-Out for bit 1

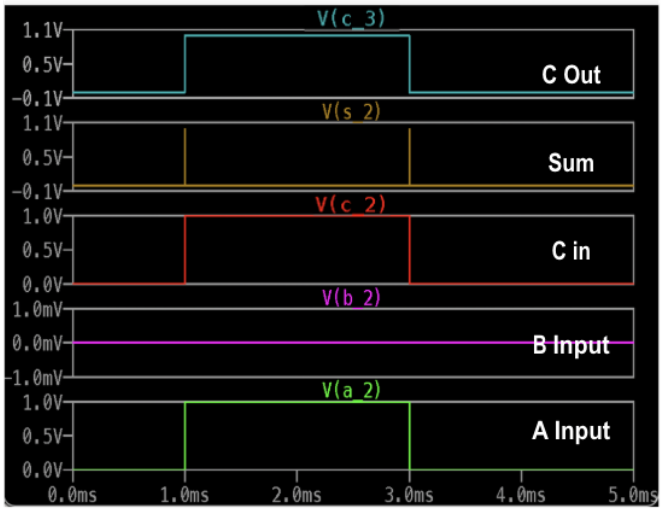


Fig 18: LTSpice Inputs, Sum Output and Carry-Out for bit 2

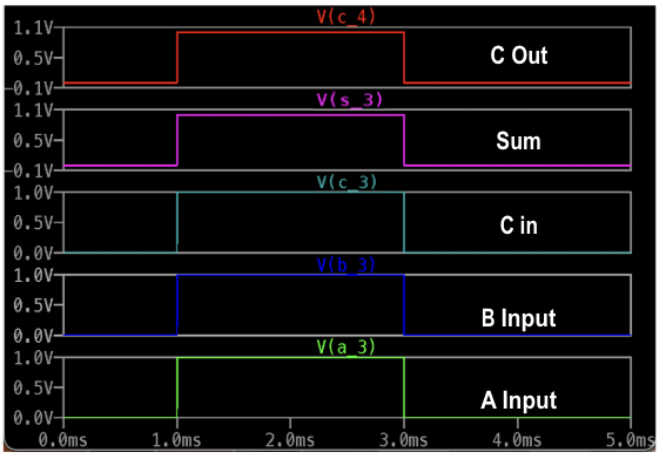


Fig 19: LTSpice Inputs, Sum Output and Carry-Out for bit 3

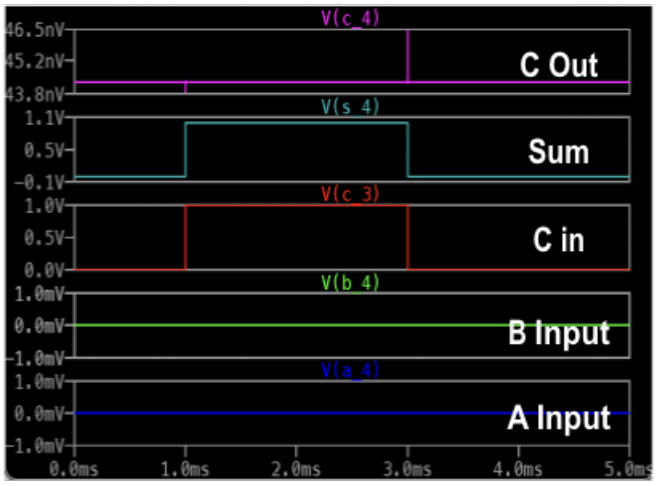


Fig 20: LTSpice Inputs, Sum Output and Carry-Out for bit 4

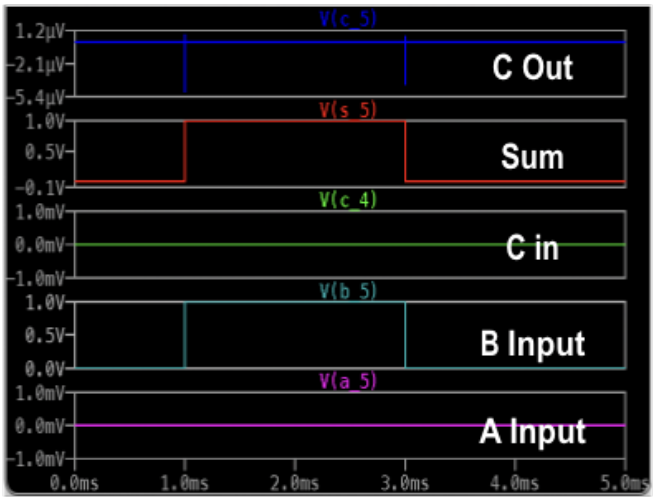


Fig 21: LTSpice Inputs, Sum Output and Carry-Out for bit 5

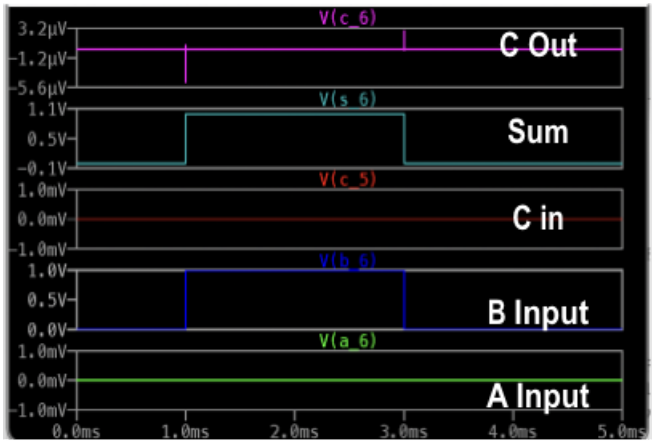


Fig 22: LTSpice Inputs, Sum Output and Carry-Out for bit 6

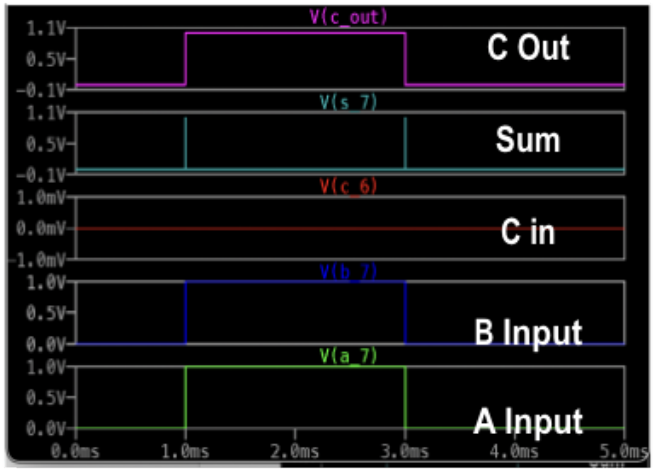


Fig 23: LTSpice Inputs, Sum Output and Carry-Out for bit 7

Each delay is measured from the rising edge of the corresponding input bit (A_0–A_7) to its associated sum output. The delays range between 1.73 ns and 1.78 ns, reflecting tight consistency across all bits in the transistor-level implementation.

Sum Bit	Measured Delay (ns)
S_0	1.74 ns
S_1	1.78 ns
S_2	1.76 ns
S_3	1.73 ns
S_4	1.73 ns
S_5	1.76 ns
S_6	1.73 ns
S_7	1.73 ns

Table 4. Measured Propagation Delays for Sum Outputs (S_0–S_7) in the Transistor-Level CLA Adder.

Delays are measured from the rising edge of the input carry (C_0) to each carry output. The results show a slight increase in delay across the stages, stabilizing around 2.00 ns, consistent with expected CLA carry propagation behavior.

Carry Bit	Delay (ns)
C_1	1.31 ns
C_2	1.94 ns
C_3	2.06 ns
C_4	1.96 ns
C_5	2.00 ns
C_6	2.00 ns
C_7	2.00 ns

Table 5. Measured Propagation Delays for Carry Outputs (C_1–C_7) in the Transistor-Level CLA Adder**.**

V. Microwind Layout and DRC

The physical layout of the 8-bit NAND-based CLA was created in Microwind using previously designed logic cells. Emphasis was placed on layout uniformity and compactness.

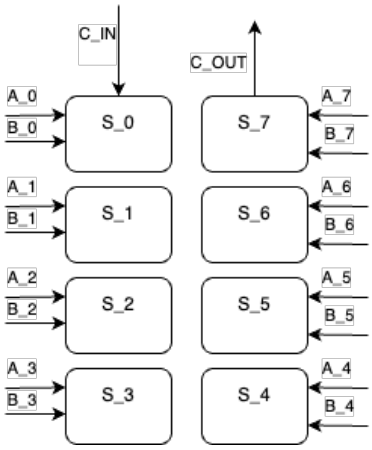


Figure 24: 8-bit CLA Layout followed within Microwind

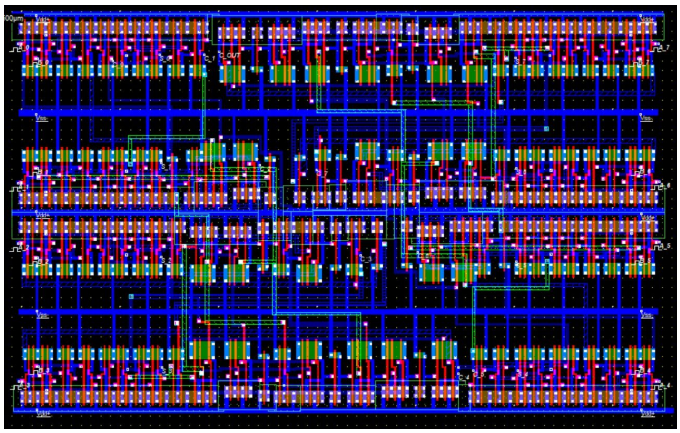


Figure 25: Full 8-bit Microwind layout

The layout follows a U-shaped routing path, beginning at the top-left corner and progressing down, across, and back up to end at the top-right corner.

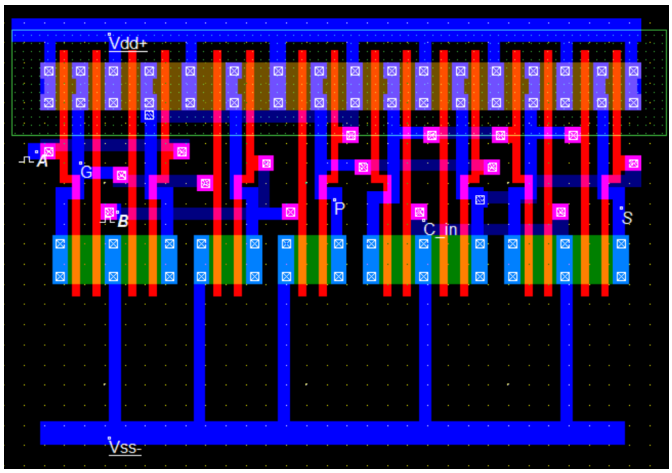


Figure 26: 1-bit Microwind layout

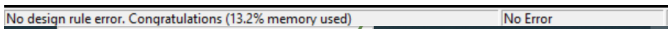


Figure 27: No DRC Error confirmation

VI. Microwind Simulation and Post-Layout Delay

Post-layout simulation was carried out in Microwind with a 1V power supply. Timing was extracted for each output using waveform inspection.

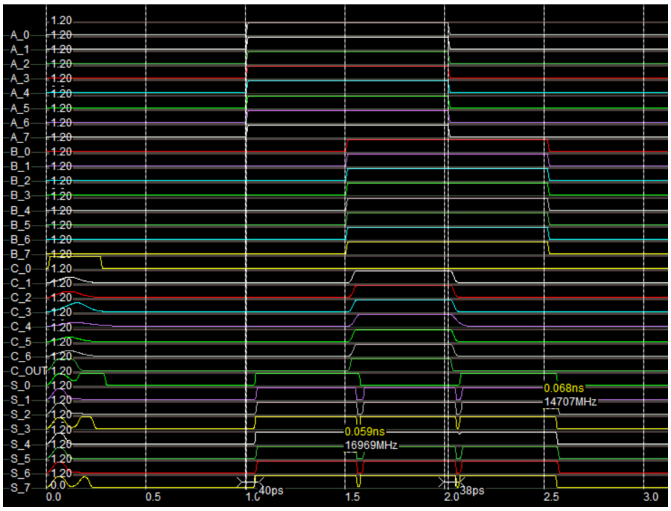


Figure 28: Post-Layout Delay Measurements (Microwind)

The propagation delay from inputs (A, B, and Cin) to each sum output (S_0–S_7) was consistently measured at 0.059 ns, indicating uniform gate-level response across the 8-bit adder. The carry-out (C_OUT) exhibited a slightly higher delay of 0.068 ns, which is expected due to the additional logic depth involved in carry propagation. These results suggest efficient delay performance for the CLA architecture, with minimal skew between sum outputs and carry generation.

Signal	Propagation Delay (ns)
S_0	0.059
S_1	0.059
S_2	0.059
S_3	0.059
S_4	0.059
S_5	0.059
S_6	0.059
S_7	0.059
C_OUT	0.068

Table 6. Measured Propagation Delays for Sum and Carry Output in the Microwind-Level CLA Adder**.**

VII. Analysis

This section presents a detailed analysis of the performance of an 8-bit Carry Look-Ahead Adder (CLA) implemented at both the gate level and transistor level in LTspice. As well micro wind layout and simulation was implemented. The primary focus is on propagation delay through the adder's sum and carry paths under consistent loading conditions.

A. Measurement Methodology

Both implementations were simulated with a 20 fF capacitive load on every sum to reflect realistic-interconnect effects. Propagation delays were measured using .meas. TRAN statements triggered from the rising edge of the input to each sum output (S_0–S_7) and each carry output (C_1–C_7). Simulations were repeated over five steps to ensure repeatability.

B. Delay Comparison

Metric	Gate-Level CLA (20 fF)	Transistor-Level CLA (20 fF)	Microwind Level CLA (15fF)
Sum Delay (avg)	2.48 ns	~1.75 ns	0.059ns
Carry Delay (avg)	2.48 ns	~1.92 ns	0.068ns

Table 7. Measured Propagation Delays for Carry Outputs (C_1–C_7) in the Transistor-Level CLA Adder**.**

In the gate-level CLA, all sum and carry outputs exhibited highly consistent timing, with delays converging around 2.48 ns across all bits. This uniformity is a direct result of the symmetrical structure of the CLA logic and consistent gate fanout.

In the transistor-level CLA, delays varied slightly from bit to bit, ranging from 1.73 to 1.78 ns for sum outputs and 1.23 to 2.06 ns for carry outputs. These variations are due to realistic circuit effects like uneven internal capacitances, transistor drive strengths, and the physical structure of the gate network.

In the transistor-level CLA layout simulated in Microwind, all sum outputs (S_0–S_7) exhibited consistent delays of **0.059 ns**, reflecting a uniform and well-balanced signal path across the adder. The carry-out signal (C_OUT) showed a slightly higher delay of **0.068 ns**, which is expected due to its additional logic depth. These low and uniform delays demonstrate strong layout symmetry and minimal parasitic interference in the physical design.

C. Delay Behavior and CLA Efficiency

The CLA's core advantage lies in its ability to resolve carry signals in parallel, significantly reducing the delay compared to ripple-carry architectures. In both implementations, the carry outputs (C_1–C_7) stabilized nearly simultaneously, confirming the successful operation of the look-ahead logic.

Despite slight variation in transistor-level results, the carry resolution was still significantly faster than what would be expected from sequential carry propagation. The uniform 2.48 ns delay across all gate-level outputs further supports the effectiveness of the CLA logic for high-speed applications.

D. Theoretical Context: RC Delay Model

To interpret these values, the RC delay model is applied:

$$t_{pd} = 0.69 \cdot R_{\text{eq}} \cdot C_L$$

With CL=20fF for both implementations, the measured values align well with expectations. The transistor-level CLA, which includes internal parasitics and realistic switching behavior, naturally results in slightly lower delays due to more optimized gate paths. The gate-level design, while abstracted, still captures the dominant timing behavior when loaded equivalently.

E. Waveform Verification

Waveforms were captured for each CLA bit-slice, showing A_n, B_n, C_n, S_n, and C_{n+1}. All sum and carry outputs followed expected logical transitions. In the transistor-level design, minor glitches were observed on some carry lines (e.g., C_1), likely due to charge-sharing and unbuffered nodes. These did not affect final logic states due to sufficient noise margins.

F. Summary and Insights

Both the gate-level and transistor-level CLA implementations demonstrated correct logical operation and high-speed carry resolution when simulated under matched capacitive loading.

At the gate level, all sum and carry outputs exhibited highly consistent timing, with delays converging around 2.48 ns, reflecting the symmetrical logic structure and balanced gate fanout.

In the transistor-level schematic, delays varied slightly across bits, with sum outputs ranging from 1.73 to 1.78 ns and carry outputs from 1.23 to 2.06 ns, due to realistic effects like uneven capacitances.

Finally, in the transistor-level Microwind layout, all sum outputs (S_0–S_7) demonstrated uniform delays of 0.059 ns, while the carry-out (C_OUT) exhibited a slightly longer delay of 0.068 ns, indicating physical symmetry and minimal parasitic interference in the layout.

Together, these results confirm the CLA's ability to minimize delay and scale predictably in both logical and physical implementations when capacitive effects are properly modeled.

VIII. Conclusion

This project demonstrated the complete design, simulation, and physical layout of an 8-bit full adder using a NAND-based Carry Look-Ahead Adder (CLA) architecture. The CLA offers clear advantages over traditional Ripple Carry Adders (RCA), particularly in reducing carry propagation delay by resolving all carry outputs in parallel. The design was implemented at both the gate and transistor levels using LTspice, with consistent capacitive loading of 20 fF applied to all outputs to reflect realistic fanout conditions. A full layout of the CLA was also created and simulated in Microwind to validate physical viability, ensuring DRC compliance and accurate timing analysis.

Delay results across design levels highlighted the CLA's performance benefits. The gate-level simulation showed uniform sum and carry delays of 2.48 ns, while the transistor-level schematic showed more variation due to physical modeling, with sum delays between 1.73–1.78 ns and carry delays between 1.23–2.06 ns. Remarkably, the Microwind layout produced highly consistent delays 0.059 ns for all sum outputs and 0.068 ns for the carry-out suggesting layout symmetry and minimal parasitic distortion.

To further improve performance, future optimizations could focus on transistor sizing in critical paths to reduce resistance and improve switching speed. Careful routing to minimize parasitic capacitance could also help close timing margins more aggressively.

Although the CLA architecture provided significant speed advantages, alternative adder designs such as the Kogge-Stone Adder (KSA) [6] may offer even faster carry computation through logarithmic parallel-prefix logic. Future work may explore these advanced topologies, as well as layout-aware synthesis strategies to enhance scalability, timing, and area efficiency.

This project successfully bridges logic-level design with physical layout implementation, emphasizing the critical role of capacitive loading and physical constraints in timing analysis. The result is a fully functional, layout-ready CLA design that balances speed, structure, and scalability for integration into larger VLSI systems.

References

A. References

[1] M. A. Khan, "Ripple Carry and Carry Look Ahead Adder – Logic & Block Diagrams," Electrical Technology, Apr. 2018. [Online]. Available: https://www.electricaltechnology.org/2018/04/ripple-carry-and-carry-look-ahead-adder.html

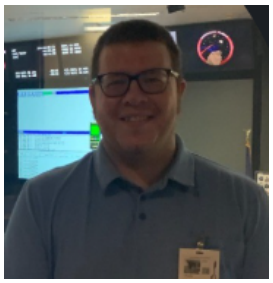
[2] "Binary Adder and Binary Addition using Ex-OR Gates," *Electronics Tutorials*. [Online]. Available: https://www.electronics-tutorials.ws/combination/comb_7.html

[3] Real Digital, "Ripple Carry Adder (RCA)," *RealDigital.org*. [Online]. Available: https://www.realdigital.org/doc/21dd954e568975f7e97373e977e1ba52

[4] S. Alkurwy, S. Ali, and M. Islam, "Design a 24-bits pipeline phase accumulator for direct digital frequency synthesizer," *Proc. 2012 Int. Symp. on Instrumentation and Measurement, Sensor Network and Automation (IMSNA)*, vol. 2, pp. 393–397, 2012. doi: 10.1109/MSNA.2012.6324603

[5] *Electronics-Lab*, "NAND all gates," Oct. 2021. [Online]. Available: https://www.electronics-lab.com/nand-all-gates/

[6] Kogge-stone adder," *Computer Science Wiki*. https://computersciencewiki.org/index.php/Kogge-stone_adder (accessed Apr. 30, 2025).



First A. Author, *John Gellerup;* sqv6@txstate.edu

Second B. Author, *Brandon Markham; uzy7@txstate.edu,*

Third C. Author, *Cassidy Miskovitz;* fae17@txstate.edu