Optimized Design of 32-Bit Comparator

Mohammad Milhem
Faculty of Engineering & Technology
Birzeit University
Ramallah, Palestine
mohamadmilhem100@gmail.com

Mumen Anbar
Faculty of Engineering & Technology
Birzeit University
Ramallah, Palestine
mumenanbar12@gmail.com

Ameer Rabie
Faculty of Engineering & Technology
Birzeit University
Ramallah, Palestine
ameerrabie2008@gmail.com

***Abstract*—This paper presents a 32-bit comparator architecture designed to optimize the comparison of binary numbers, using the logic of a prefix adder (CLA). Employing a hierarchical tree structure, the proposed design enhances computation speed, achieving a $O(\log n)$ latency. This architecture effectively computes the final carry or borrow bit with remarkable efficiency, making it highly suitable for high-speed digital systems. The proposed comparator demonstrates substantial performance improvements, particularly in applications requiring minimal latency.**

***Index Terms*—CMOS, Comparator, CLA, delay, area, power**

I. Introduction

In digital systems, comparators are basic arithmetic elements that are crucial for figuring out the relationship between two binary integers, such as whether one is more or less than the other or if they are equal. Comparators are essential components of general-purpose and application-specific architectures as crucial datapath elements, especially in sorting and signal processing networks. In multiprocessing settings, multi-access memories, and parallel computing, the latter is essential.

Many comparator systems using both serial and parallel architectures have been suggested in recent years. Although serial comparators work well in small systems and with short input lengths, their lower operational speed and increased circuit complexity present serious problems when scaling to bigger inputs. As a result, parallel designs have emerged as the superior and substitute option for high-performance comparators that can process longer input lengths.

A 32-bit comparator circuit built on a tree architecture is shown in this paper. The approach compares two numbers effectively by using a modified prefix adder logic. by applying the divide-and-conquer strategy. The rest of this essay is structured as follows: The current architectures for comparators—including parallel and serial designs—are covered in Section II. The suggested comparator design is introduced in Section III, together with information on its benefits and methods of use. The simulation results are shown in Section IV, where the suggested design's benefits in performance and efficiency are highlighted by a comparison with the solutions now in use.

II. current Architectures

There are two main categories for existing comparator architectures: serial and parallel designs. Serial comparators sequentially process bits one at a time, beginning with the most important bit (MSB) and ending with the least important bit (LSB). This method is simpler but takes longer because it is done step by step. On the other hand, parallel comparators assess every bit at the same time, cutting down on comparison time but bolstering design complexity and hardware needs. Parallel designs are favored when speed is crucial, even though they are more complex to implement.

III. Proposed Architecture

The proposed comparator architecture is designed to efficiently compare two binary numbers by utilizing the principles of a prefix adder and divide-and-conquer concepts. This section provides a detailed explanation of the architecture, detailing its functionality, internal components, and the specific role of each functional block in the comparison process.

A. Functionality Overview

The proposed comparator is an enhanced version of a prefix adder, optimized for comparing two binary numbers, A and B. The primary function of this comparator is to determine whether $A > B$, $A = B$, or $A < B$. This is achieved by computing a final carry bit, which indicates the relative magnitude of the two numbers. The architecture employs a divide-and-conquer approach, where each sub-segment of bits is processed independently, and the results are progressively combined at subsequent levels of the tree structure. Unlike traditional adders, which require carry computation for every bit position, this comparator only necessitates the final carry, thereby reducing both complexity and latency.

B. Recursive Carry Computation

Following the generation of the g_i and p_i signals for each bit position, the proposed architecture advances into a tree structure designed to recursively combine these signals. This process is essential for computing the final carry or borrow bit, which determines the outcome of the comparison between the binary numbers A and B. The recursive carry computation is governed by the equation:

$$c_i = g_i + p_i c_{i-1}$$

Here, c_i denotes the carry bit at the ith position. The signal g_i represents the generate signal, indicating whether a carry is generated at that position, while p_i is the propagate signal,

indicating whether the carry from the previous position c_{i-1} should be propagated to the current position.

The final carry bit for the most significant position, c_{n-1}, can be expressed as:

$$c_{n-1} = g_{n-1} \vee p_{n-1}g_{n-2} \vee p_{n-1}p_{n-2}g_{n-3} \vee \cdots \vee p_{n-1}p_{n-2}\ldots p_1 g_0$$

However, as this equation illustrates, directly computing the carry bit through a linear process with $O(n)$ latency involves multiple sequential operations, leading to significant delays, especially for large bit-widths. To address this, a hierarchical approach is employed.

C. Hierarchical Comparison Process

By structuring the computation in a tree-like hierarchy, the architecture reduces the number of sequential operations required, thus improving both speed and efficiency, thereby achieving a latency of $O(\log(n))$. This hierarchical method allows for parallel processing at each level of the tree, significantly reducing the overall latency of the carry computation and making the architecture more suitable for high-speed applications.

The comparison process in the proposed architecture is hierarchical. At each level of the tree, the generate and propagate signals from the previous level are used to compute higher-order group generate and propagate signals through a block called PG block. This process continues until the final level, where the comparison result is obtained. The final signals are used to determine the relationship between A and B.

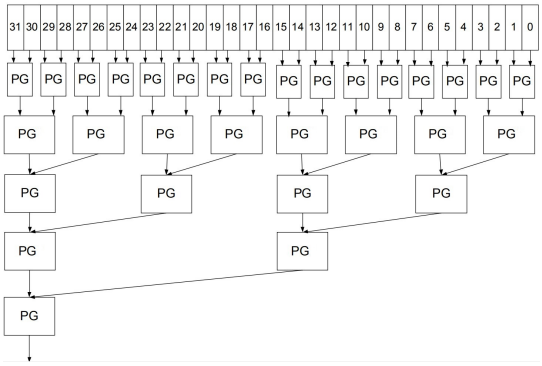

Fig. 1. Hierarchical Design of Proposed Comparator

D. Detailed Block Description

1) Prefix Adder Concept: The first stage and the core of the proposed comparator architecture is the prefix-adder block. In this architecture, each bit position i of the input numbers A and B generates two signals: the generate signal g_i and the propagate signal p_i.

- **Generate Signal** (g_i): This signal indicates whether the current bit position may generate a carry. It is defined as:

$$g_i = a_i \cdot \overline{b_i}$$

where a_i and b_i are the bits of the numbers A and B, respectively, and $\overline{b_i}$ is the complement of b_i.

- **Propagate Signal** (p_i): This signal indicates whether the current bit position may propagate a carry generated by a previous bit position. It is defined as:

$$p_i = a_i \oplus \overline{b_i}$$

where $\oplus$ denotes the XOR operation.

By initializing the g_i and p_i values for every bit pair, the prefix-adder block sets the foundation for the recursive carry computation in the subsequent stages of the tree. This initialization is crucial for ensuring that the correct values are propagated and combined as the architecture progresses towards computing the final carry bit, which ultimately determines the result of the comparison.

2) PG Block: The next stages of the comparator consists of Propagate and Generate (PG) blocks. Each PG block takes two sets of input (p_i, g_i) and (p_j, g_j) from the lower left and right sub-blocks respectively, and produces the corresponding generate g_i and propagate p_i signals. These signals are then passed to the next level of the tree structure, where they are recursively combined to compute higher-order generate and propagate signals.

The intermediate levels of the tree structure consist of additional PG blocks that take the outputs of the previous level and combine them to produce the next level's generate and propagate signals. This process is repeated until the final level, where the final signals are computed.

The final output of the comparator is determined by the propagated signals p_i which determines $A = B$. Otherwise, carry bit determines the result, if the final carry is 1, then $A > B$. Otherwise, $A < B$.

IV. Design Process and Results

The design process began with the construction of low-level blocks (i.e. NOT, NAND and NOR gates), or standard cells, which were then utilized in the final design. The layout for each block was manually constructed, and the entire process was implemented using 22nm technology.

A. Inverter

1) Schematic: The most used gate during this project, it was designed it using CMOS logic, having a ratio equals 2 between NMOS and PMOS widths with 8nm width of the NMOS.

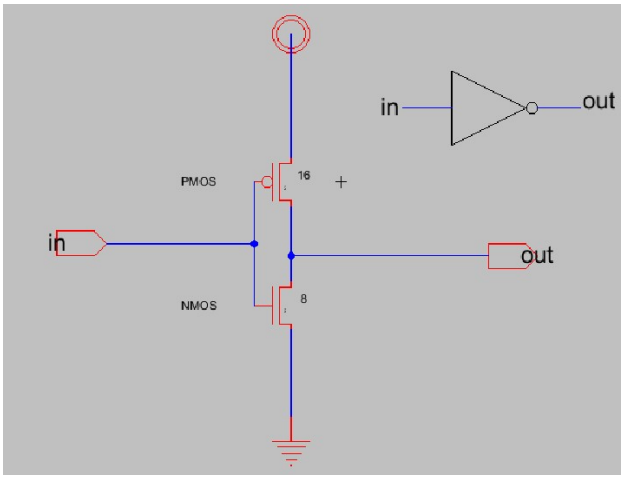

Fig. 2. Inverter Schematic Design

The simulation of the inverter shows the correct inversion of the input signal with the desired timing characteristics.

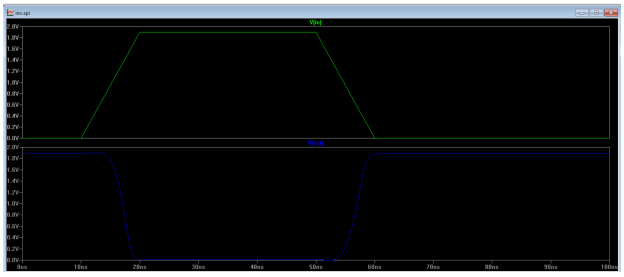

Fig. 3. Inverter Simulation Results

2) Layout: Inverter layout and simulation results.

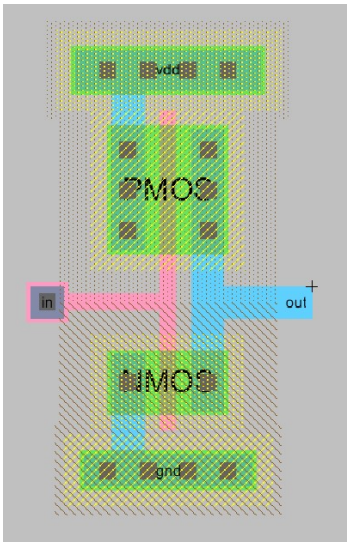

Fig. 4. Inverter Schematic Design

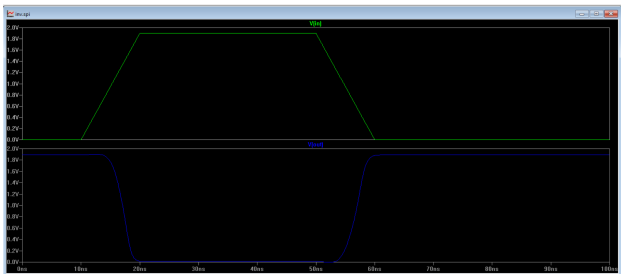

Fig. 5. Inverter Simulation Results

B. NAND2

1) Schematic: Designing a 2 input NAND gate using 4 CMOS devices with having small delay as much as possible.

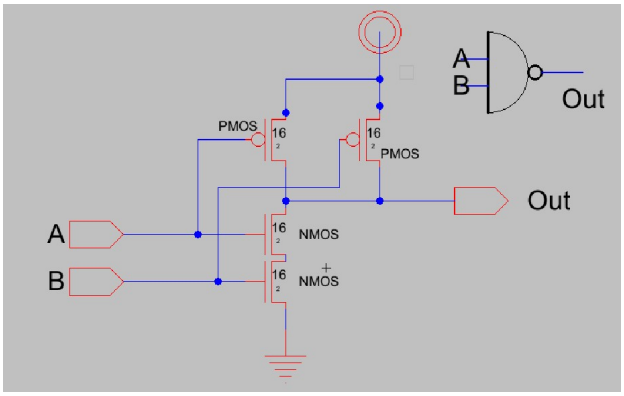

Fig. 6. NAND2 Schematic Design

Simulation results confirm the correct functioning of the NAND2 gate, ensuring it meets the required performance criteria.

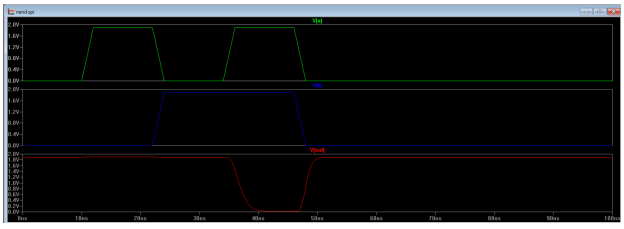

Fig. 7. NAND2 Simulation Results

2) Layout: Layout design about NAND2.

$$tr = (2.85919e{-}9), tf = (3.56197e{-}9), td = (1.498131e{-}9)$$

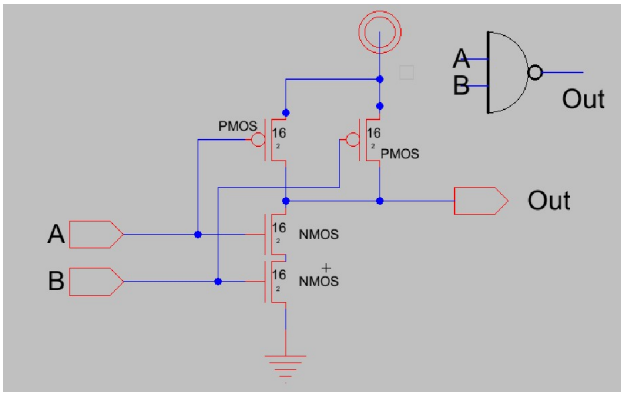

Fig. 8. NAND2 Schematic Design

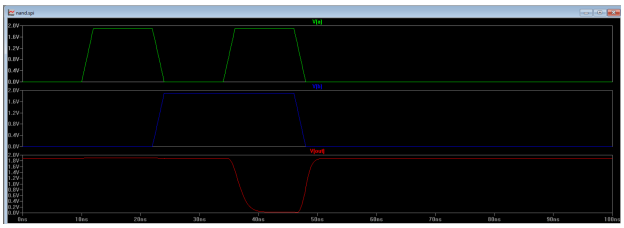

Fig. 9. NAND2 Simulation Results

C. XOR

1) Schematic: Schematic Design of XOR gate.

$tr = (2.85919e{-}9), tf = (3.56197e{-}9), td = (1.498131e{-}9)$

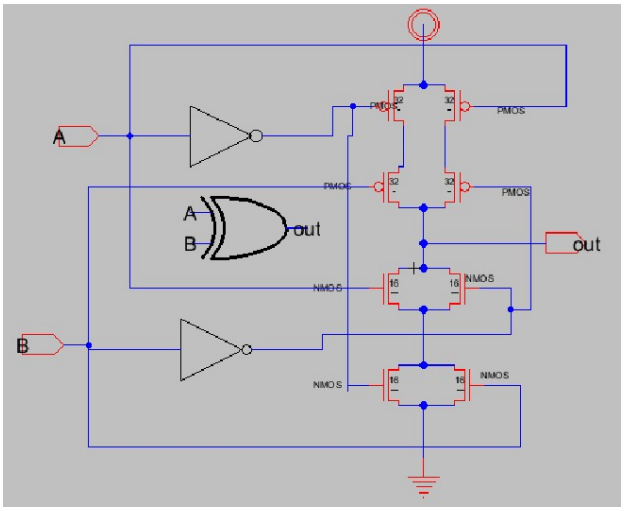

Fig. 10. XOR Schematic Design

The simulation verifies that the XOR gate performs correctly under all input conditions, adhering to the expected timing.

2) Layout: Xor layout design.

$tr = (3.119764e{-}9), tf = (3.56197e{-}9), td = (2.91761e{-}9)$

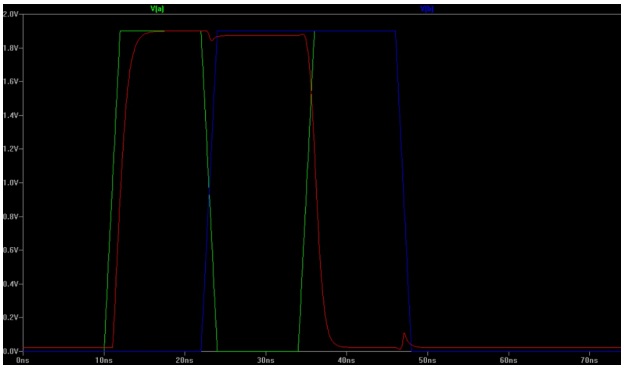

Fig. 11. XOR Simulation Results

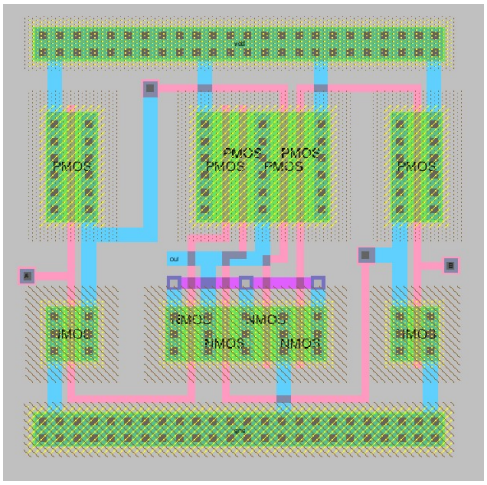

Fig. 12. XOR Layout Design

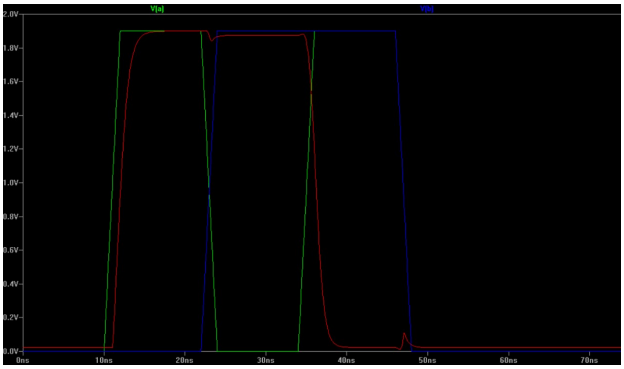

Fig. 13. XOR Simulation Results

D. Prefix-Adder

1) Schematic: The Prefix-Adder forms the core of the proposed comparator architecture. Its design ensures efficient computation of the two signals p_i and g_i.

Simulation results demonstrate the Prefix-Adder's ability to compute signals in a small amount of delay.

$tr = (3.119764e{-}9), tf = (3.56197e{-}9), td = (2.91761e{-}9)$

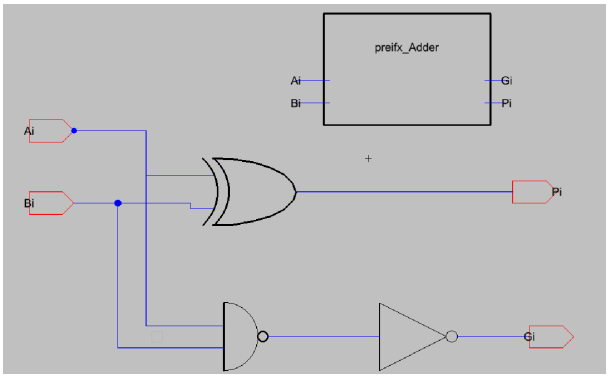

Fig. 14. Prefix-Adder Schematic Design

2) Layout: Layout of prefix-adder.

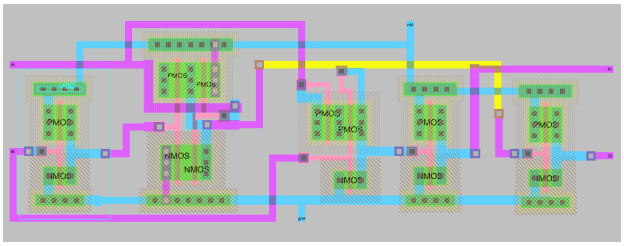

Fig. 15. Prefix-Adder Layout Design

E. Generate

1) Schematic: The Generate block is crucial in the prefix-adder structure, responsible for determining whether a carry should be generated at each bit position.

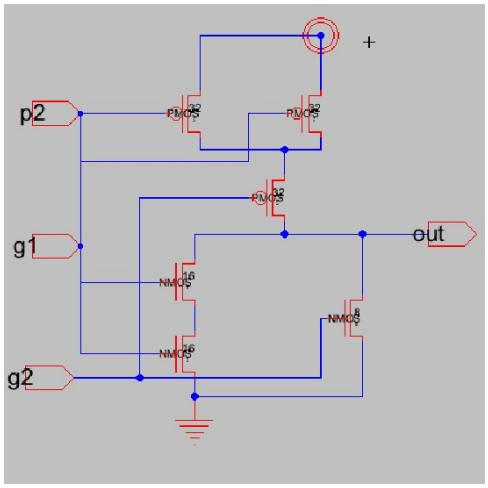

Fig. 16. Generate Block Schematic Design

2) Layout: Layout of generate block.

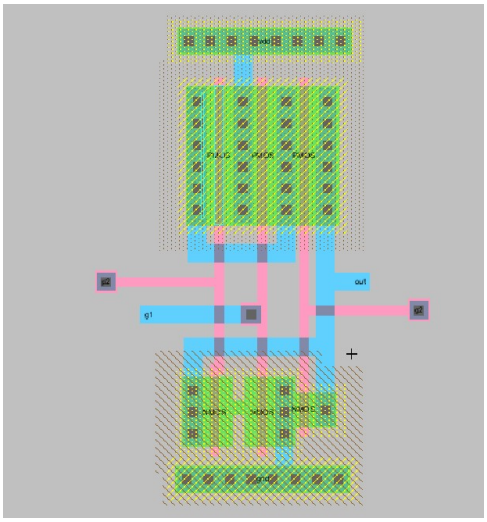

Fig. 17. Generate Block Layout Design

F. PG

1) Schematic: The Propagate and Generate (PG) block is a critical component that works in tandem with other blocks to propagate and generate signals across the comparator's tree structure.

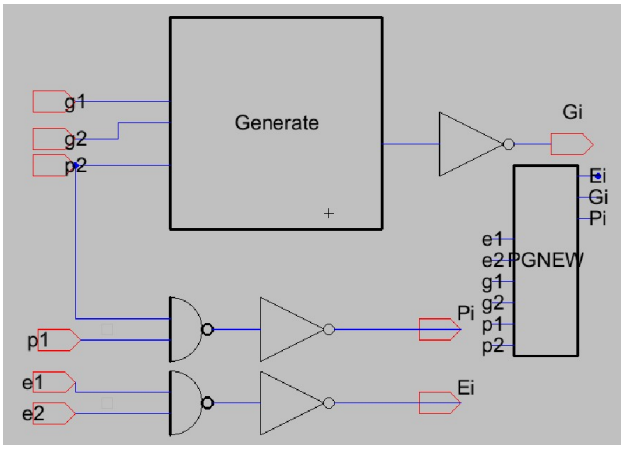

Fig. 18. PG Block Schematic Design

2) Layout: Layout of PG block.

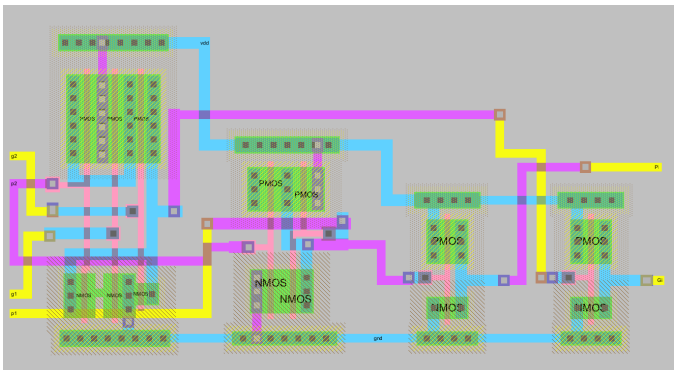

Fig. 19. PG Block layout Design

V. Main Component Comparator

A. Design

1) Schematic: The main component comparator schematic design where each of the previous blocks were gathered to form the final design of it.

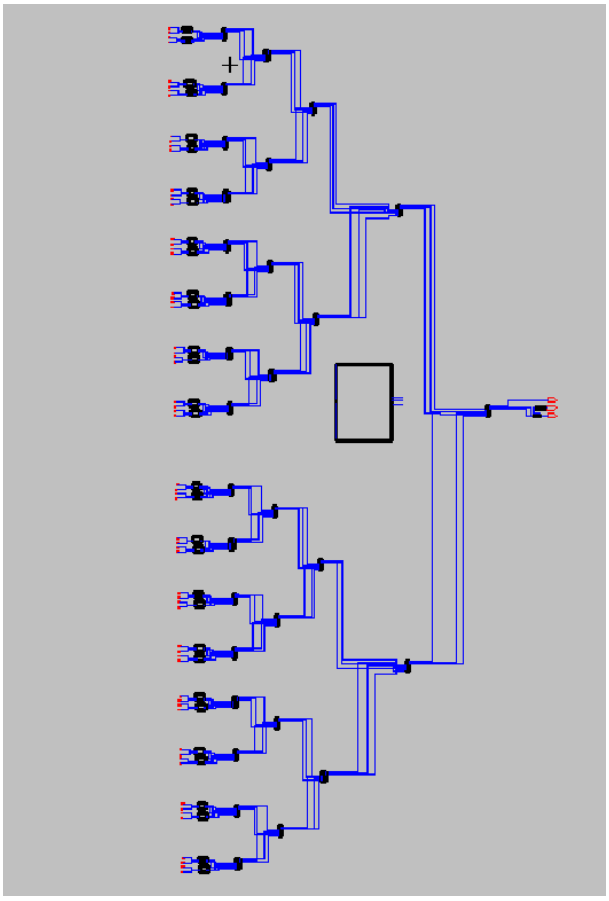

Fig. 20. Main Component Schematic Design

2) Layout: Layout of PG block.

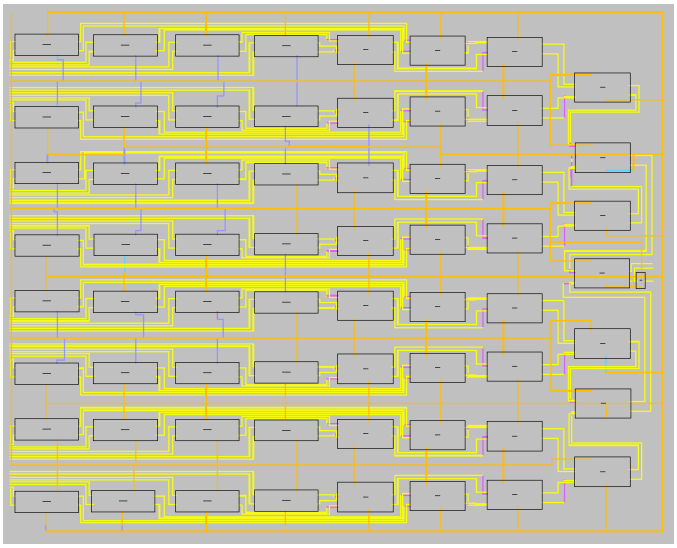

Fig. 21. Main Component Layout Design

B. Results and Discussion

Several time and power analysis were done on this case with varying input and power supply values.

1) Time Analysis: Time analysis was done to figure out the values of rise time, fall time and delay time of the proposed component.

1) Rise Time

TABLE I

Vdd (V)	LT	Eq	GT
0.95	2.0682	1.8649	2.0158
1.9	2.4374	2.0952	2.7825

2) Fall Time

TABLE II

Vdd (V)	LT (ns)	Eq (ns)	GT (ns)
0.95	2.9223	2.7374	2.7712
1.9	2.5914	2.1954	2.1724

3) Propagation Delay

TABLE III

Vdd (V)	LT (ns)	Eq (ns)	GT (ns)
0.95	2.2682	1.8649	2.0158
1.9	1.9877	1.5684	1.4356

From previous studies, it can be clearly shown that changing the supply voltage V_{DD} in a circuit presents a trade-off between delay and power consumption. When V_{DD} is increased, the circuit operates faster, reducing the delay since transistors switch more quickly; however, this comes at the cost of higher power consumption and vice versa.

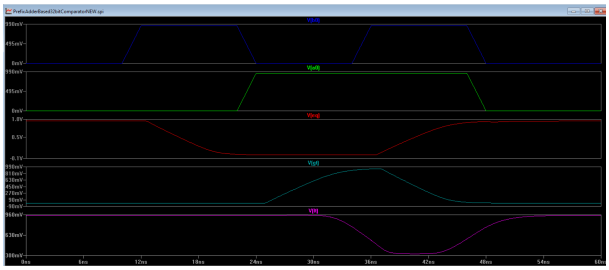

Fig. 22. Propagation Delay When $V_{dd} = 0.95V$.

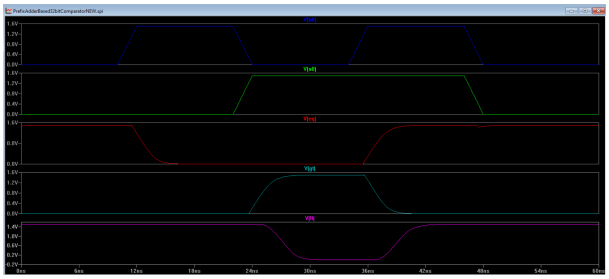

Fig. 23. Propagation Delay When $V_{dd} = 1.9V$.

2) Power:

1) Static power was tested when both inputs are low, power consumed was as follows:

$$P = 300.149nW$$

2) Dynamic Power Dynamic power of each greater than and less than cases were tested, results were as follows:

TABLE IV

Vdd (V)	LT	Eq	GT
0.95	31.761μ W	$300.149nW$	$36.015\mu W$
1.5	$38.628\mu W$	$29.77\mu W$	$41.278\mu W$

VI. Future Work

The proposed comparator architecture demonstrates significant improvements in both speed and power consumption. However, several further optimizations can be explored to enhance its performance even more. These optimizations primarily focus on refining the tree structure and increasing the throughput of the comparator.

A. PG to PG* Block Replacement

One potential optimization involves selectively replacing some levels of the tree structure from conventional Propagate-Generate (PG) blocks to the proposed PG* blocks. The key distinction between a PG block and a PG* block lies in the nature of the inputs and outputs they handle. Where PG* saves 2 inverters of each PG* block since it uses inverters from previous PG ones as common inverters.

The previous approach is possible only because of the dual nature of the pull-up and pull-down circuits in CMOS design. The equations governing these operations are as follows:

OutputPG

$$\text{Generate} = \overline{G_2} \cdot (\overline{P_2} + \overline{G_1}) \quad (1)$$

$$\text{Propagate} = \overline{P_1 \cdot P_2} \quad (2)$$

$$\text{Equate} = \overline{E_1} + \overline{E_2} \quad (3)$$

OutputPG*

$$\text{Generate} = G_2 + P_2 \cdot G_1 \quad (4)$$

$$\text{Propagate} = P_1 \cdot P_2 \quad (5)$$

$$\text{Equate} = E_1 + E_2 \quad (6)$$

By incorporating PG* blocks in alternate positions within the tree structure, it is possible to reduce the number of inverters in each block. This reduction directly contributes to lowering both the delay and power consumption. Specifically, in a 32-bit comparator, the critical path consists of $\log_2(n) = 5$ PG blocks. Replacing these with PG* blocks reduces the critical path delay by 5 inverter delays. Additionally, with 31 PG blocks in the comparator, this modification yields a power reduction equivalent to the elimination of 62 inverters.

B. Pipeline Optimization

Another potential enhancement involves pipelining the tree structure. By inserting D flip-flops between each level of the tree, the architecture can be transformed into a pipelined design. This modification would allow for one comparison to be completed per clock cycle, thereby significantly increasing the throughput of the comparator. Pipelining introduces a form of parallelism that enables the comparator to handle multiple comparisons simultaneously, making it more suitable for high-frequency applications.

C. Summary of Future Work

These proposed optimizations, including the selective use of PG* blocks and the introduction of pipelining, offer promising avenues for further enhancing the performance of the comparator. The implementation of these techniques could lead to even greater reductions in delay and power consumption, as well as increased throughput, making the comparator more efficient and better suited for a wide range of digital applications.

In summary, the future work on this architecture could focus on exploring these optimizations in greater detail, conducting in-depth simulations, and implementing these improvements in real-world applications to validate their effectiveness.

VII. Conclusion

The proposed comparator architecture offers a highly efficient means of comparing binary numbers by leveraging the principles of a prefix adder. Its hierarchical tree structure allows for rapid computation of the final carry or borrow bit, enabling the comparator to achieve a latency of $O(\log n)$. This makes it an ideal choice for high-speed digital systems where quick and accurate comparison of binary numbers is essential.