Indian Institute of Information Technology
Design and Manufacturing, Kurnool

# 8-Bit Kogge-Stone Adder: RTL to GDSII

VLSI System Design (EC-307)

Faculty: **Dr. P. Ranga Babu**

Submitted by:

**Pranjal Upadhyay**
Roll No: **523EC0012**
Integrated B. Tech and M. Tech
Department of Electronics and Communication Engineering

Date: October 18, 2025

# Contents

## 1 Abstract

This project focuses on the complete design flow of an 8-Bit Kogge-Stone adder, a high-performance parallel prefix adder used in digital arithmetic circuits. The implementation spans from Register Transfer Level (RTL) design to Graphic Design System II (GDSII) layout, ensuring a fully synthesizable and manufacturable VLSI design. The adder is optimized for speed and area efficiency in 90 nm CMOS technology, utilizing advanced EDA tools for synthesis, timing analysis, place-and-route, and physical verification. Key aspects include RTL coding in Verilog, functional verification through simulation, synthesis for gate-level netlist generation, static timing analysis, floorplanning, placement, routing, and final DRC/LVS checks. The project demonstrates the RTL-to-GDSII flow, highlighting challenges in timing closure, power optimization, and design rule compliance, resulting in a robust adder suitable for high-speed applications in microprocessors and DSP systems.

## 2 Introduction

In contemporary Very Large Scale Integration (VLSI) design, arithmetic circuits form the core of most computational systems. Among these, binary adders constitute a fundamental component within arithmetic and logic units (ALUs), digital signal processors (DSPs), and general-purpose processors. The performance of these systems is largely determined by the efficiency of the adder architecture employed, as addition is a frequent and timing-critical operation in most data paths.

Traditional adder architectures such as the Ripple Carry Adder (RCA) exhibit linear propagation delay with respect to the operand width, since each carry signal must propagate sequentially through all bit positions. Although simple and area-efficient, this structure becomes a major bottleneck in high-speed designs. To overcome this limitation, parallel prefix adders (PPAs) have been developed to compute carry signals in a hierarchical and parallel manner, thereby significantly reducing overall latency.

Among the various PPA topologies, the **Kogge–Stone Adder (KSA)** is recognized for its superior speed and regular structure. It achieves logarithmic carry computation time through an efficient prefix network that allows simultaneous carry generation across multiple bit positions. The KSA offers minimal logic depth and bounded fan-out, resulting in reduced delay compared to other prefix-based designs such as Brent–Kung or Sklansky adders. However, these advantages are achieved at the expense of increased interconnect complexity and routing area.

In this work, an **8-bit Kogge–Stone Adder** has been designed, synthesized, and physically implemented using a complete **RTL-to-GDSII** design flow in **180 nm CMOS technology**. The objective of this project is to realize a high-performance adder with optimized timing characteristics while maintaining acceptable area and power efficiency. The design process follows a structured methodology beginning with the Register Transfer Level (RTL) modeling and functional verification using Verilog Hardware Description Language (HDL). Logic synthesis is performed using **Cadence Genus** to generate a gate-level netlist based on standard cell libraries. Physical implementation, including floorplanning, placement, clock tree synthesis (CTS), routing, and final layout generation, is carried out using **Cadence Innovus**. Comprehensive post-synthesis and post-layout analyses are conducted to evaluate the design in terms of **timing, power, and area**. The final layout is verified to be Design Rule Check (DRC) and Layout Versus Schematic (LVS) clean, ensuring manufacturability and functional correctness.

This project demonstrates a complete ASIC implementation of a high-speed arithmetic module, showcasing both the theoretical efficiency of the Kogge–Stone architecture and the practical aspects of digital design flow in a real silicon environment.

# 3 Theory and Working Principle

The Kogge–Stone Adder (KSA) is a parallel prefix adder architecture designed for high-speed binary addition. It minimizes the carry propagation delay by computing carry signals in a logarithmic number of stages with respect to the operand width. The architecture is optimized to achieve minimal logical depth and bounded fan-out, resulting in improved timing performance at the expense of increased wiring and silicon area. The Kogge–Stone topology is widely adopted in high-performance arithmetic logic units (ALUs), digital signal processors (DSPs), and microprocessor datapaths where low-latency addition is critical.

## 3.1 Fundamental Concept of Parallel Prefix Adders

In binary addition, each bit position requires two intermediate signals known as the *generate* and *propagate* functions, defined for each bit index $i$ as follows:

$$G_i = A_i \cdot B_i \tag{1}$$

$$P_i = A_i \oplus B_i \tag{2}$$

where $A_i$ and $B_i$ are the $i^{th}$ bits of the two operands. The generate signal $G_i$ indicates that a carry will be produced at that bit, whereas the propagate signal $P_i$ indicates that a carry-in will be propagated to the next stage.
The carry output for each bit position can be recursively defined as:

$$C_{i+1} = G_i + (P_i \cdot C_i) \tag{3}$$

In a conventional Ripple Carry Adder (RCA), each carry signal must be computed sequentially, resulting in a linear propagation delay proportional to the number of bits. In contrast, the Kogge–Stone Adder utilizes a hierarchical prefix computation network that evaluates carry signals in parallel, thereby reducing the overall delay to a logarithmic function of the adder width.

## 3.2 Prefix Operator Definition

The prefix computation in the Kogge–Stone Adder is based on a binary associative operator, denoted by $\circ$, which combines two pairs of generate and propagate signals as follows:

$$(G_k, P_k) \circ (G_j, P_j) = (G_k + P_k \cdot G_j,\ P_k \cdot P_j) \tag{4}$$

This operator enables the recursive computation of carry generation across multiple bit positions. For an $n$-bit adder, the prefix network computes the group generate and propagate terms through $\log_2(n)$ levels of logic, thus achieving a logarithmic carry computation time.

## 3.3 Kogge–Stone Architecture

The Kogge–Stone Adder consists of three distinct functional stages: pre-processing, prefix computation, and post-processing. Each stage contributes a specific function toward efficient and parallel carry computation.

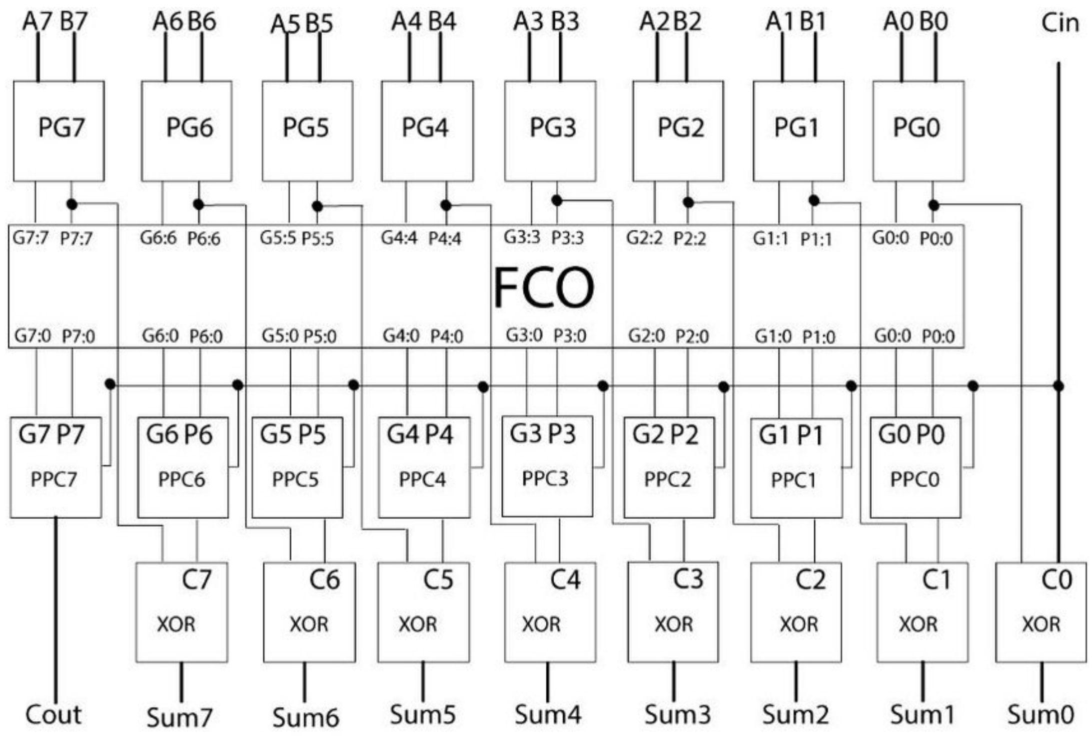


Figure 1: Complete architecture of 8-bit Kogge–Stone Adder.

### 3.3.1 Pre-Processing Stage

The pre-processing stage computes the individual generate ($G_i$) and propagate ($P_i$) signals for each bit of the input operands. This stage operates entirely in parallel and provides the base signals required for subsequent prefix computations.

### 3.3.2 Prefix Computation Stage

The prefix computation stage forms the core of the Kogge–Stone architecture. It employs a tree-like network of prefix cells to iteratively combine generate and propagate pairs over increasing bit spans. The process ensures that each bit position has access to the correct carry-in value derived from all preceding bits.

For an 8-bit Kogge–Stone Adder, the carry computation network requires three levels of prefix computation, corresponding to $\log_2(8) = 3$. Each level doubles the span of carry propagation, leading to complete carry availability after three hierarchical stages.

### 3.3.3 Post-Processing Stage

In the post-processing stage, the final sum bits are computed using the propagate signals and the carry-in values obtained from the prefix network. The sum for each bit is given by:

$$S_i = P_i \oplus C_i \tag{5}$$

The final carry-out of the adder is taken from the most significant carry node of the prefix tree. This stage performs a single-level XOR operation and contributes minimally to the overall delay.

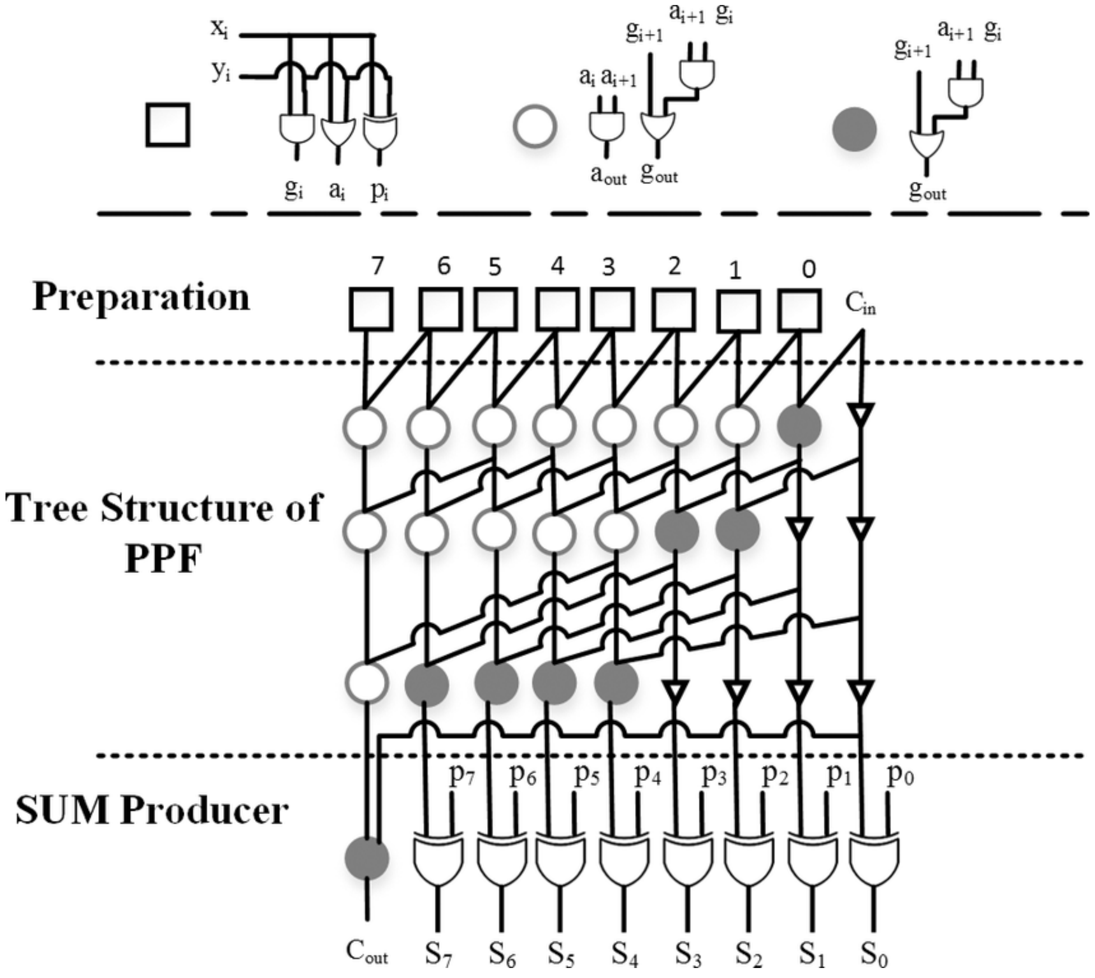


Figure 2: Complete structure of 8-bit Kogge–Stone Adder.

### 3.4 Characteristics and Design Metrics

The Kogge–Stone Adder offers a delay of $O(\log_2 n)$, compared to $O(n)$ for the Ripple Carry Adder, making it significantly faster for wide operand additions. However, this improvement comes with an increase in area and wiring complexity, as the prefix network requires a larger number of interconnects and buffer stages. Despite these trade-offs, the Kogge–Stone architecture remains one of the most efficient and widely adopted solutions for high-speed addition in performance-critical digital systems.

## 4 Design Implementation in Verilog

The Kogge–Stone Adder was implemented using Verilog HDL to describe the RTL behavior and structural hierarchy of the design. The functionality was verified through simulation using an appropriate testbench to ensure correctness of the arithmetic operations and carry propagation.

### 4.1 Verilog Source Code

```verilog
`timescale 1ns / 1ps
module kogge_stone_adder #(parameter PRECISION = 8)(
    input  [PRECISION-1:0] operand_a_i,
    input  [PRECISION-1:0] operand_b_i,
    output [PRECISION-1:0] result_o,
    output overflow_o
);
    // Ceiling log2 function for number of stages
    function integer clog2;
        input integer value;
        integer i;
        begin
            clog2 = 0;
            for (i = value - 1; i > 0; i = i >> 1)
                clog2 = clog2 + 1;
        end
    endfunction

    localparam NUM_STEPS = clog2(PRECISION);

    wire [(NUM_STEPS+1)*PRECISION-1:0] generates;
    wire [NUM_STEPS*PRECISION-1:0]     propagates;

    genvar k, i;

    // Stage 0: Initial generate and propagate
    generate
        for (i = 0; i < PRECISION; i = i + 1) begin : init_stage
            assign generates[i] = operand_a_i[i] & operand_b_i[i];
            assign propagates[i] = operand_a_i[i] ^ operand_b_i[i];
        end
    endgenerate

    // Prefix stages
    generate
        for (k = 1; k < NUM_STEPS; k = k + 1) begin : prefix_stage
            for (i = 0; i < PRECISION; i = i + 1) begin : prefix_bit
                if (i >= (1 << (k-1))) begin : update
                    assign generates[k*PRECISION + i] = generates[(k-1)*PRECISION + i] |
                        (propagates[(k-1)*PRECISION + i] & generates[(k-1)*PRECISION + i - (1 << (k-1))]);
                    assign propagates[k*PRECISION + i] = propagates[(k-1)*PRECISION + i] &
                        propagates[(k-1)*PRECISION + i - (1 << (k-1))];
                end else begin : passthrough
                    assign generates[k*PRECISION + i] = generates[(k-1)*PRECISION + i];
                    assign propagates[k*PRECISION + i] = propagates[(k-1)*PRECISION + i];
                end
            end
        end
    endgenerate

    // Final stage
    generate
        for (i = 0; i < PRECISION; i = i + 1) begin : final_stage
```

```verilog
            if (i >= (1 << (NUM_STEPS-1))) begin : update
                assign generates[NUM_STEPS*PRECISION + i] = generates[(NUM_STEPS-1)*PRECISION + i] |
                    (propagates[(NUM_STEPS-1)*PRECISION + i] & generates[(NUM_STEPS-1)*PRECISION + i - (1 << (NUM_STEPS-1))]);
            end else begin : passthrough
                assign generates[NUM_STEPS*PRECISION + i] = generates[(NUM_STEPS-1)*PRECISION + i];
            end
        end
    endgenerate

    // Compute sum bits
    assign result_o[0] = propagates[0];
    generate
        for (i = 1; i < PRECISION; i = i + 1) begin : sum_stage
            assign result_o[i] = propagates[i] ^ generates[NUM_STEPS*PRECISION + i - 1];
        end
    endgenerate

    // Overflow detection
    assign overflow_o = (PRECISION > 1) ?
                        (generates[NUM_STEPS*PRECISION + PRECISION - 1] |
                         (propagates[PRECISION-1] & generates[NUM_STEPS*PRECISION + PRECISION - 2])) :
                        generates[NUM_STEPS*PRECISION];
endmodule
```

Listing 1: Verilog implementation of 8-bit Kogge–Stone Adder

## 4.2 Verilog Testbench

```
`timescale 1ns / 1ps
module tb_kogge_stone_adder;

    localparam PRECISION = 8;
    localparam SIM_DELAY = 10;

    // Testbench signals
    reg  [PRECISION-1:0] tb_operand_a;
    reg  [PRECISION-1:0] tb_operand_b;
    wire [PRECISION-1:0] tb_result;
    wire                 tb_overflow;

    // Expected results for verification
    wire [PRECISION:0]   expected_sum;
    wire                 expected_overflow;

    assign expected_sum      = tb_operand_a + tb_operand_b;
    assign expected_overflow = expected_sum[PRECISION];

    // Instantiate DUT
    kogge_stone_adder #(
        .PRECISION(PRECISION)
    ) dut_inst (
        .operand_a_i(tb_operand_a),
        .operand_b_i(tb_operand_b),
        .result_o   (tb_result),
        .overflow_o (tb_overflow)
    );

    // Test stimulus
    initial begin
        $display("-------------------------------");
        $display("Starting Kogge-Stone Adder Testbench...");
        $display("PRECISION = %0d", PRECISION);
        $display("-------------------------------");
        $display("Time\t A \t + \t B \t | \t DUT Result \t DUT
           Ovflw \t | \t Expected \t Exp Ovflw \t Status");
        $display("-------------------------------");

        tb_operand_a = 0;
        tb_operand_b = 0;
        #SIM_DELAY;
        check_result(8'd0, 8'd0);

        check_result(8'd10, 8'd25);
        check_result(8'd88, 8'd42);
        check_result({PRECISION{1'b1}}, 8'd0); // Max value + 0
        check_result({PRECISION{1'b1}}, 8'd1); // Overflow test
        check_result(8'd150, 8'd150);          // Overflow test

        // Random test cases
        $display("\n--- Running 10 Random Test Cases ---");
        for (integer i = 0; i < 10; i = i + 1) begin
            check_result($random, $random);
        end
        $display("--- End of Random Test Cases ---\n");

        $display("------------------------------");
        $display("Testbench simulation finished.");
        $display("------------------------------");
        $finish;
    end

    // Self-checking task
    task check_result;
        input [PRECISION-1:0] a;
        input [PRECISION-1:0] b;
        begin
            tb_operand_a = a;
            tb_operand_b = b;
            #SIM_DELAY;

            if (tb_result === expected_sum[PRECISION-1:0] &&
               tb_overflow === expected_overflow) begin
                $display("%0t\t %d\t + \t %d\t | \t\t %d \t\t %b
                   \t\t | \t\t %d \t\t %b \t\t PASS",
                         $time, tb_operand_a, tb_operand_b,
                            tb_result, tb_overflow,
                         expected_sum[PRECISION-1:0],
                            expected_overflow);
            end
```

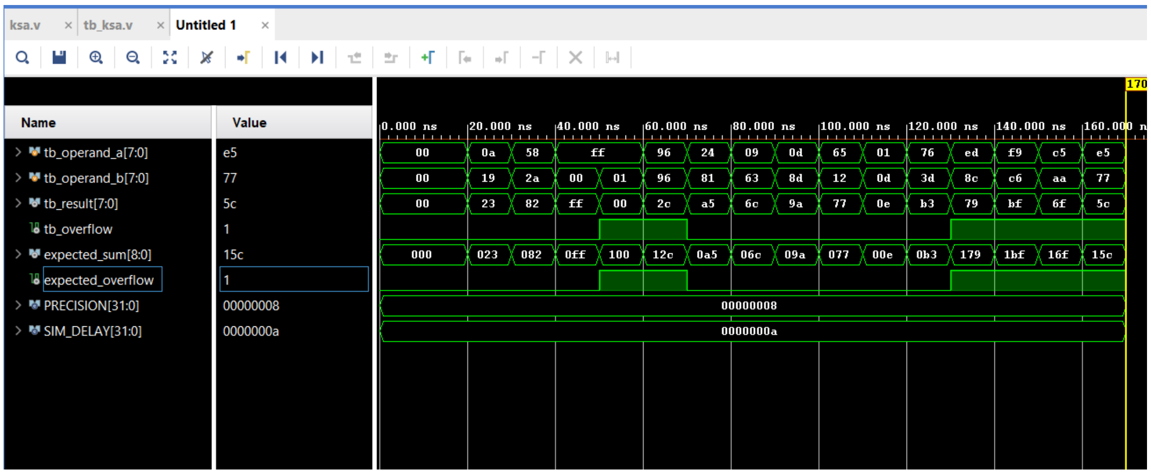
```
                else begin
                    $error("MISMATCH DETECTED!");
                    $display("%0t\t %d\t + \t %d\t | \t\t %d \t\t %b \t\t | \t\t %d \t\t %b \t\t FAIL",
                                $time, tb_operand_a, tb_operand_b, tb_result, tb_overflow,
                                expected_sum[PRECISION-1:0], expected_overflow);
                end
            end
        endtask

endmodule
```

Listing 2: Testbench for 8-bit Kogge–Stone Adder

Both the Verilog source code (Listing 1) and the corresponding testbench (Listing 2) were simulated and verified to ensure correct arithmetic operation and carry propagation across all bit positions.

## 4.3 Simulation Results

The functional verification of the 8-bit Kogge–Stone Adder was carried out using the Verilog testbench described in the previous section. The simulation was performed in the Xilinx Vivado environment to validate the correctness of the adder's logic and carry propagation mechanism. The design was compiled, elaborated, and simulated at RTL level, and the resulting waveforms were analyzed to confirm expected arithmetic behavior.



| Name | Value |
|---|---|
| tb_operand_a[7:0] | e5 |
| tb_operand_b[7:0] | 77 |
| tb_result[7:0] | 5c |
| tb_overflow | 1 |
| expected_sum[8:0] | 15c |
| expected_overflow | 1 |
| PRECISION[31:0] | 00000008 |
| SIM_DELAY[31:0] | 0000000a |

```
tb_operand_a   00  0a  58  ff  96  24  09  0d  65  01  76  ed  f9  c5  e5
tb_operand_b   00  19  2a  00  01  96  81  63  8d  12  0d  3d  8c  c6  aa  77
tb_result      00  23  82  ff  00  2c  a5  6c  9a  77  0e  b3  79  bf  6f  5c
expected_sum   000 023 082 0ff 100 12c 0a5 06c 09a 077 00e 0b3 179 1bf 16f 15c
PRECISION      00000008
SIM_DELAY      0000000a
```

Figure 3: RTL Simulation waveform of 8-bit Kogge–Stone Adder showing sum and carry outputs for multiple input combinations.

During the simulation, several representative input vectors were applied to verify the adder's performance across different logical cases, including:

- Addition with zero operands to validate the base functionality.
- Propagation-only cases (e.g., $A = 8'b01010101$, $B = 8'b10101010$) to confirm correct carry propagation.
- Carry-generation-heavy cases (e.g., $A = 8'b11111111$, $B = 8'b00000001$) to examine the carry chain behavior.
- Random test vectors to ensure robustness against general input conditions.

The simulation waveform (Figure 3) confirms that:

1. The `Sum` output matches the expected arithmetic sum of the operands for all tested cases.
2. The `Cout` (carry-out) signal is correctly generated when the addition exceeds 8-bit capacity.
3. The propagation delay between input changes and output stabilization is consistent with theoretical expectations of a parallel-prefix adder structure.

No functional mismatches or timing anomalies were observed during RTL simulation, indicating that the Verilog description of the Kogge–Stone Adder is logically correct and stable across all operating conditions. The verified RTL was subsequently used as input for synthesis in Cadence Genus.

## 5 Cadence Implementation and Results - 180 nm

The design of the 8-bit Kogge–Stone Adder was synthesized and physically implemented using Cadence Genus and Innovus. This section details the methodology, results, and analysis of the implementation in 180 nm CMOS technology.

### 5.1 Synthesis in Cadence Genus

The Verilog RTL of the 8-bit Kogge–Stone Adder was compiled and elaborated in Cadence Genus. Standard cell libraries corresponding to 180 nm CMOS technology were used for logic mapping. Timing constraints were applied via an SDC file, including clock period, input/output delays, and design effort levels.
The synthesis process generated a gate-level netlist and produced comprehensive reports for timing, area, and power. The key metrics for the 180 nm implementation are summarized in Table 1.

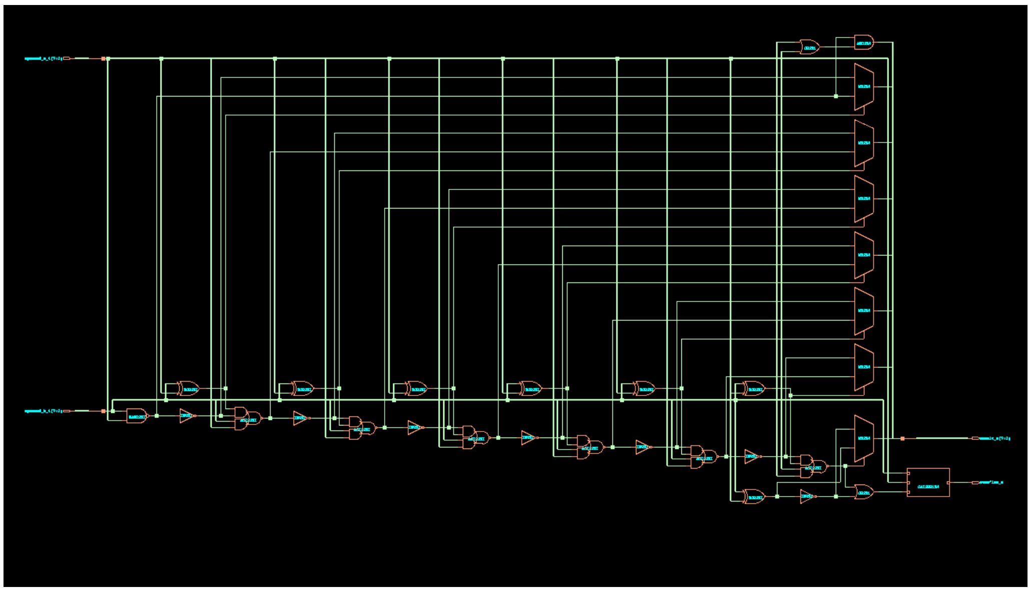


Figure 4: Gate-level schematic of 8-bit Kogge–Stone Adder synthesized in Cadence Genus.

Table 1: Synthesis results of 8-bit Kogge–Stone Adder in Cadence Genus (180 nm).

| **Metric** | **Value** | **Unit** | **Notes** |
|---|---|---|---|
| Total Area | 652 | $\mu m^2$ | Cell area before routing |
| Maximum Delay | TBD | ns | Path delay to be filled from timing report |
| Total Power | 85.8 | µW | Combined dynamic + static power |
| Number of Cells | 39 | - | 32 standard cells + 7 inverters placed |

## 5.2 Physical Design in Cadence Innovus

Post-synthesis, the gate-level netlist was imported into Cadence Innovus for physical implementation. The design flow included floorplanning, placement, clock tree synthesis (CTS), routing, and final layout generation. Standard design rules were enforced, and DRC/LVS verification was performed to ensure design correctness and manufacturability.

### 5.2.1 Floorplan

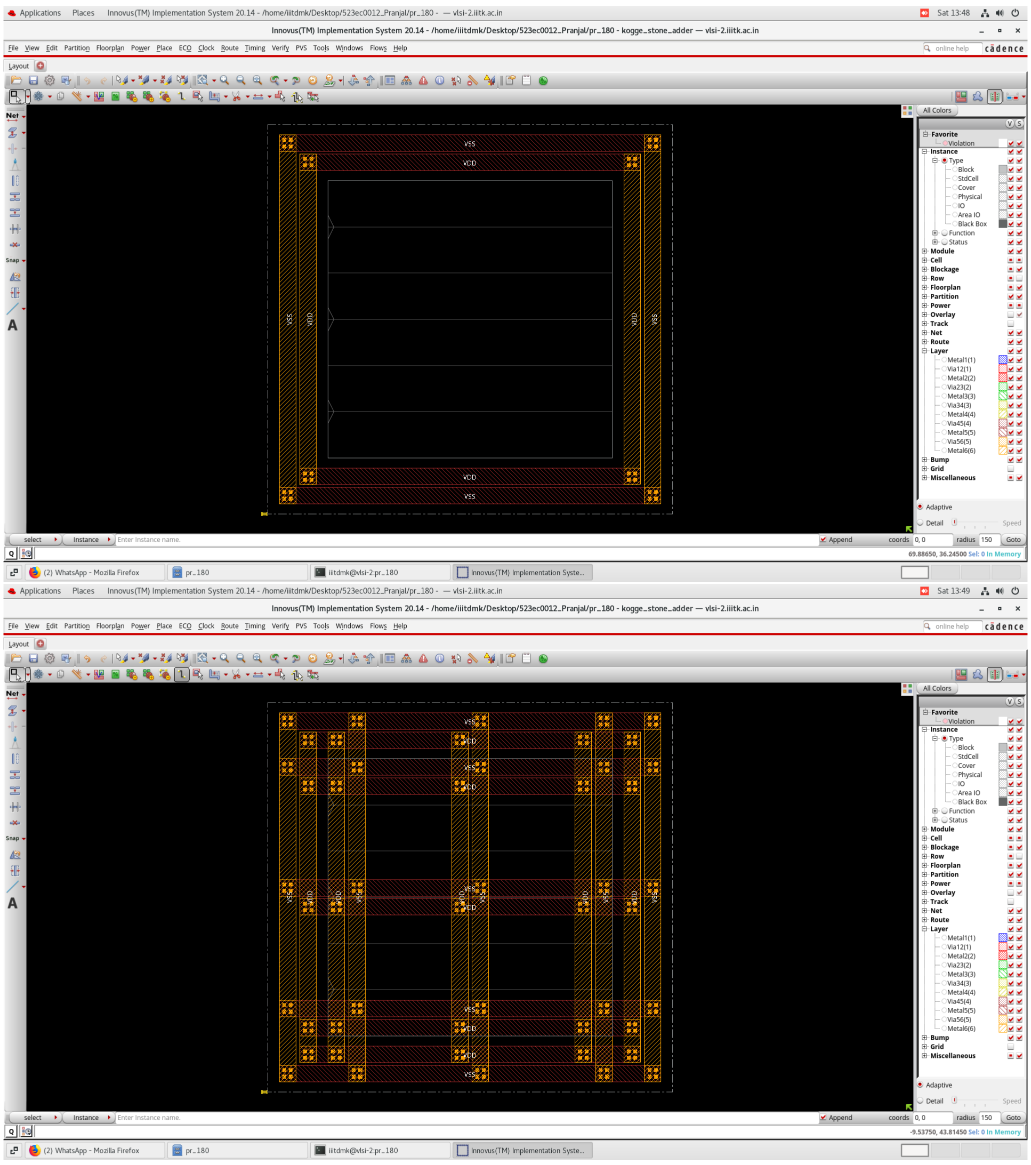

Innovus(TM) Implementation System 20.14 - /home/iitdmk/Desktop/523ec0012_Pranjal/pr_180 - kogge_stone_adder — vlsi-2.iiitk.ac.in
File View Edit Partition Floorplan Power Place ECO Clock Route Timing Verify PVS Tools Windows Flows Help
cadence
VSS
VDD
VDD
VSS
Sat 13:48
Sat 13:49

Figure 5: Floorplan of the 8-bit Kogge–Stone Adder in Cadence Innovus.

### 5.2.2 Layout

Design Rule Check (DRC) and Layout Versus Schematic (LVS) verification confirmed that the layout adheres to manufacturing constraints and accurately represents the netlist. Post-layout timing analysis was performed to evaluate the impact of parasitic capacitances on delay and to ensure timing closure.

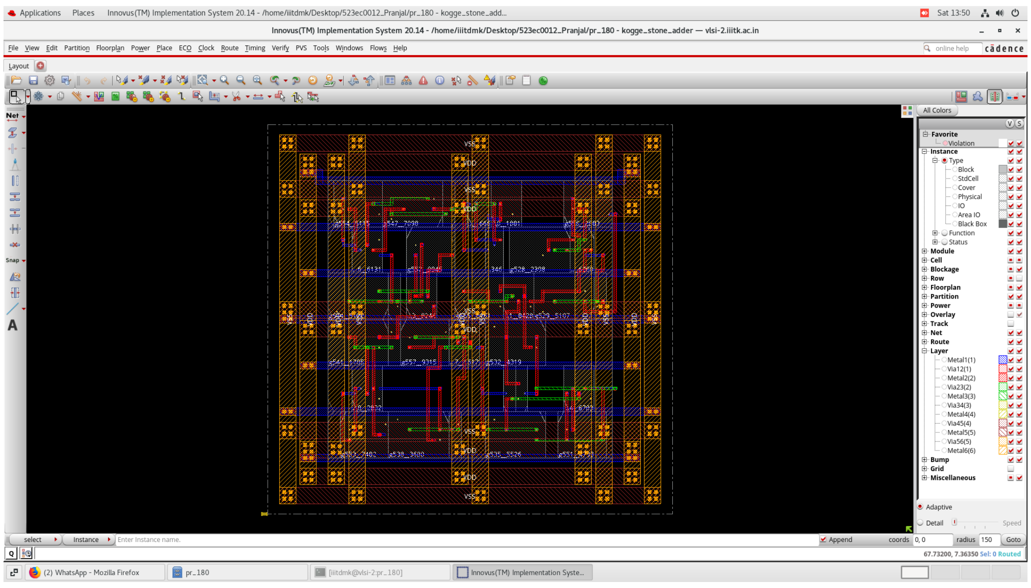
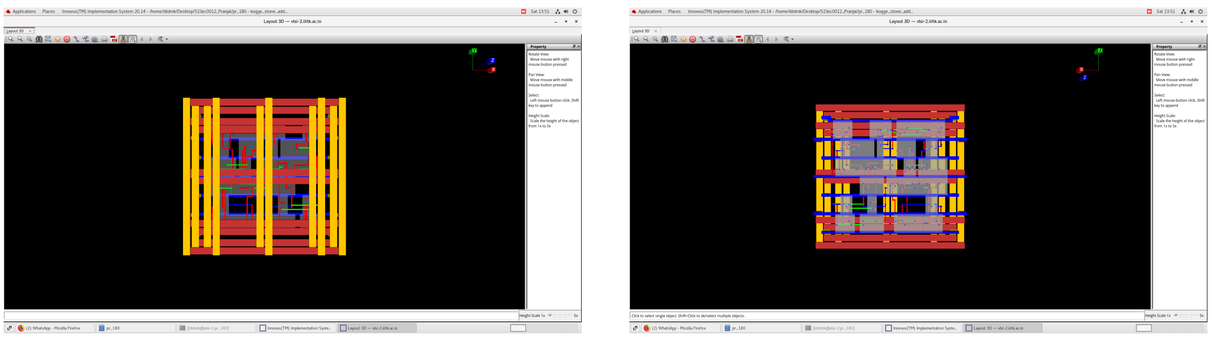



Figure 6: Routed layout of the 8-bit Kogge–Stone Adder. Top: 2D layout showing placement and interconnects. Bottom: 3D routed layouts illustrating layer stacking and interconnect connectivity.

## 5.3 Post-Layout Verification

Table 2: Post-layout timing, power, and area metrics (180 nm).

| Metric | Value | Unit | Notes |
|---|---|---|---|
| Total Area | 1881.9 | $\mu m^2$ | Die area: 44.22 x 42.56 $\mu m^2$ |
| Maximum Delay | TBD | ns | Path delay to be filled from post-layout timing report |
| Total Power | 85.8 | µW | Dynamic + static power |

# 6 Cadence Implementation and Results - 90 nm

The design of the 8-bit Kogge–Stone Adder was synthesized and physically implemented using Cadence Genus and Innovus. This section details the methodology, results, and analysis of the implementation in 90 nm CMOS technology.

## 6.1 Synthesis in Cadence Genus

The Verilog RTL of the 8-bit Kogge–Stone Adder was compiled and elaborated in Cadence Genus. Standard cell libraries corresponding to 90 nm CMOS technology were used for logic mapping. Timing constraints were applied via an SDC file, including clock period, input/output delays, and design effort levels.

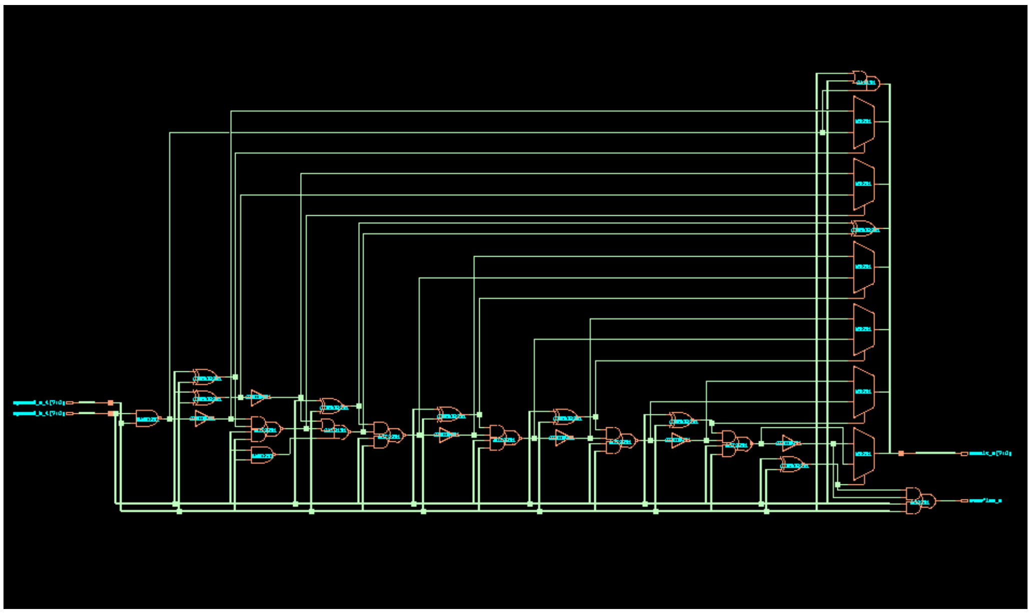


Figure 7: Gate-level schematic of 8-bit Kogge–Stone Adder synthesized in Cadence Genus (90 nm).

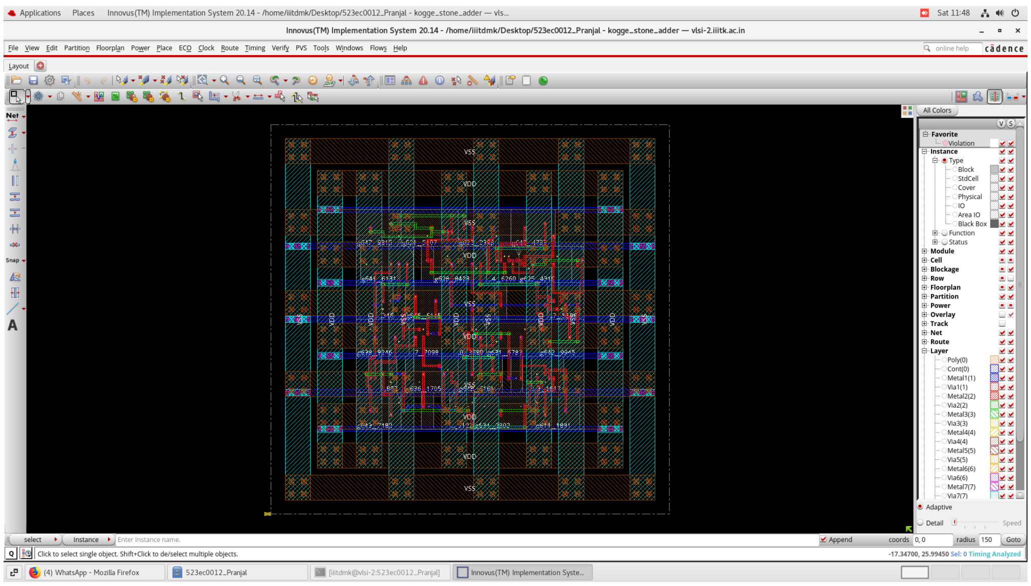
The synthesis process generated a gate-level netlist and produced comprehensive reports for timing, area, and power. The key metrics for the 90 nm implementation are summarized in Table 3.

Table 3: Synthesis results of 8-bit Kogge–Stone Adder in Cadence Genus (90 nm).

| **Metric** | **Value** | **Unit** | **Notes** |
|---|---|---|---|
| Total Area | 176.36 | $\mu m^2$ | Cell area from synthesis report |
| Maximum Delay | 2.05 | ns | Critical path delay from timing report |
| Total Power | 21.93 | µW | Combined dynamic + static power |
| Number of Cells | 30 | - | From synthesis report |

## 6.2 Routed Layout in Cadence Innovus

Post-synthesis, the gate-level netlist was imported into Cadence Innovus for routing and final layout generation. Standard design rules were enforced, and DRC/LVS verification was performed to ensure design correctness and manufacturability.


Innovus(TM) Implementation System 20.14 - /home/iiitdmk/Desktop/523ec0012_Pranjal - kogge_stone_adder — vlsi-2.iiitk.ac.in
File View Edit Partition Floorplan Power Place ECO Clock Route Timing Verify PVS Tools Windows Flows Help
cadence
Layout
VSS
VDD
Favorite
Instance
Module
Cell
Blockage
Row
Floorplan
Partition
Power
Overlay
Track
Net
Route
Layer
Click to select single object. Shift+Click to de/select multiple objects.

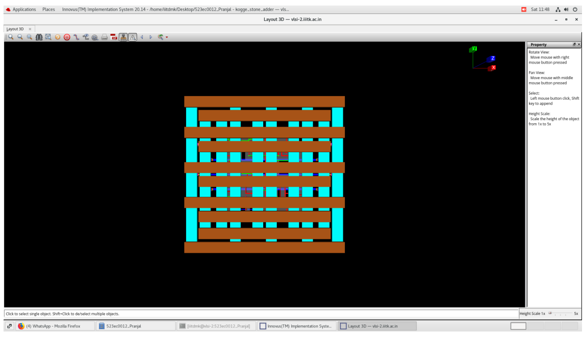
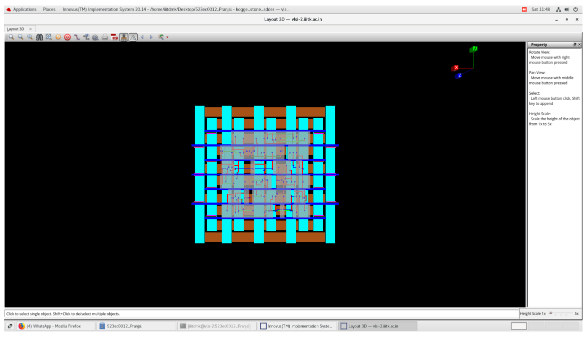



Figure 8: Routed layout of the 8-bit Kogge–Stone Adder in 90 nm. Top: 2D layout showing placement and interconnects. Bottom: 3D routed layouts illustrating layer stacking and interconnect connectivity (if available).

## 6.3 Post-Layout Verification

Design Rule Check (DRC) and Layout Versus Schematic (LVS) verification confirmed that the layout adheres to manufacturing constraints and accurately represents the netlist. Post-layout timing analysis was performed to evaluate the impact of parasitic capacitances on delay and to ensure timing closure.

Table 4: Post-layout timing, power, and area metrics (90 nm).

| Metric | Value | Unit | Notes |
|---|---|---|---|
| Total Area | 791.13 | $\mu m^2$ | Die area from Innovus report (28.42 x 27.84) |
| Maximum Delay | 5.97 | ns | Path delay from post-layout timing report |
| Total Power | 25.08 | µW | Dynamic + static power from post-layout report |

## 7 Conclusion

This report presents the complete design, synthesis, and physical implementation of an 8-bit Kogge–Stone Adder using Cadence Genus and Innovus. The study covers the design methodology, simulation verification, synthesis metrics, and post-layout verification in both 180 nm and 90 nm CMOS technologies. The report documents all stages from RTL to GDSII, including timing, power, and area analyses, providing a comprehensive account of the design process.

## 8 References

1. A. K. Sahu and D. S. Kushwah, "A Review on Different Parallel Prefix Adders for High Speed and Low Power Applications," *International Journal of Scientific Research and Engineering Trends (IJSRET)*, vol. 9, no. 4, pp. 317-321, Jul.-Aug. 2023.

2. A. Mishra and N. Sharma, "Design and Performance Analysis of 64-bit Kogge Stone Adder using GDI and FinFET Technique," *International Research Journal of Engineering and Technology (IRJET)*, vol. 7, no. 3, pp. 4185-4190, Mar. 2020.

3. "Kogge Stone Adder : Circuit, Design, Advantages & Its Applications," *ElProCus*. [Online]. Available: `https://www.elprocus.com/kogge-stone-adder/`.

## 9 GitHub Repository

The complete project files, including Verilog RTL, testbench, SDC constraints, Cadence reports, and layouts, are available at:
`https://github.com/upadhyaypranjal/8-Bit-Kogge-Stone-Adder`