BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

EEE 468 (January 2024)

VLSI Laboratory

Final Project Report

Section: G2 Group: 08

16 Bit Carry Skip Adder

Course Instructors:

Nafis Sadik, Lecturer
Rafid Hassan Palash, Part-Time Lecturer

Signature of Instructor: _____

Academic Honesty Statement:

IMPORTANT! Please carefully read and sign the Academic Honesty Statement, below. Type the student ID and name, and put your signature. *You will not receive credit for this project experiment unless this statement is signed in the presence of your lab instructor.*

"In signing this statement, We hereby certify that the work on this project is our own and that we have not copied the work of any other students (past or present), and cited all relevant sources while completing this project. We understand that if we fail to honor this agreement, We will each receive a score of ZERO for this project and be subject to failure of this course."

Signature: _____ **Full Name: Abu Yousuf Md. Eftekhar Sadik** **Student ID: 1906085**	**Signature:** _____ **Full Name: Md. Shahin Ferdous** **Student ID: 1906088**
Signature: _____ **Full Name: Safin Ahammed** **Student ID: 1906100**	**Signature:** _____ **Full Name: Bickrom Roy** **Student ID: 1906185**

Table of Contents

1 Abstract 1

2 Introduction 1

3 Design 3

3.2 Design Method 3

3.3 Circuit Diagram 3

4 Implementation 5

4.1 Code 5

4.2 Result & Analysis 8

5 Design Analysis and Evaluation 44

5.1 Novelty 44

6 Reflection on Individual and Team work 45

6.1 Individual Contribution of Each Member 45

6.4 Log Book of Project Implementation 45

7 Conclusion 46

8 References 46

1 Abstract

In this project, we developed a 16-bit carry skip adder, which is a type of arithmetic circuit that improves the performance of binary addition operations. The adder was implemented in SystemVerilog, a hardware description language, and we utilized Cadence tools for the design process. To ensure the functionality and reliability of our design, we created a comprehensive testbench that simulates various input scenarios and validates the output against expected results. We also performed layered verification, which involves multiple levels of testing to confirm that the design meets all specified requirements. Once the design was verified, we proceeded with Register Transfer Level (RTL) synthesis to transform our high-level design into a gate-level representation. This step is crucial for preparing the design for physical implementation. Finally, we executed the place and route (PnR) process in Innovus, a tool used for the physical design of integrated circuits. This involved optimizing the layout of the adder to ensure efficient use of space and power while maintaining performance targets.

2 Introduction

The 16-bit Carry-Skip Adder (CSA) is an advanced digital circuit specifically engineered for efficient binary addition. Unlike traditional ripple carry adders (RCAs), which encounter significant delays due to the sequential carry propagation through each bit, the CSA introduces a novel approach to minimize this delay. The CSA operates by dividing the 16 bits into smaller groups, each capable of performing addition in parallel. When a carry occurs in one group, the CSA quickly assesses whether the carry will affect subsequent groups. If not, the adder can "skip" the carry propagation for those groups, allowing for faster overall computation. This method strikes an effective balance between simplicity in design, speed of operation, and economical use of hardware resources. Binary addition is the fundamental process of adding two binary numbers by evaluating each pair of corresponding bits. This operation is carried out bit by bit, starting from the least significant bit and moving towards the most significant bit. During this addition process, it is common for a carry to be generated when the sum of two bits, along with any carry from the previous column, exceeds the value that can be represented by a single bit (which is 1 in binary). This carry may propagate to the next bit position, affecting subsequent calculations. One of the critical challenges in binary addition, particularly in large-bit-width adders, is managing this carry propagation in an efficient manner. Excessive carry delays can significantly hinder performance, particularly as the width of the numbers being added increases. A common implementation of binary addition is the Ripple Carry Adder (RCA). In an RCA, the carry output from one stage must sequentially propagate through each subsequent stage in order for the addition to be completed. For instance, if we are working with an N-bit RCA, the worst-case scenario results in a delay that is proportional to O(N). This means that the total time required for the carry to ripple through N stages can lead to a significant delay, particularly for larger numbers. Because each stage must wait for the carry to be resolved at the previous stage, this sequential dependency can severely limit the speed of operation for adders with larger bit widths. As a result, Ripple Carry Adders are generally less suitable for applications that demand high-speed processing, where

faster adder architectures, like Carry Lookahead Adders, are preferred due to their ability to mitigate carry propagation delays more effectively. The Carry-Skip Adder is designed to effectively tackle the notorious carry propagation delay, a challenge typically encountered in traditional adder circuits. This innovative approach incorporates a mechanism that allows the adder to "skip" redundant carry calculations when certain conditions are met, enhancing its overall performance and speed.

Grouping of Bits: For a 16-bit addition operation, the adder is segmented into smaller and more manageable groups. In this case, it is organized into four distinct groups, each containing four bits. This partitioning allows for concurrent processing within each group, thereby minimizing delays.

Propagation Condition: Each group has a specific condition that determines whether the carry can propagate across all four bits. This is known as the propagate condition (P). If P holds true for a particular group—meaning all bits in that group are capable of transmitting the carry—the carry can seamlessly flow through the entire group.

Skip Logic: When the propagate condition is satisfied (P=1), the carry input for that group can be directly passed to the next group without waiting for the carry to ripple through each bit in the current group. This skipping mechanism bypasses unnecessary delay, ensuring that carries are handled more efficiently and speeding up the overall addition process.

3 Design

3.1 Design Method

To design a 16-bit carry-skip adder, we structured the system using four distinct blocks, each representing a 4-bit carry-skip adder. The implementation of each 4-bit carry-skip adder relied on a 4-bit ripple carry adder, which efficiently processes the addition of binary numbers. In our design, each full adder was created using a half adder circuit, which allowed for the effective handling of carry operations. To optimize the addition process, we incorporated a multiplexer (mux) for each 4-bit carry-skip adder. This multiplexer played a crucial role in facilitating the carry skip operation, enabling faster addition by allowing the adder to bypass certain carry calculations under specific conditions. Overall, this design efficiently combines the strengths of ripple carry addition with the advantages of carry skipping, improving performance for 16-bit binary addition.

3.2 Circuit Diagram

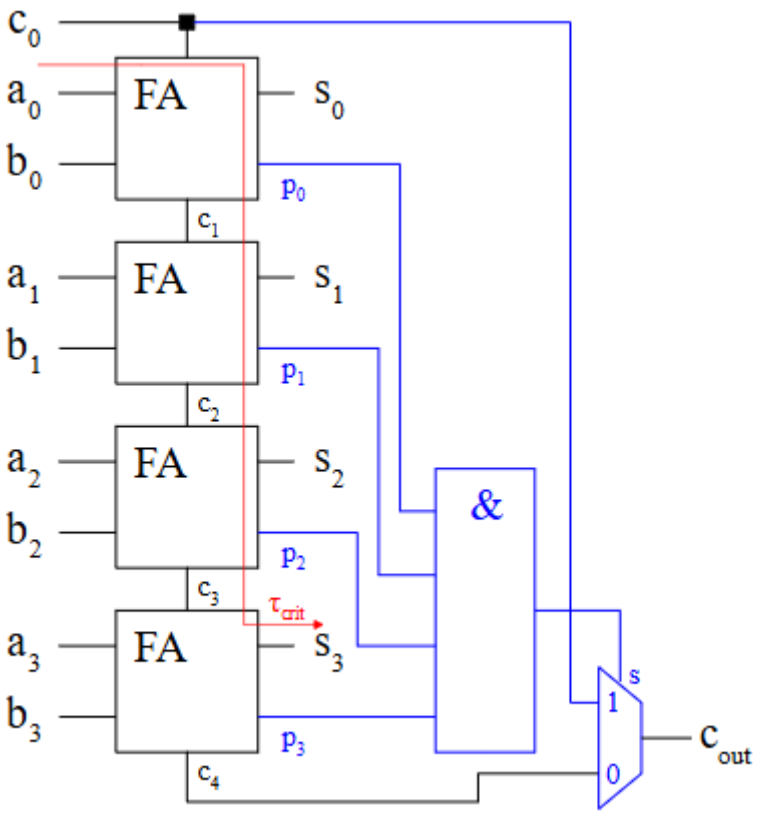


Fig: 4 Bit Carry Skip Adder

This is a 4-bit carry-skip adder, which is an efficient arithmetic circuit used for adding two 4-bit binary numbers. The inputs for the adder are represented as a[3:0] and b[3:0], where each input is a 4-bit binary number. The addition process is carried out using four full adders, each designated to compute one bit of the resultant sum. The full adders generate two outputs for each bit: the sum output and the carry output. The sum outputs from each adder contribute to

the final result, while the carry outputs are crucial for determining whether a carry-over occurs to the next higher bit. To optimize the addition process, the carry propagation signals (denoted as p[3:0]) are connected through an AND gate. This gate monitors the propagation conditions of the carry; specifically, it checks whether all the propagation signals are set to 1. If they are, it indicates that the carry can be skipped for the next higher bit, allowing the adder to operate more quickly by bypassing unnecessary calculations. However, if any of the propagation signals are 0, the carry bit c4 needs to be computed more deliberately. In this case, the carry is processed through a 2:1 multiplexer (MUX), which selects between the skipped carry and the calculated carry output based on the state of the propagation signals. This design enables faster addition by reducing the time taken to generate carry signals, thus improving the overall efficiency of the 4-bit carry-skip adder.

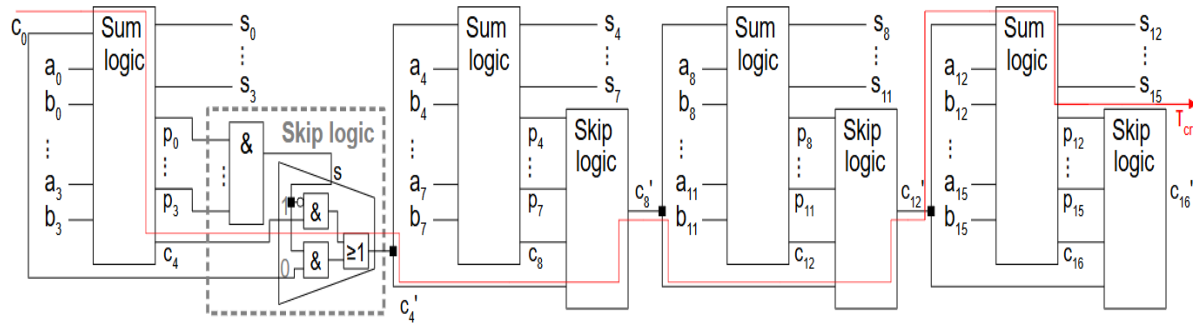


Fig: Full Circuit Diagram of 16 Bit Carry Skip Adder

The circuit diagram presented illustrates a 16-bit carry skip adder, which is a type of digital circuit used for performing addition on binary numbers. Within this diagram, you will notice that the 4-bit carry skip adder block is utilized repeatedly, specifically a total of four times. Each of these 4-bit blocks is responsible for handling a portion of the overall 16-bit addition operation. By employing the carry skip technique, the adder effectively minimizes the propagation delay typically associated with carry operations in traditional adder designs, allowing for faster computation. This design is particularly useful in systems where efficient and rapid arithmetic operations are required.

4 Implementation

4.1 Code

Code for 16 bit carry skip adder

```
module carry_skip_16bit(a, b, cin, sum, cout);
input [15:0] a,b;
input cin;
output cout;
output [15:0] sum;

wire [2:0] c;

carry_skip_4bit csa1(.a(a[3:0]), .b(b[3:0]), .cin(cin), .sum(sum[3:0]), .cout(c[0]));
carry_skip_4bit csa2 (.a(a[7:4]), .b(b[7:4]), .cin(c[0]), .sum(sum[7:4]), .cout(c[1]));
carry_skip_4bit csa3(.a(a[11:8]), .b(b[11:8]), .cin(c[1]), .sum(sum[11:8]), .cout(c[2]));
carry_skip_4bit csa4(.a(a[15:12]), .b(b[15:12]), .cin(c[2]), .sum(sum[15:12]), .cout(cout));

endmodule
```

Code for 4 bit carry skip adder

```
////////////////////////////////////
// Carry Skip Adder 4 bit
////////////////////////////////////

module carry_skip_4bit(a, b, cin, sum, cout);
input [3:0] a,b;
input cin;
output [3:0] sum;
output cout;
wire [3:0] p;
wire c0;
wire bp;

ripple_carry_4_bit rca1 (.a(a[3:0]), .b(b[3:0]), .cin(cin), .sum(sum[3:0]), .cout(c0));
generate_p p1(a,b,p,bp);
mux2X1 m0(.in0(c0),.in1(cin),.sel(bp),.out(cout));
```

Code for 4 ripple carry 4 bit

```
////////////////////////////////
//4-bit Ripple Carry Adder
////////////////////////////////
module ripple_carry_4_bit(a, b, cin, sum, cout);
input [3:0] a,b;
input cin;
wire c1,c2,c3;
output [3:0] sum;
output cout;

full_adder fa0(.a(a[0]), .b(b[0]),.cin(cin), .sum(sum[0]),.cout(c1));
full_adder fa1(.a(a[1]), .b(b[1]), .cin(c1), .sum(sum[1]),.cout(c2));
full_adder fa2(.a(a[2]), .b(b[2]), .cin(c2), .sum(sum[2]),.cout(c3));
full_adder fa3(.a(a[3]), .b(b[3]), .cin(c3), .sum(sum[3]),.cout(cout));
endmodule
```

Code for propagation generation

```
////////////////////////////////////////
// Propagate Generation
////////////////////////////////////////

module generate_p(a,b,p,bp);
input [3:0] a,b;
output [3:0] p;
output bp;
assign p= a^b;//get all propagate bits
assign bp= &p;// and p0p1p2p3 bits
endmodule
```

Code for 1 bit full adder

```
////////////////////////////////////////
//1bit Full Adder
////////////////////////////////////////
module full_adder(a,b,cin,sum, cout);
input a,b,cin;
output sum, cout;
wire x,y,z;
half_adder h1(.a(a), .b(b), .sum(x), .cout(y));
half_adder h2(.a(x), .b(cin), .sum(sum), .cout(z));
or or_1(cout,z,y);
endmodule
```

Code for 1 bit half adder

```
// 1 bit Half Adder
////////////////////////////////////
module half_adder( a,b, sum, cout );
input a,b;
output sum, cout;
xor xor_1 (sum,a,b);
and and_1 (cout,a,b);
endmodule
```

Code for 2:1 MUX

```
//2X1 Mux
//////////////////////////

module mux2X1( in0,in1,sel,out);
input in0,in1;
input sel;
output out;
assign out=(sel)?in1:in0;
endmodule
```

Testbench:

```
// Initialize the inputs
a = 16'd0;
b = 16'd0;
cin = 1'b0;

// Test case 1: Add zero + zero, expect sum = 0, cout = 0
#10;
$display("Test 1: a = %d, b = %d, cin = %b -> sum = %d, cout = %b", a, b, cin, sum, cout);

// Test case 2: Add max values for a and b with cin = 0
a = 16'hFFFF;
b = 16'hFFFF;
cin = 1'b0;
#10;
$display("Test 2: a = %h, b = %h, cin = %b -> sum = %h, cout = %b", a, b, cin, sum, cout);

// Test case 3: Add a random value with carry-in = 1
a = 16'h1234;
b = 16'h5678;
cin = 1'b1;
#10;
$display("Test 3: a = %h, b = %h, cin = %b -> sum = %h, cout = %b", a, b, cin, sum, cout);

// Test case 4: Another random test case
a = 16'hABCD;
b = 16'h1234;
cin = 1'b0;
#10;
$display("Test 4: a = %h, b = %h, cin = %b -> sum = %h, cout = %b", a, b, cin, sum, cout);

// Test case 5: Test for overflow with max value and cin = 1
a = 16'hFFFF;
b = 16'h0001;
cin = 1'b1;
#10;
$display("Test 5: a = %h, b = %h, cin = %b -> sum = %h, cout = %b", a, b, cin, sum, cout);

// End of test cases
```

4.2 Results and Analysis

EP Waveform:

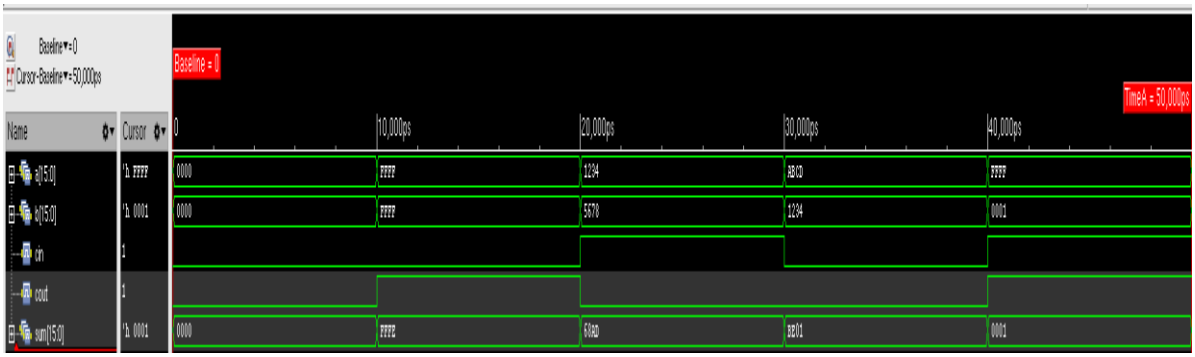


Fig: EP Waveform

Carry not skipped case:

a = 1234 = 0001 0010 0011 0100

b = 5678 = 0101 0110 0111 1000

cin = 1 cout = 0

Carry Skipped Case:

a = ABCD = 1010 1011 1100 1101

b = 1234 = 0001 0010 0011 0100

cin = 0 cout = 0

Layered Verification:

In the VLSI design, design hierarchy is a fundamental concept that involves systematically breaking down intricate systems into smaller, more manageable sub-modules. This process of decomposition allows designers to focus on individual components, which can be designed and verified separately. By employing this hierarchical methodology, the design process becomes more streamlined and organized. It not only enhances reusability—allowing previously designed modules to be integrated into new projects—but also significantly improves the efficiency of verification and debugging.

```
        Loading native compiled code:     ..................... Done
        Design hierarchy summary:
                                  Instances  Unique
                Modules:                 66       8
                Interfaces:               1       1
                Programs:                 1       1
                Primitives:              80       3
                Registers:               52      79
                Scalar wires:            11       -
                Expanded wires:          48       3
                Vectored wires:           5       -
                Named events:             2       6
                Initial blocks:           4       4
                Clocking blocks:          2       2
                Clocking items:           8       8
                Parallel blocks:          1       1
                Cont. assignments:        8       3
                Pseudo assignments:       6       6
                Assertions:               1       1
                Compilation units:        1       1
                SV Class declarations:    7       7
                SV Class specializations: 7       7
                Simulation timescale:     1ps
        Writing initial simulation snapshot: worklib.testbench:sv
Loading snapshot worklib.testbench:sv ..................... Done
SVSEED set randomly from command line: 186535490
```

Fig: Design Hierarchy Summary

This is the design hierarchy summary of layered verification.

```
irun(64): 15.10-s015: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
file: ../testbench/testbench.sv
        program worklib.test:sv
                errors: 0, warnings: 0
        interface worklib.csa_if:sv
                errors: 0, warnings: 0
        module worklib.testbench:sv
                errors: 0, warnings: 0
ncvlog: *W,SPDUSD: Include directory ../design/ given but not used.
        Total errors/warnings found outside modules and primitives:
                errors: 0, warnings: 1
                Caching library 'worklib' ....... Done
        Elaborating the design hierarchy:
        Top level design units:
                $unit_0x60396aef
                testbench
        Building instance overlay tables: .................... Done
```

Fig: Violation and Error Check

We see there is no violation and error here.

```
ncsim> run
-----------------Scoreboard Test Starts-------------------
Passed : a=48818 b= 6958 cin=0 Expected sum=55776 cout=0 Resulted sum=55776 cout
=0
Failed : a=20200 b=48871 cin=1 Expected sum= 3536 cout=0 Resulted sum= 3536 cout
=1
Failed : a=49184 b=51586 cin=0 Expected sum=35234 cout=0 Resulted sum=35234 cout
=1
Failed : a=52147 b=61545 cin=0 Expected sum=48156 cout=0 Resulted sum=48156 cout
=1
Failed : a=54650 b=41613 cin=1 Expected sum=30728 cout=0 Resulted sum=30728 cout
=1
-----------------Scoreboard Test Ends-------------------
Simulation complete via $finish(1) at time 65 NS + 1
../testbench/environment.sv:42     $finish;  // End the simulation after all tas
ks are completed
ncsim> exit
```

Fig: Scoreboard test results

In the scoreboard test result we see one test result is passed and other test results are failed as we intentionally added some bugs to check the testbench.

```
irun(64): 15.10-s015: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
Loading snapshot worklib.testbench:sv ..................... Done
SVSEED set randomly from command line: -1939984310
ncsim> run
-----------------Scoreboard Test Starts------------------
Passed : a=  602 b=40915 cin=1 Expected sum=41518 cout=0 Resulted sum=41518 cout=0
Passed : a=42175 b= 9596 cin=1 Expected sum=51772 cout=0 Resulted sum=51772 cout=0
Passed : a=28079 b=29216 cin=1 Expected sum=57296 cout=0 Resulted sum=57296 cout=0
Failed : a=62194 b=37106 cin=1 Expected sum=33765 cout=0 Resulted sum=33765 cout=1
Passed : a=46932 b=18423 cin=0 Expected sum=65355 cout=0 Resulted sum=65355 cout=0
-----------------Scoreboard Test Ends------------------
Simulation complete via $finish(1) at time 65 NS + 1
../testbench/environment.sv:42     $finish;  // End the simulation after all tasks are completed
ncsim> exit
```

This is another scoreboard test results.

Synthesis:

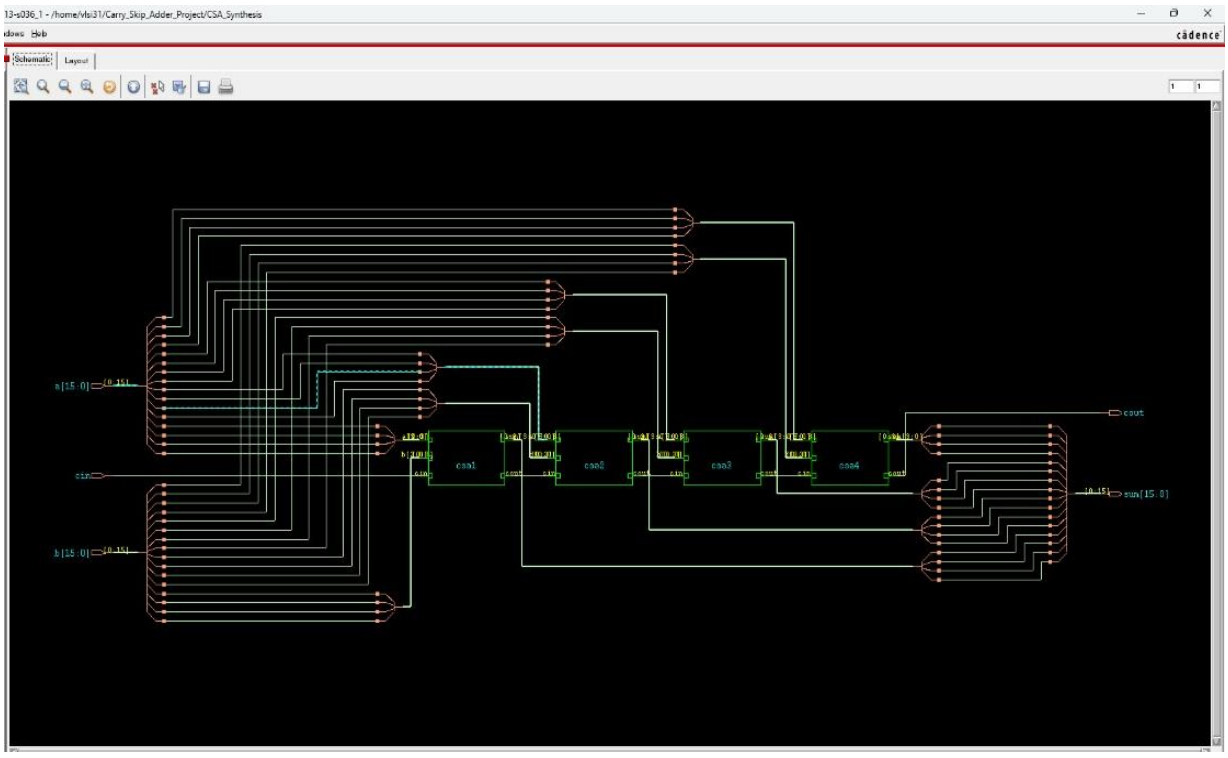


Fig: Full circuit of 16 bit CSA

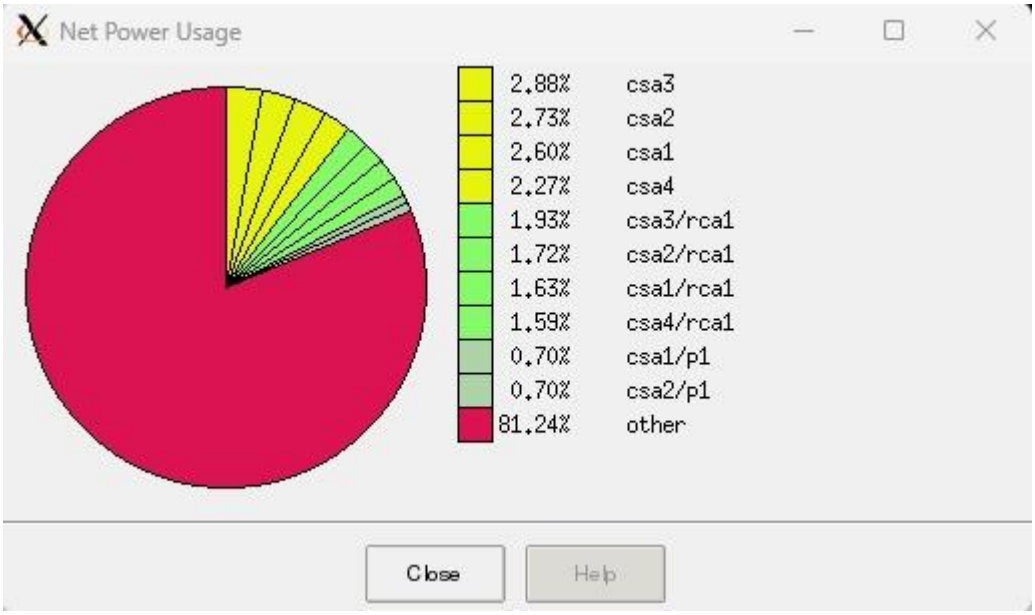


Fig: Pie Chart of Net Power Usage

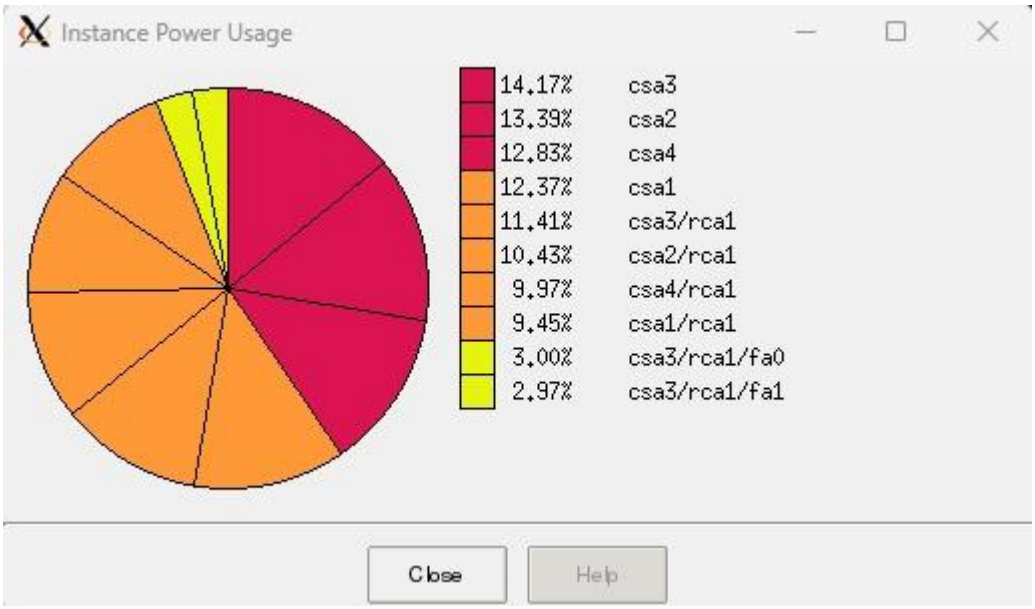
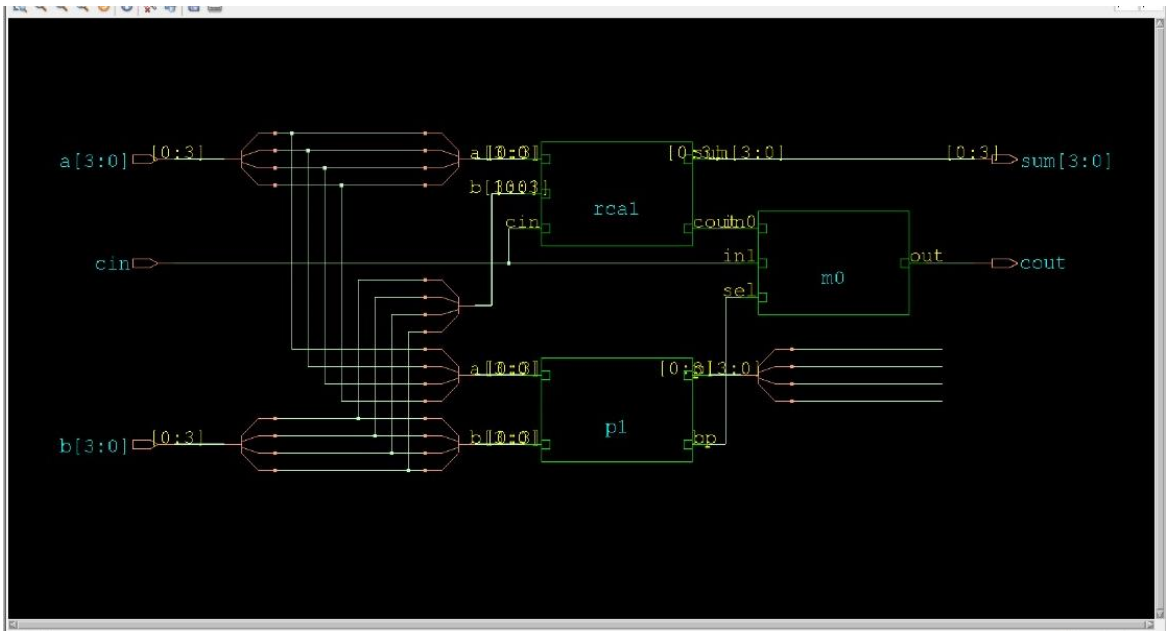
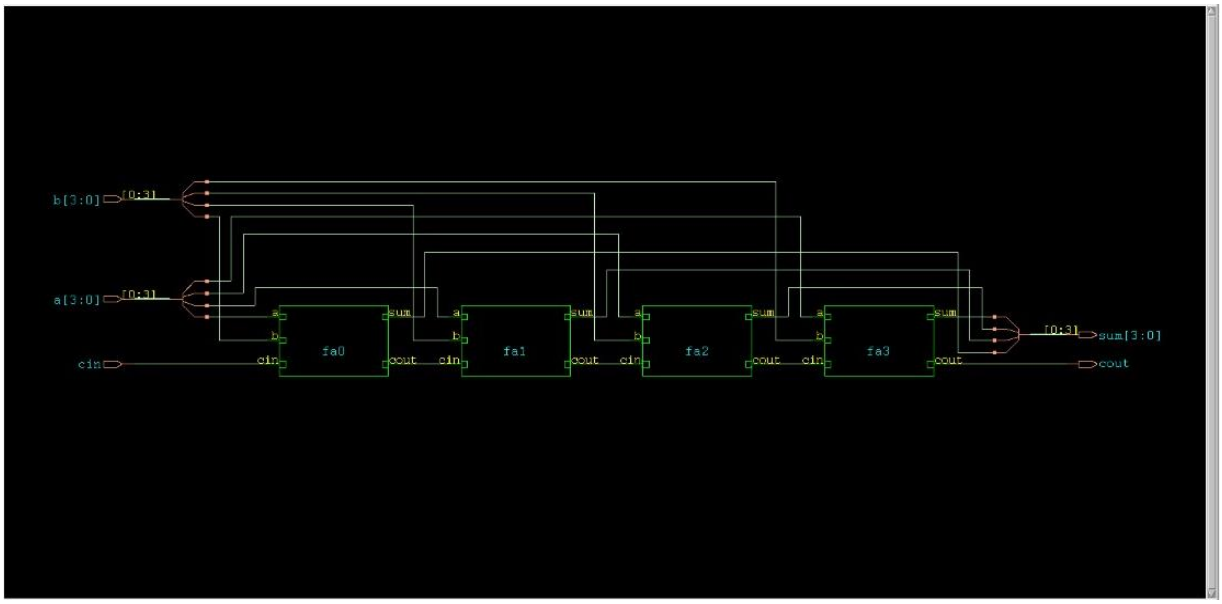


Fig: Pie Chart of Instance Power Usage

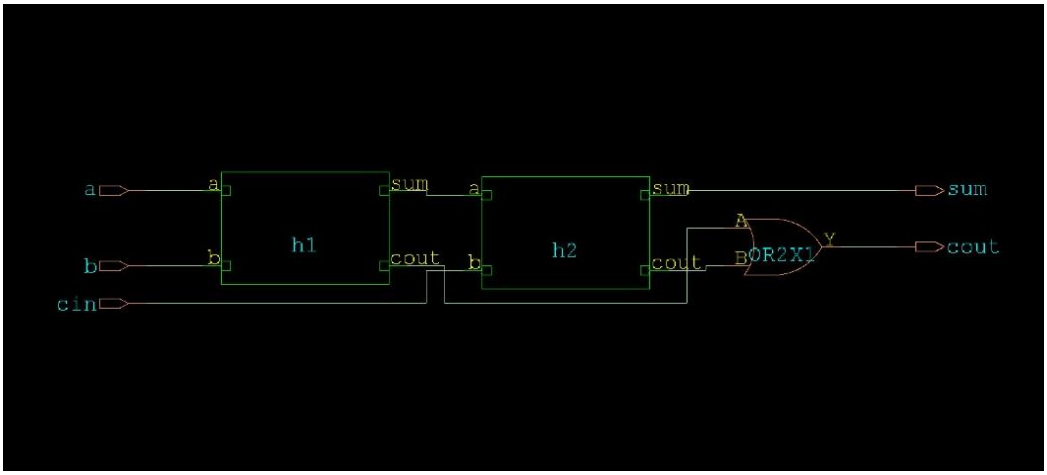
It is seen from the pie chart that in this instance csa3 consumes the maximum power.



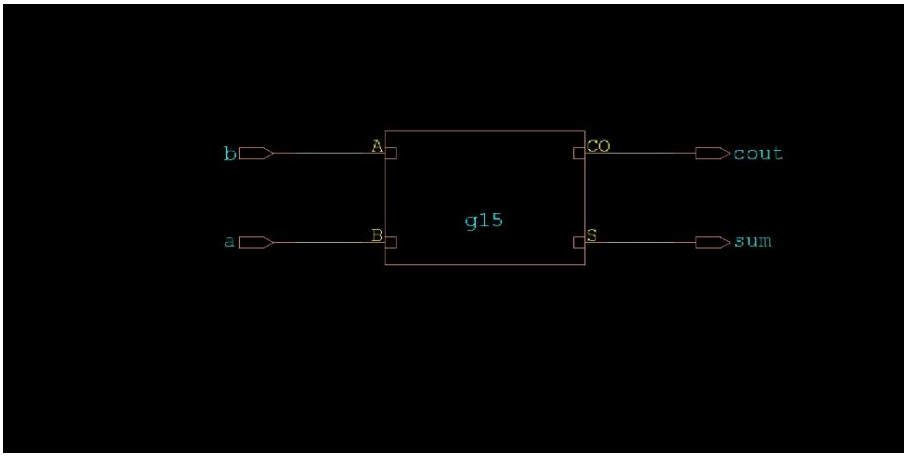
Here 4 bit carry skip adder is shown which consists of a ripple carry adder and propagation generation.



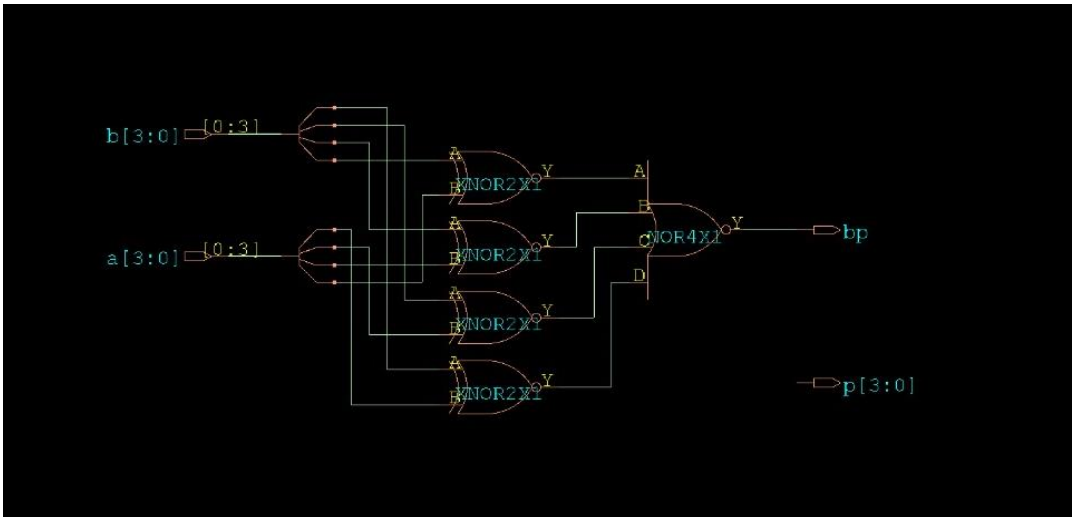
This is a 4 bit ripple carry adder consists of 4 full adders.



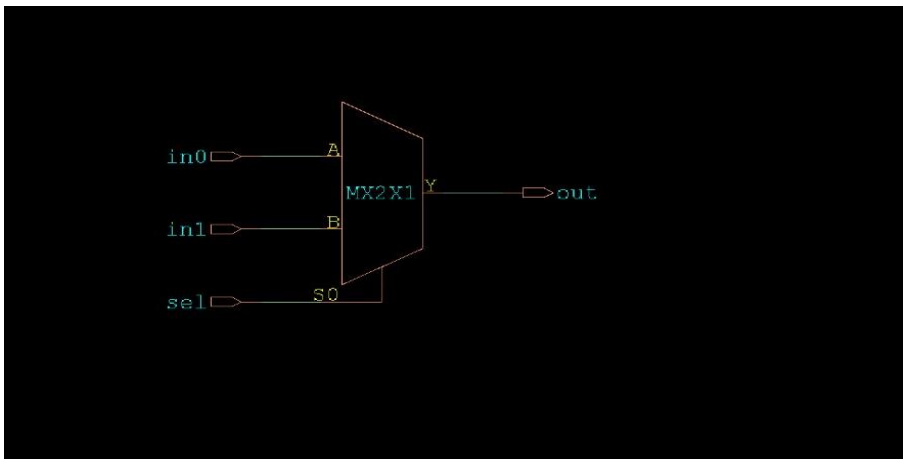
This is the 1 bit full adder circuit using 2 half adder circuit.



Circuit for 1 bit half adder is shown.



This is the circuit for propagation generation.



Here is the circuit for 2:1 MUX.

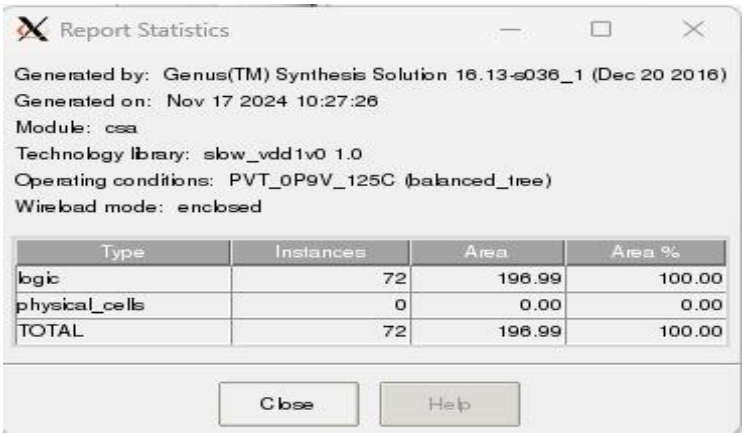


Fig: Report Statistics

This is the overall area report. There are 72 instances of cell and the total area is 196.99nm^2.

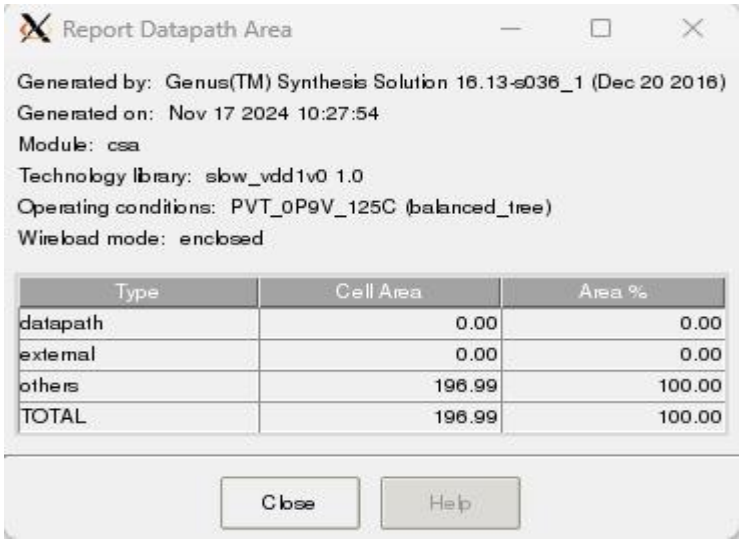


Fig: Report Datapath Area

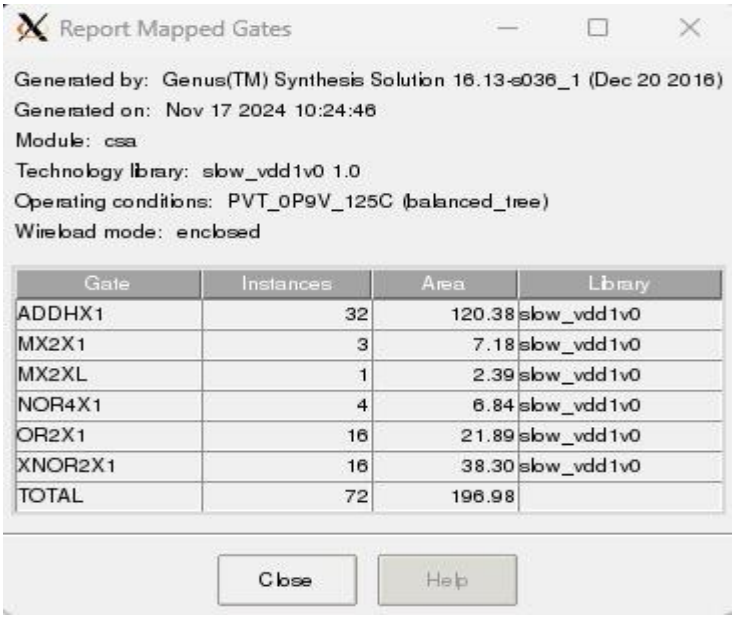


Fig: Report Mapped Gates

In the mapped gate report it is seen that the adder has the highest area compared to other gates and MUX.

Layout: In VLSI, the term "layout" describes the physical arrangement of a digital or analog circuit on a semiconductor chip. This layout encompasses the precise geometric placement of various components, including transistors, resistors, capacitors, and the intricate wiring that connects them. A well-designed layout not only delineates where each component will reside on the semiconductor substrate but also accounts for critical factors such as the electrical characteristics, thermal performance, and manufacturability of the circuit. The layout must adhere to specific design rules to ensure that the components are spaced adequately and that the interconnections can be effectively fabricated without defects. Furthermore, the layout serves as a bridge between the abstract circuit design and the actual physical implementation. It is a vital step in the VLSI design flow because it directly influences how the circuit will be fabricated on the silicon wafer. The layout process involves careful consideration of various parameters, including area optimization, power consumption, signal integrity, and scalability, making it essential for achieving reliable and efficient integrated circuits.

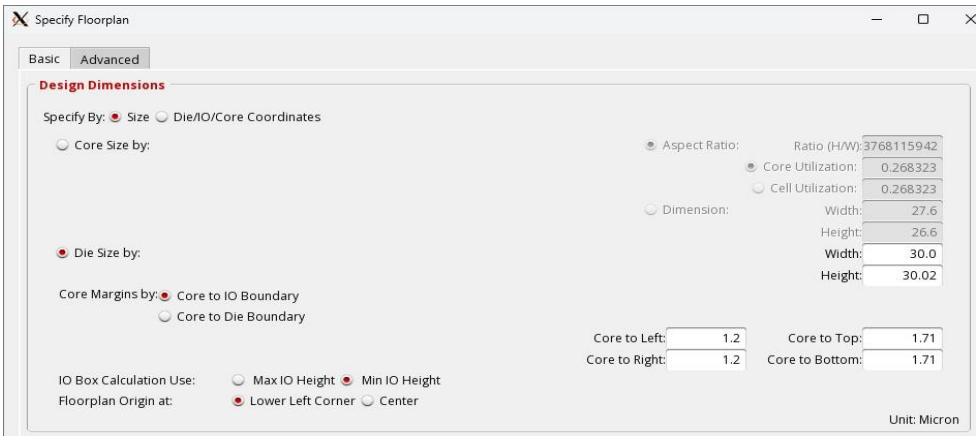


Fig: Floorplan Specification

This is the first floorplan we designed. The height and width of die size was both assumed to be 30nm. However the core to left and right was 1.2 micron and core to top and bottom was 1.71 micron.

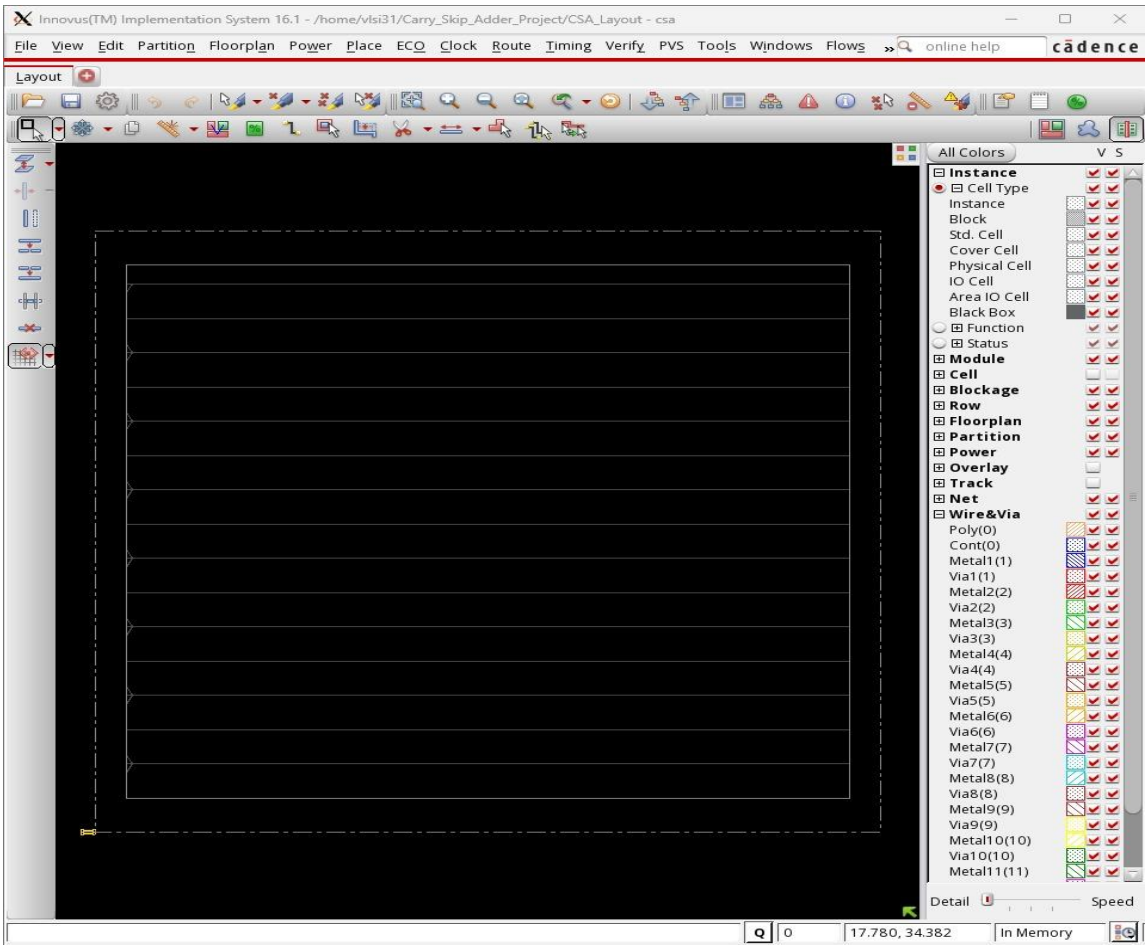


Fig: Innovus View after Floorplan Specification

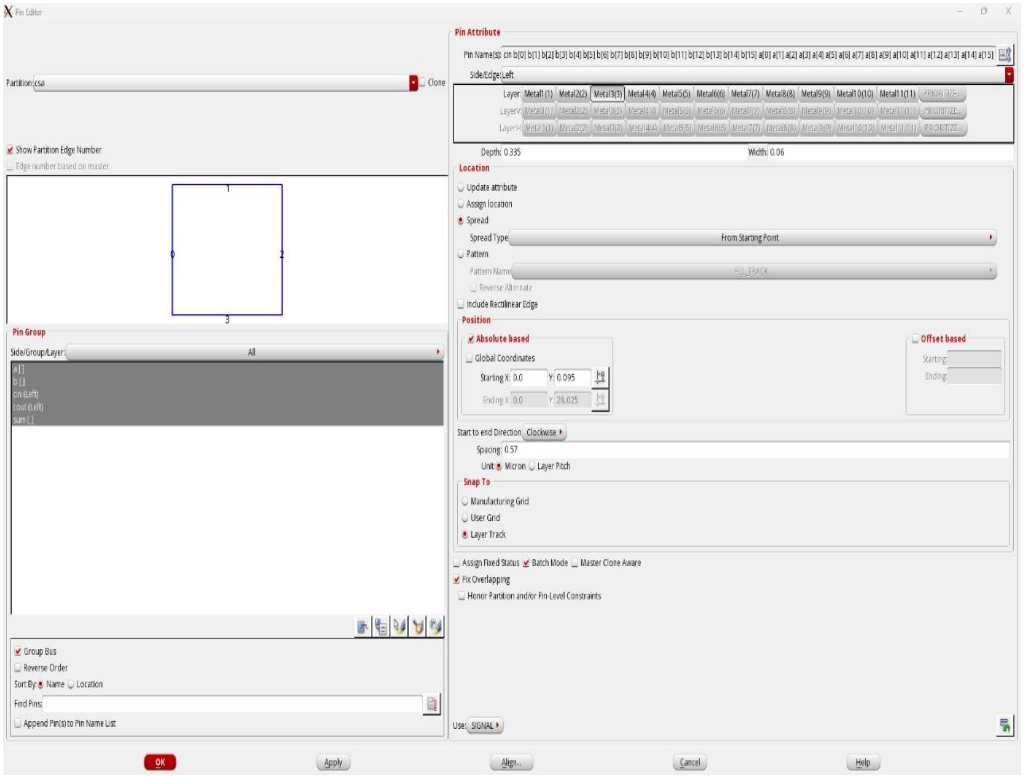


Fig: I/O Pin Specification

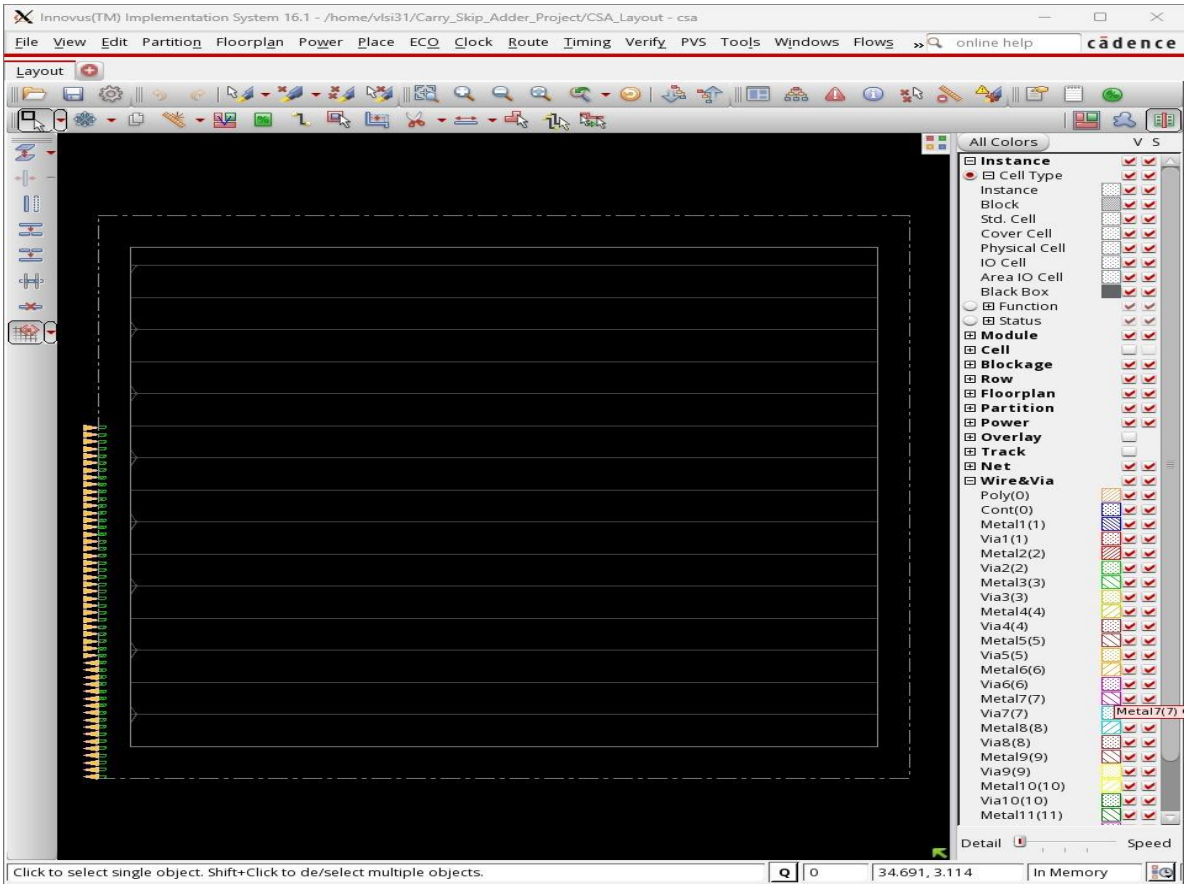


Fig: I/O Pin Addition

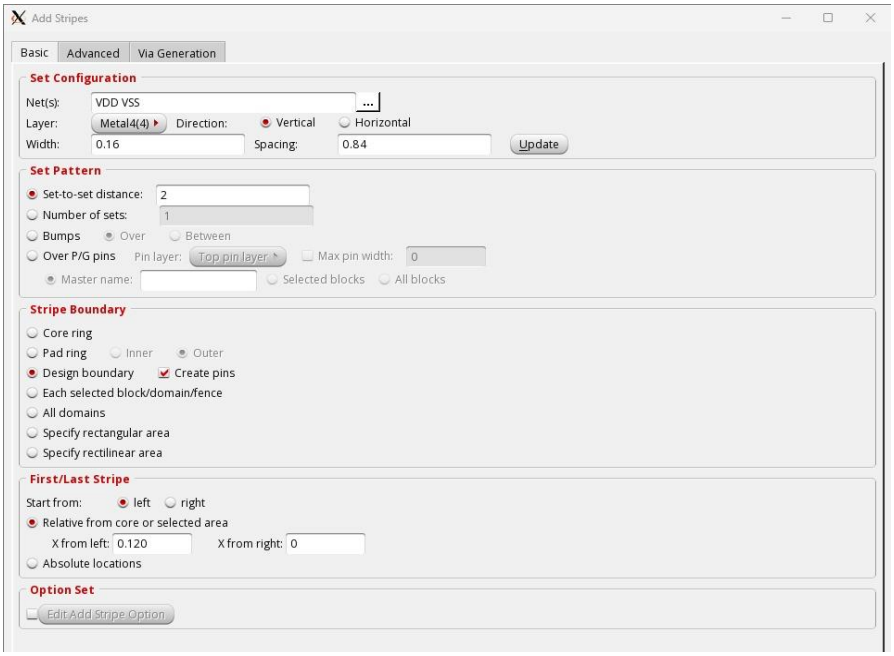


Fig: Stripe Specification

We specified width as 0.16 nm and spacing as 0.84 nm.

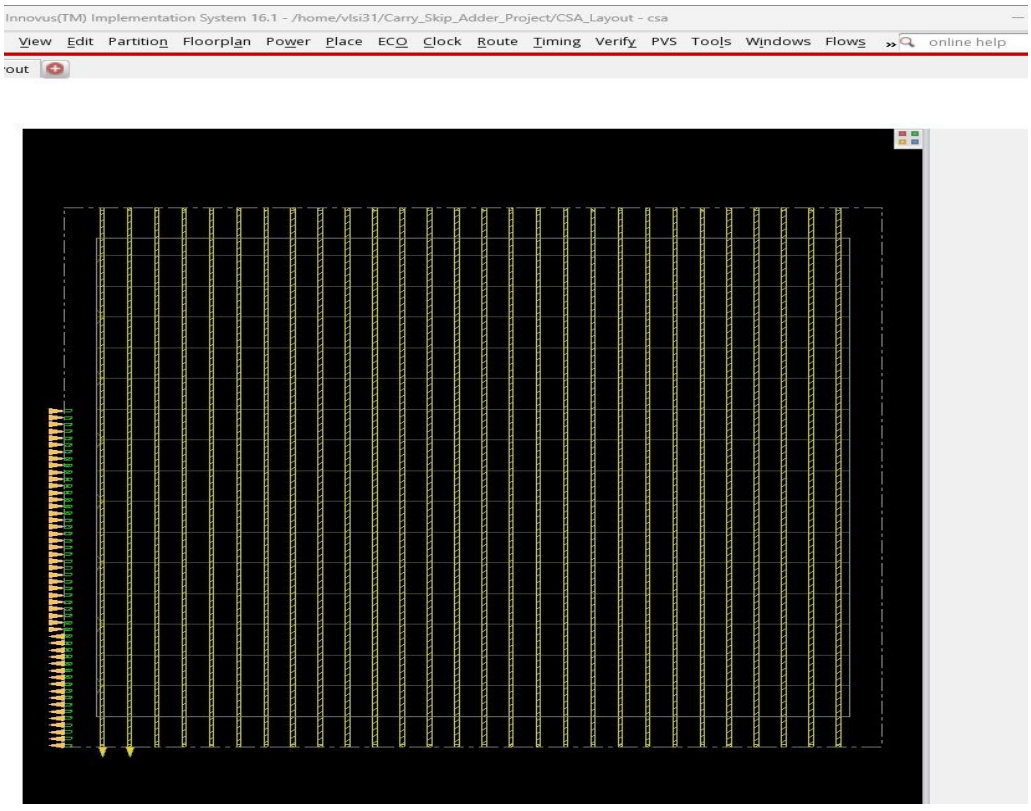


Fig: Supply of VDD and VSS

Route planning: Route planning in VLSI is a vital process that involves mapping out the physical pathways necessary for interconnecting various circuit components on a semiconductor chip. During this stage, interconnections are established using multiple metal layers and vias, which serve as vertical connections that link different horizontal layers of circuitry. These connections are dictated by the design netlist—a comprehensive representation of how each component should be interconnected. This process is particularly important as it plays a critical role in the overall physical design flow of the chip. Effective route planning not only ensures that all components are connected according to specifications but also has a direct impact on the chip's overall performance, physical footprint, and energy efficiency. By carefully considering the routes for interconnections, designers can optimize for signal integrity, reduce latency, and minimize power consumption, all of which are essential for producing high-performance integrated circuits.

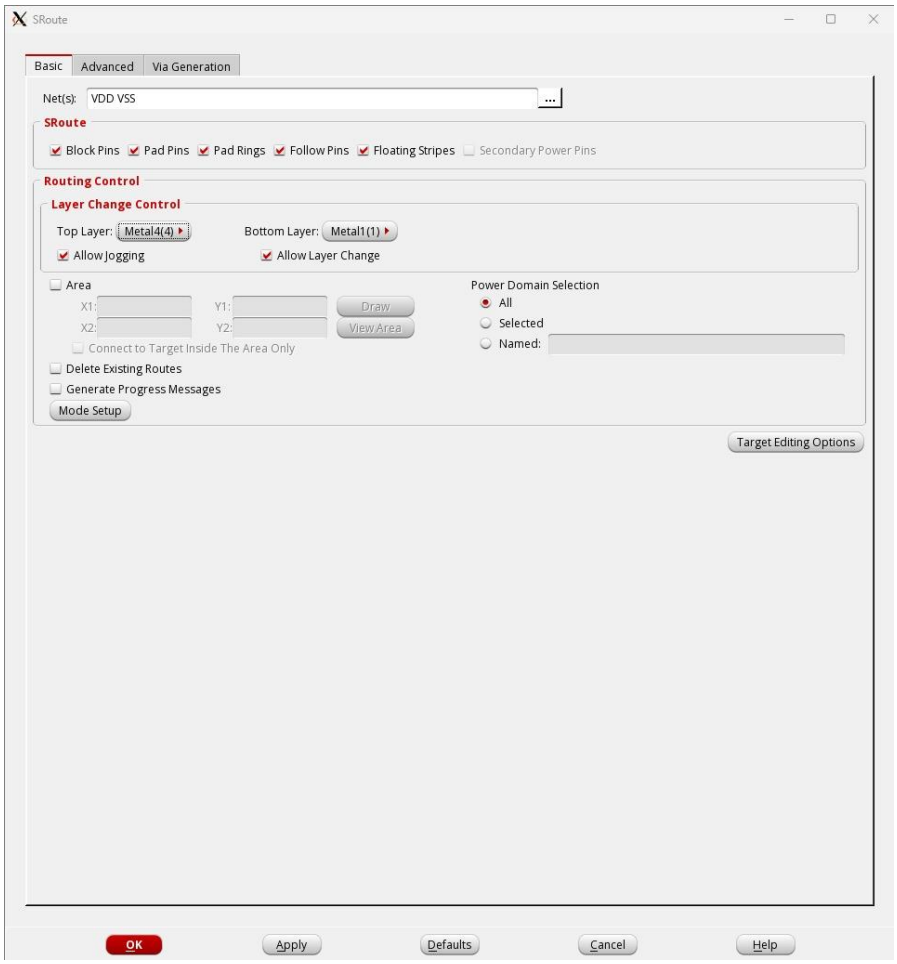


Fig: Metal Specification for special route

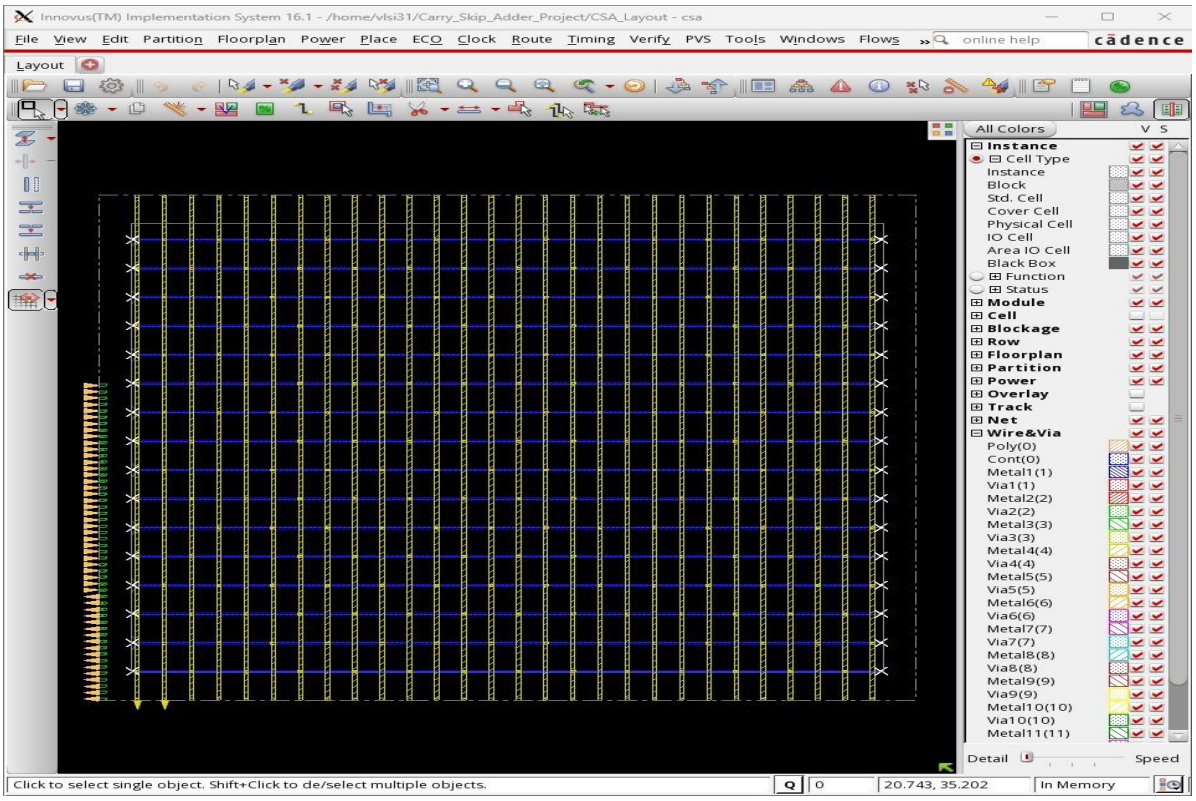


Fig: Special Route Innovus View

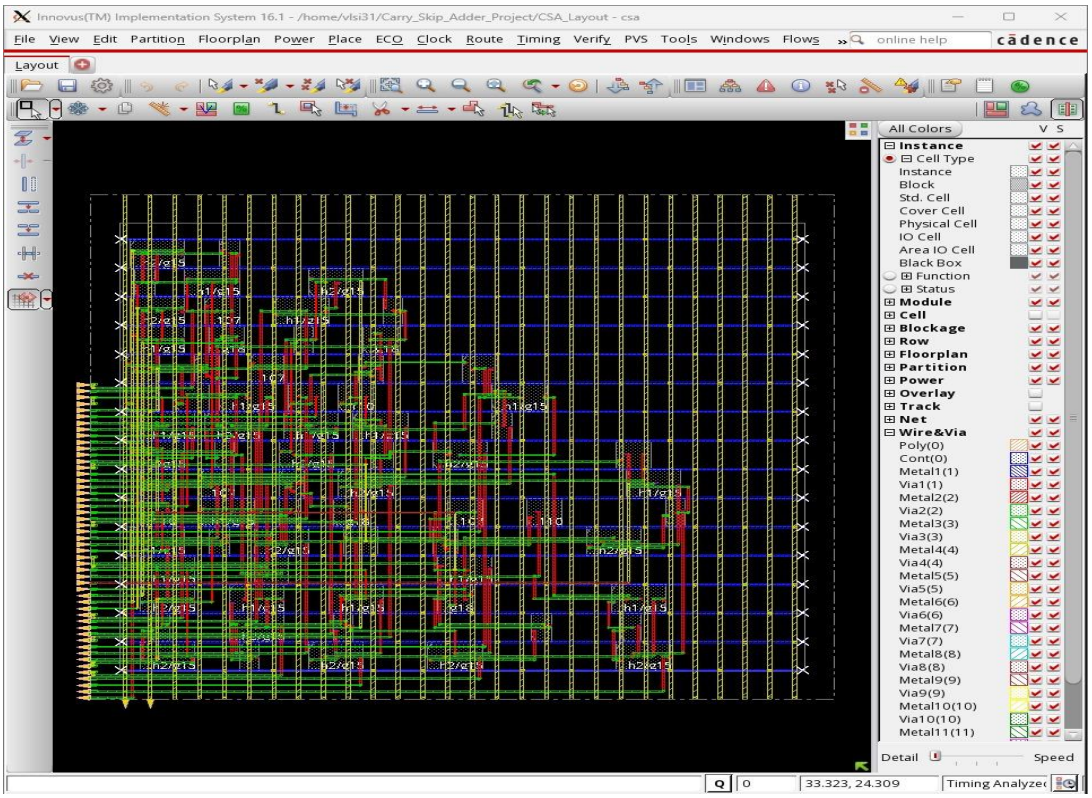


Fig: Layout Placement

Layout Clock Tree Synthesis Report:

```
------------------------------------------------------------
          timeDesign Summary
------------------------------------------------------------

Setup views included:
 func@BC_rcbest0.hold

+--------------------+---------+---------+
|     Setup mode     |   all   | default |
+--------------------+---------+---------+
|           WNS (ns):|  0.000  |  0.000  |
|           TNS (ns):|  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |
|          All Paths:|    0    |    0    |
+--------------------+---------+---------+

+---------------+-------------------------------+-----------------+
|               |              Real             |       Total     |
|    DRVs       +------------------+------------+-----------------|
|               |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms) |
+---------------+------------------+------------+-----------------+
|   max_cap     |      0 (0)       |   0.000    |      0 (0)      |
|   max_tran    |      0 (0)       |   0.000    |      0 (0)      |
|   max_fanout  |      0 (0)       |     0      |      0 (0)      |
|   max_length  |      0 (0)       |     0      |      0 (0)      |
+---------------+------------------+------------+-----------------+

Density: 27.826%
Routing Overflow: 0.00% H and 0.00% V
------------------------------------------------------------
Reported timing to dir ./timingReports
Total CPU time: 1.23 sec
Total Real time: 1.0 sec
Total Memory Usage: 1094.40625 Mbytes
```

Fig: Density and Timing Analysis

The density is only 27.826%. Total CPU time is 1.23 second.

```
RC Extraction called in multi-corner(1) mode.
RCMode: PreRoute
      RC Corner Indexes            0
Capacitance Scaling Factor   : 1.00000
Resistance Scaling Factor    : 1.00000
Clock Cap. Scaling Factor    : 1.00000
Clock Res. Scaling Factor    : 1.00000
Shrink Factor                : 1.00000
```

Fig: Scaling Analysis

Here capacitance and resistance scaling factor is 1.

```
Ending "Constraint file reading stats" (total cpu=0:00:00.0, real=0:00:0
1.0, peak res=294.3M, current mem=696.3M)
Current (total cpu=0:00:13.8, real=0:03:38, peak res=294.3M, current mem
=696.3M)
Total number of combinational cells: 327
Total number of sequential cells: 152
Total number of tristate cells: 10
Total number of level shifter cells: 0
Total number of power gating cells: 0
Total number of isolation cells: 0
Total number of power switch cells: 0
Total number of pulse generator cells: 0
Total number of always on buffers: 0
Total number of retention cells: 0
List of usable buffers: BUFX2 BUFX12 BUFX16 BUFX20 CLKBUFX2 BUFX3 BUFX4
BUFX6 BUFX8 CLKBUFX12 CLKBUFX16 CLKBUFX20 CLKBUFX3 CLKBUFX4 CLKBUFX6 CLK
BUFX8
Total number of usable buffers: 16
List of unusable buffers:
Total number of unusable buffers: 0
List of usable inverters: CLKINVX1 CLKINVX2 CLKINVX12 CLKINVX16 CLKINVX2
0 CLKINVX4 CLKINVX3 CLKINVX6 CLKINVX8 INVX1 INVX2 INVX12 INVX16 INVX20 I
NVXL INVX3 INVX4 INVX6 INVX8
Total number of usable inverters: 19
List of unusable inverters:
Total number of unusable inverters: 0
List of identified usable delay cells: DLY1X1 DLY1X4 DLY2X1 DLY2X4 DLY3X
1 DLY3X4 DLY4X1 DLY4X4
Total number of identified usable delay cells: 8
List of identified unusable delay cells:
Total number of identified unusable delay cells: 0
```

Fig: Number of Cells

This report shows the number of combinational cells, sequential cells and tristate cells. The total number of combinational, sequential and tristate cells are 327, 152 and 10.

Nano Route Planning:

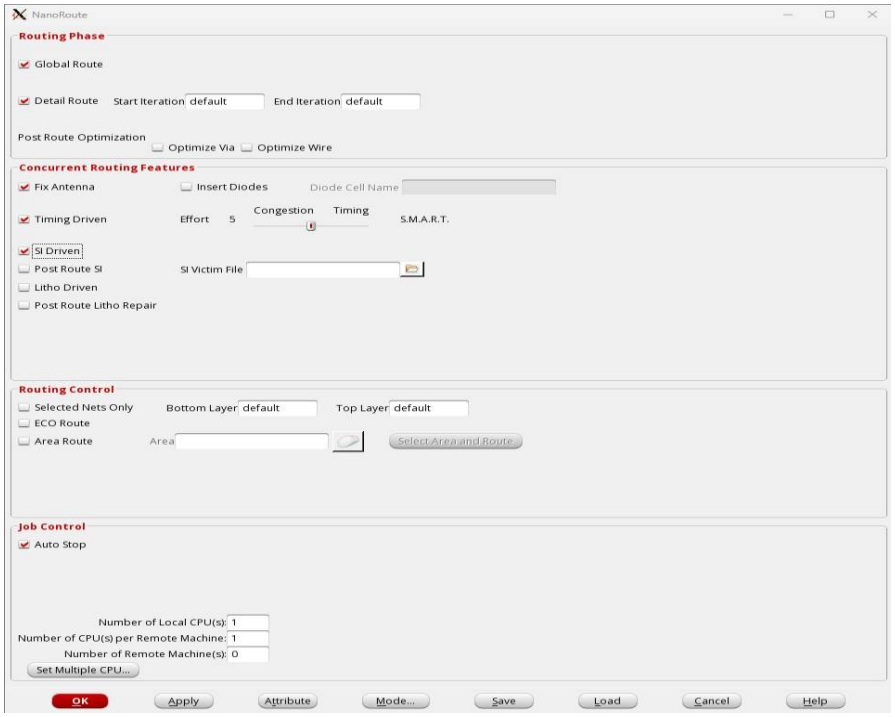


Fig: Nano route specification

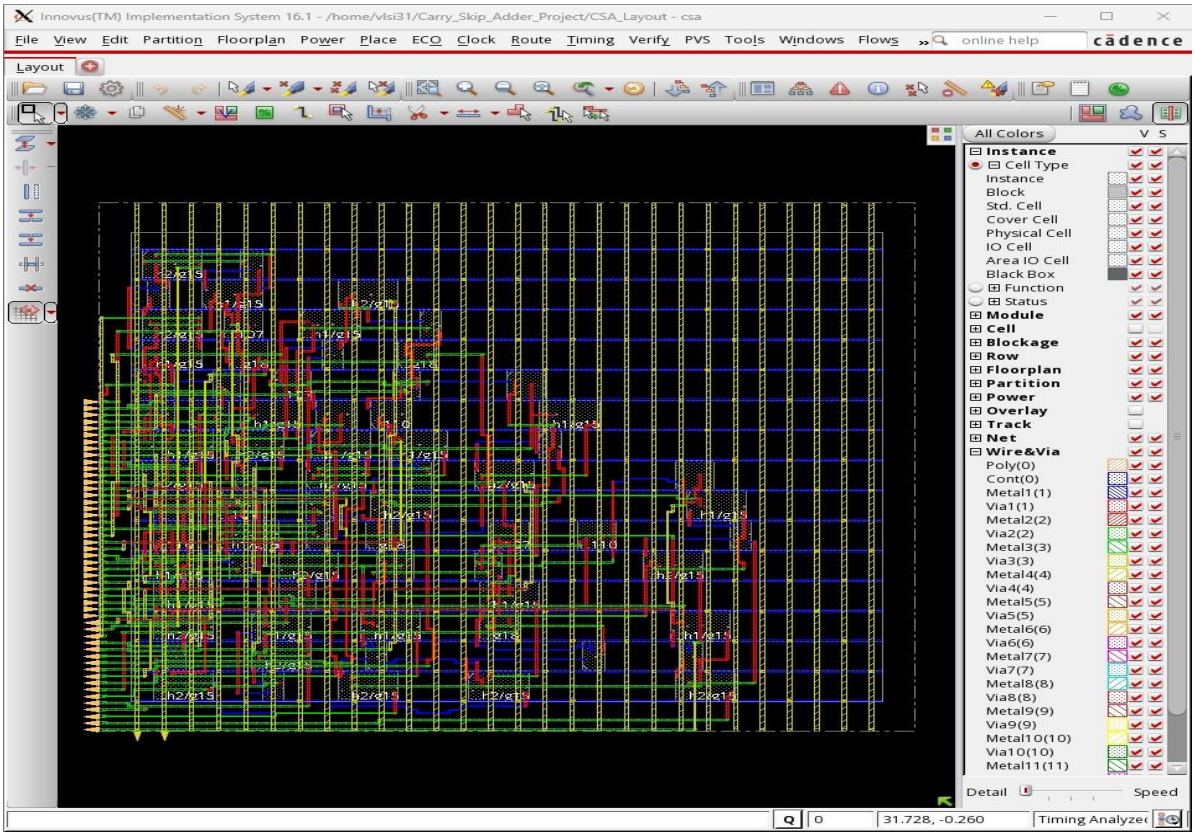


Fig: Innovus View of Nano Route Planning

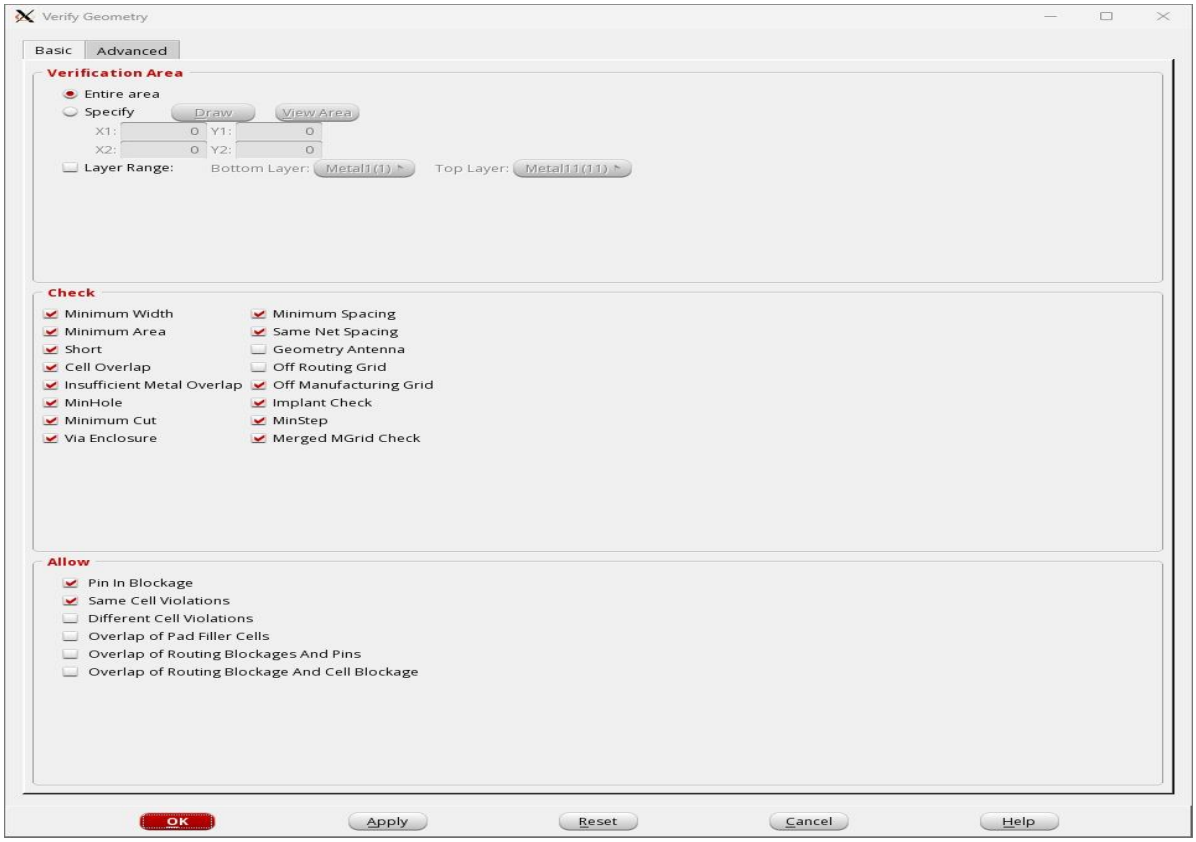
Post Route Timing & SI Analysis

```
------------------------------------------------------------------
          timeDesign Summary
------------------------------------------------------------------

Hold  views included:
 func@BC_rcbest0.hold

+--------------------+---------+---------+
|     Hold mode      |   all   | default |
+--------------------+---------+---------+
|           WNS (ns):|  0.000  |  0.000  |
|           TNS (ns):|  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |
|          All Paths:|    0    |    0    |
+--------------------+---------+---------+

Density: 27.826%
------------------------------------------------------------------
Reported timing to dir ./timingReports
Total CPU time: 3.19 sec
Total Real time: 3.0 sec
Total Memory Usage: 1185.441406 Mbytes
```

Fig: Post Route Timing and SI Analysis

After route timing and SI analysis the density is 27.826%.

Geometry Verification:




```
innovus 25> innovus 25>  *** Starting Verify Geometry (MEM: 1408.4) ***

  VERIFY GEOMETRY ..... Starting Verification
  VERIFY GEOMETRY ..... Initializing
  VERIFY GEOMETRY ..... Deleting Existing Violations
  VERIFY GEOMETRY ..... Creating Sub-Areas
                  ..... bin size: 1920
  VERIFY GEOMETRY ..... SubArea : 1 of 1
  VERIFY GEOMETRY ..... Cells          :  0 Viols.
  VERIFY GEOMETRY ..... SameNet        :  0 Viols.
  VERIFY GEOMETRY ..... Wiring         :  0 Viols.
  VERIFY GEOMETRY ..... Antenna        :  0 Viols.
  VERIFY GEOMETRY ..... Sub-Area : 1 complete 0 Viols. 0 Wrngs.
VG: elapsed time: 0.00
Begin Summary ...
  Cells       : 0
  SameNet     : 0
  Wiring      : 0
  Antenna     : 0
  Short       : 0
  Overlap     : 0
End Summary

  Verification Complete : 0 Viols.  0 Wrngs.

**********End: VERIFY GEOMETRY**********
 *** verify geometry (CPU: 0:00:00.1  MEM: 168.1M)
```

From geometry verification it is evident that our geometry verification shows no violations and warning.

Filler Addition: Filler addition in VLSI layout is a critical technique used during the design process of digital circuits. This practice involves the careful placement of filler cells—specialized circuit components—into the gaps that exist between standard cells in a digital design. Standard cells are the fundamental building blocks of a digital circuit, used to implement various logic functions. However, during the layout phase of VLSI design, certain spaces may arise between these cells due to the way they are arranged or aligned. These gaps, if left unfilled, can lead to several problems during the manufacturing process, including variations in the etching or deposition of materials, which can affect the yield and performance of the final product. Filler cells do not possess any logical functions like the standard cells; instead, their primary purpose is to satisfy design rules and maintain the integrity of the overall layout. They help to ensure uniformity in electrical characteristics across the chip by providing a consistent area of material. This uniformity is crucial for preventing issues such as signal degradation and electromagnetic interference. Moreover, the presence of filler cells enhances the structural robustness of the circuit, enabling better reliability during the fabrication process. By filling these voids, designers mitigate the risk of defects that can arise if certain areas of the chip are underdosed or overdosed with manufacturing materials. Overall, while filler cells may seem like a minor component in the grand scheme of digital design, their careful integration is essential for ensuring a functional, reliable, and manufacturable VLSI layout.

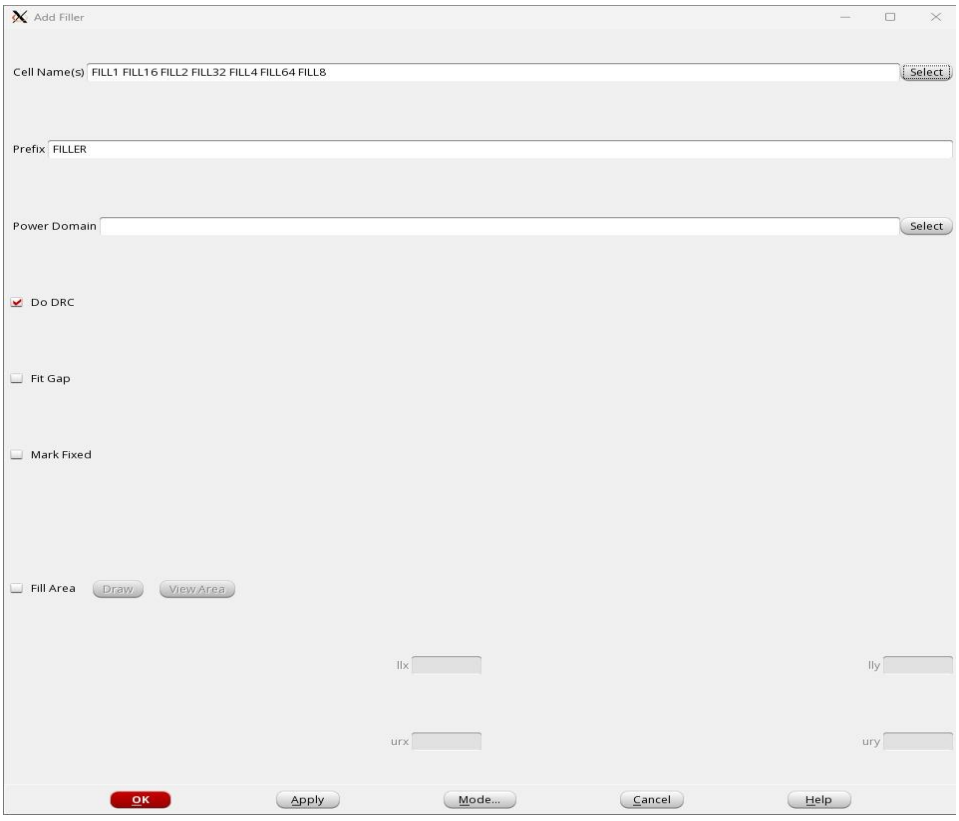


Fig: Filler specification

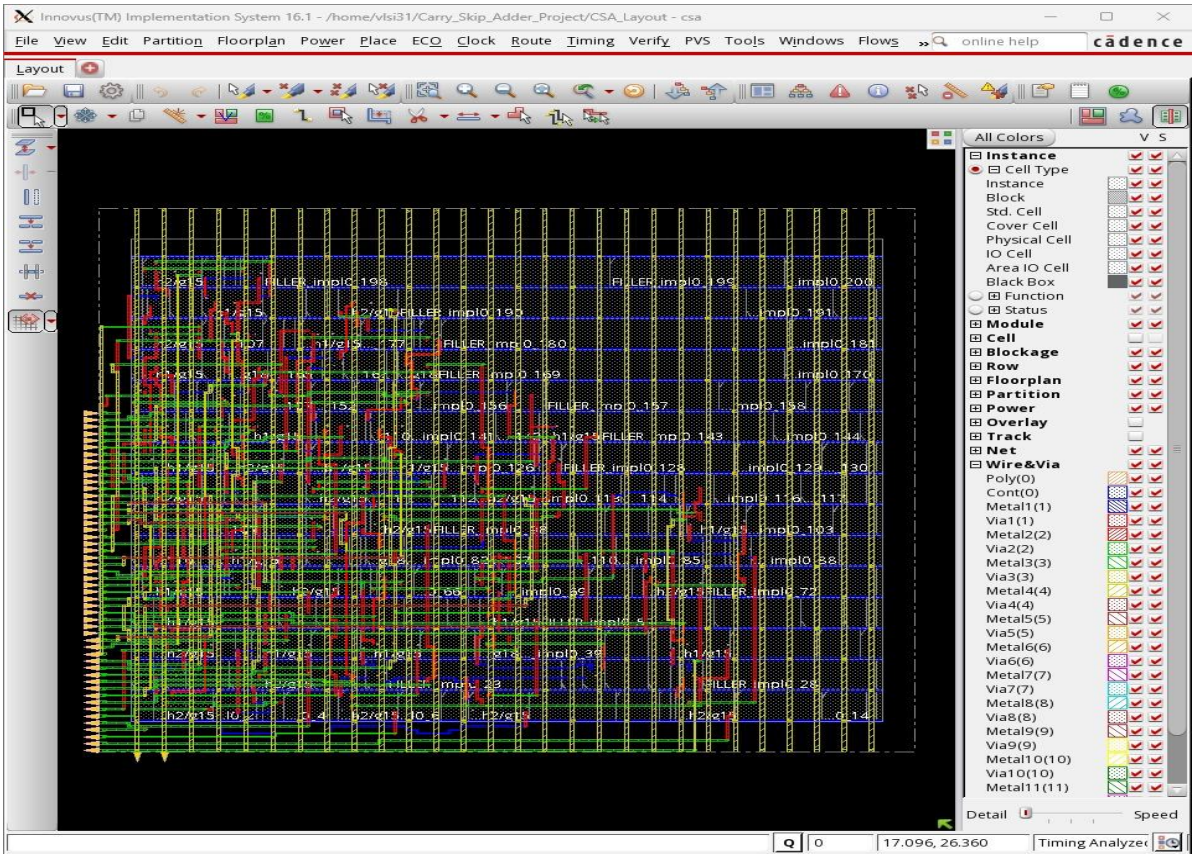


Fig: Innovus View after Filler Cell Addition

Metal Filling: Metal filling, often referred to as dummy metal insertion, is an essential technique in the design of VLSI (Very Large Scale Integration) layouts. This process involves strategically incorporating non-functional metal shapes into the chip architecture. The primary goal of these additions is to improve the uniformity of metal density across the entire chip surface. These supplementary metal structures, known as dummy fills or metal fills, do not carry out any electrical functions; instead, they serve a critical role in enhancing the manufacturing process. By creating a balanced distribution of metal across the layout, these fills help prevent potential issues such as warping or uneven etching during fabrication. Ultimately, the presence of these dummy fills is key to ensuring that the final product is of high quality, reliable, and meets the stringent specifications required for modern semiconductor devices. This technique involves the incorporation of non-functional metal shapes into the design to help achieve a more uniform distribution of metal density across the entire chip. These extra metal structures, known as dummy fills or metal fills, serve no electrical function; instead, their primary role is to optimize the chip's manufacturing process. By strategically placing these dummy fills, designers can minimize potential issues during fabrication, such as variations in metal thickness and stress that could impact the chip's performance and reliability. Thus, while they may appear superfluous, these metal fills are vital for ensuring the production of high-quality chips that meet rigorous standards in the semiconductor industry.

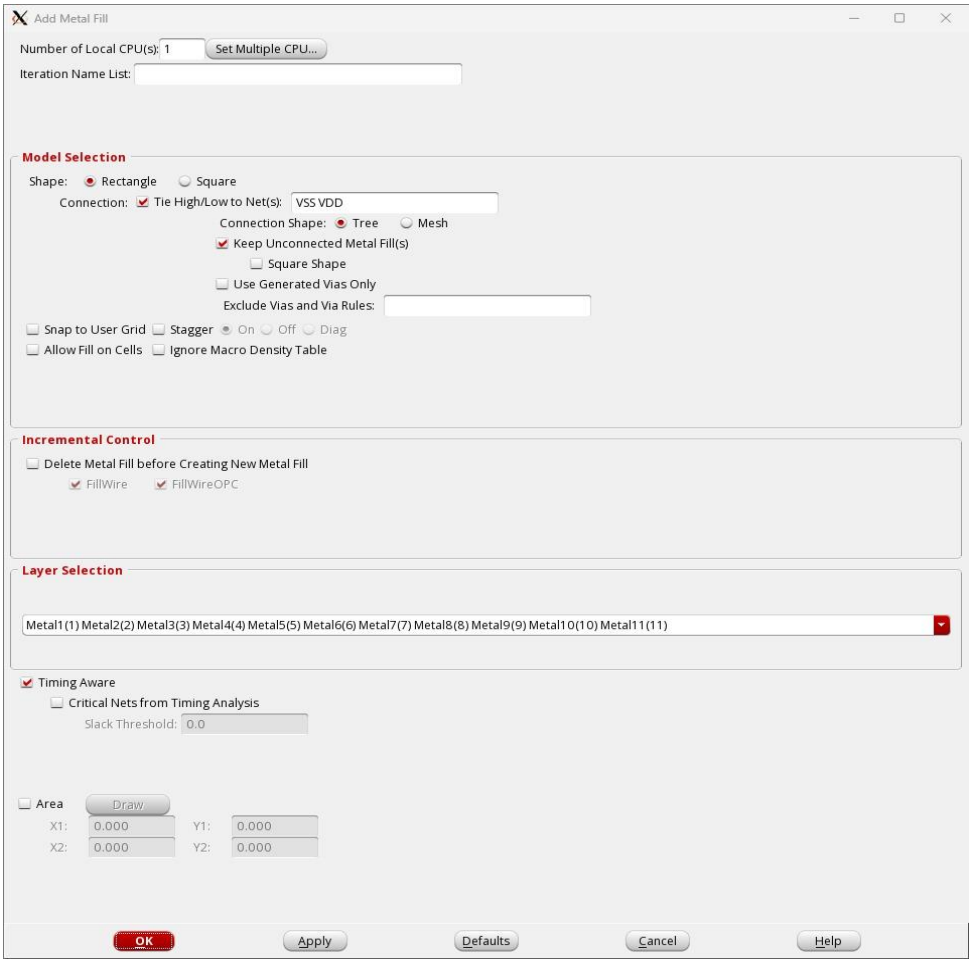


Fig: Metal Specification

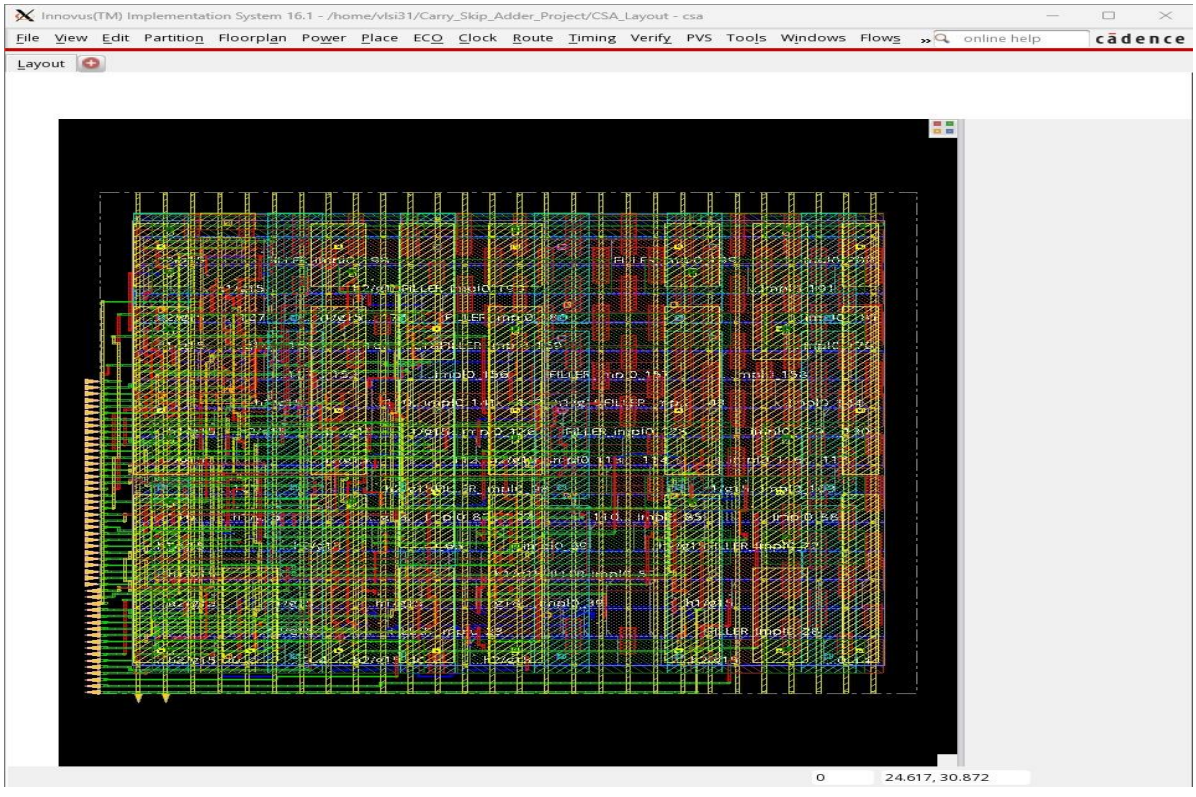


Fig: Innovus View after Metal Filling

GDS Data Exportation:

```
innovus 27> source /export_gds.tcl
couldn't read file "/export_gds.tcl": no such file or directory
innovus 28> source export_gds.tcl
Writing Netlist "csa.lvs_netlist.vg" ...
Pwr name (VDD).
Gnd name (VSS).
1 Pwr names and 1 Gnd names.
Creating all pg connections for top cell (csa).
Finding the highest version number among the merge files
Merge file: /home/cad/VLSI2Lab/Digital/library/gsclib045.gds has version
 number: 5

Parse map file...
Type 'man IMPOGDS-399' for more detail.
Type 'man IMPOGDS-399' for more detail.
Type 'man IMPOGDS-399' for more detail.
Type 'man IMPOGDS-399' for more detail.
Type 'man IMPOGDS-399' for more detail.
Type 'man IMPOGDS-399' for more detail.
Writing GDSII file ...
       ****** db unit per micron = 2000 ******
       ****** output gds2 file unit per micron = 2000 ******
       ****** unit scaling factor = 1 ******
```

```
Stream Out Information Processed for GDS version 5:
Units: 2000 DBU

Object                                  Count
----------------------------------------------
Instances                                 273

Ports/Pins                                104
    metal layer Metal3                     49
    metal layer Metal4                     55

Nets                                      790
    metal layer Metal1                     93
    metal layer Metal2                    405
    metal layer Metal3                    215
    metal layer Metal4                     77

    Via Instances                         465

Special Nets                               42
    metal layer Metal1                     15
    metal layer Metal4                     27

    Via Instances                         672

Metal Fills                                88
    metal layer Metal1                      2
    metal layer Metal2                     84
    metal layer Metal3                      2

    Via Instances                         236
```

Fig: Number of Metal Fills

```
Output for instance
Output for bump
Output for physical terminals
Output for logical terminals
Output for regular nets
Output for special nets and metal fills
Output for via structure generation
Statistics for GDS generated (version 5)
----------------------------------------
Stream Out Layer Mapping Information:
GDS Layer Number          GDS Layer Name
----------------------------------------
    233                          COMP
    234                       DIEAREA
    6                            Cont
    7                           Metal1
    8                             Via1
    9                           Metal2
    10                            Via2
    11                          Metal3
    30                            Via3
    31                          Metal4
    7                           Metal1
    9                           Metal2
    11                          Metal3
    31                          Metal4
```

```
Metal FillOPCs                    0

    Via Instances                 0

Text                            241
    metal layer Metal1           26
    metal layer Metal2           96
    metal layer Metal3           62
    metal layer Metal4           57


Blockages                         0


Custom Text                       0


Custom Box                        0

Trim Metal                        0

Merging with GDS libraries
Scanning GDS file /home/cad/VLSI2Lab/Digital/library/gsclib045.gds to re
gister cell name ......
Merging GDS file /home/cad/VLSI2Lab/Digital/library/gsclib045.gds ......
        ****** Merge file: /home/cad/VLSI2Lab/Digital/library/gsclib045.
gds has version number: 5.
        ****** Merge file: /home/cad/VLSI2Lab/Digital/library/gsclib045.
gds has units: 2000 per micron.
        ****** unit scaling factor = 1 ******
######Streamout is finished!
```

Fig: Name and number of Layer

Layout Verification:

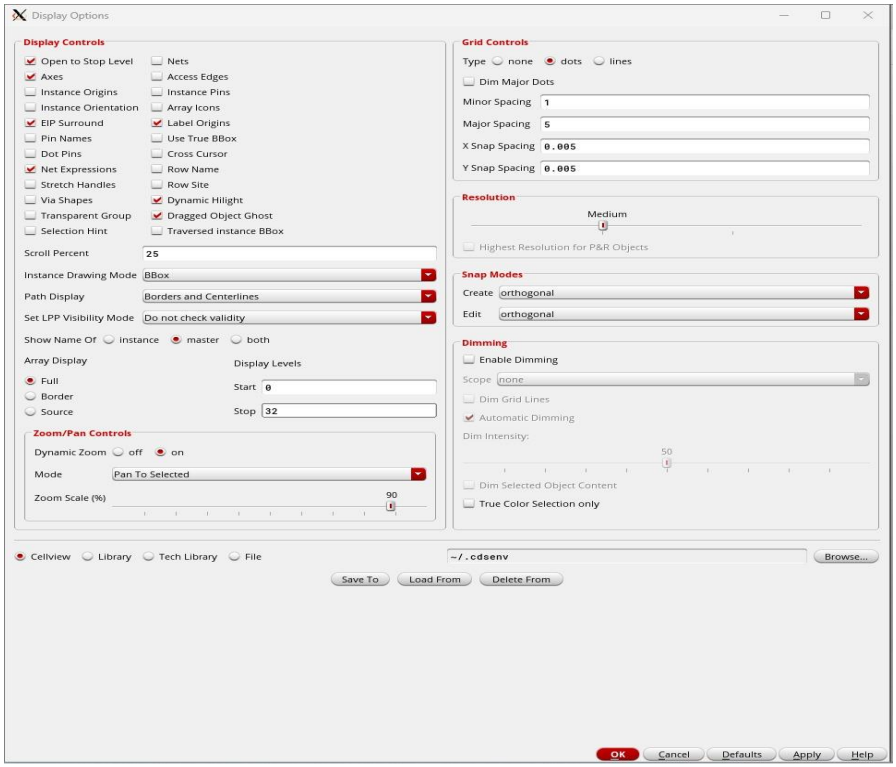


Fig: Display Specification

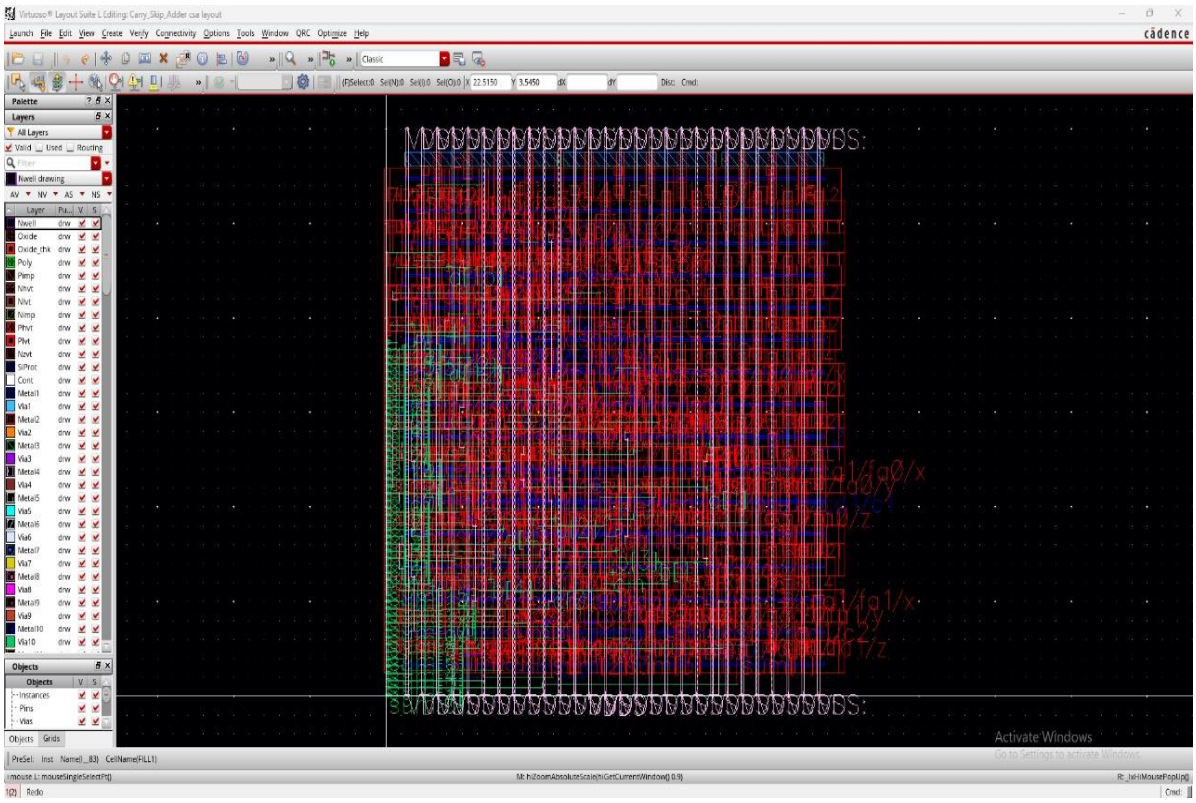


Fig: Virtuoso layout view

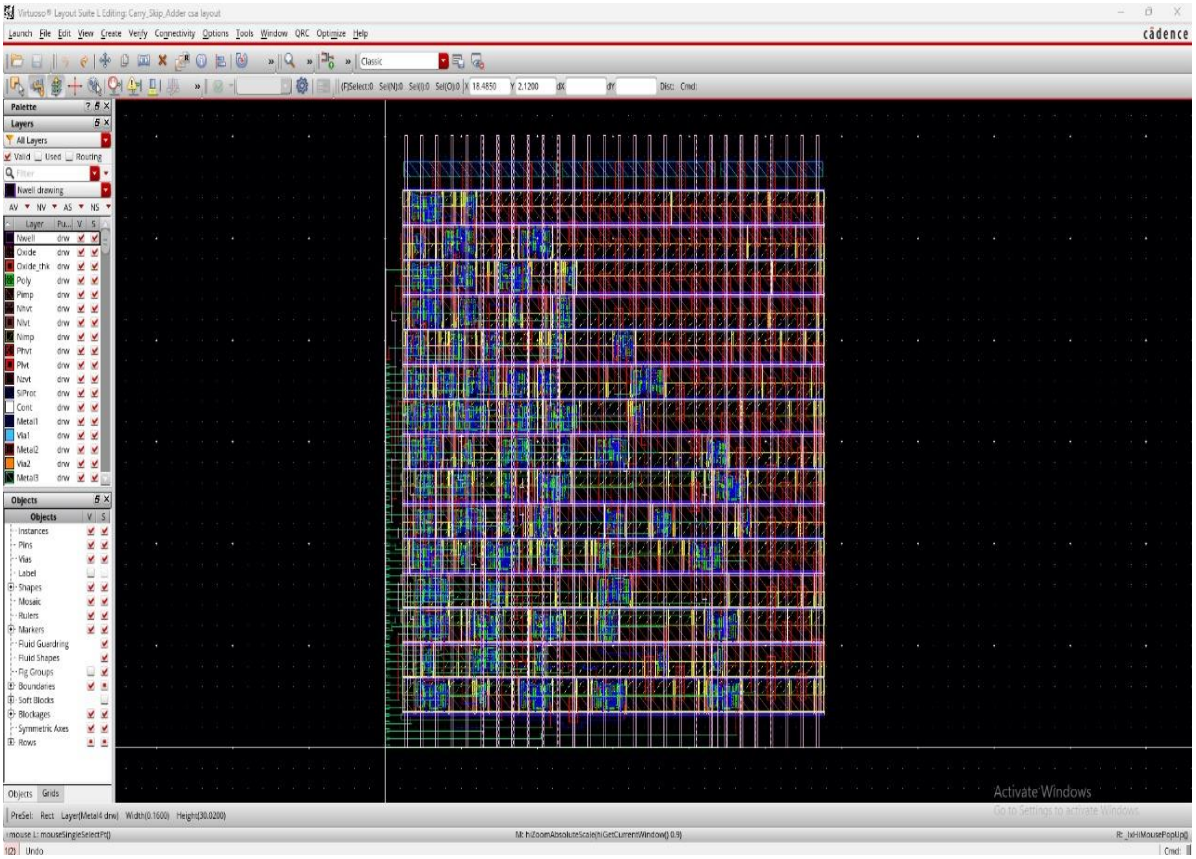


Fig: Physical View of Carry Skip Adder

DRC Check:

```
Total CPU Time                  : 1(s)
Total Real Time                 : 2(s)
Peak Memory Used                : 20(M)
Total Original Geometry         : 1824(15608)
Total DRC RuleChecks            : 562
Total DRC Results               : 0 (0)
Summary can be found in file csa.sum
ASCII report database is /home/vlsi31/Carry_Skip_Adder_Project/CSA_Layout/csa.drc_errors.ascii
Checking in all SoftShare licenses.


Design Rule Check Finished Normally. Tue Dec  3 15:36:35 2024
```

Fig: DRC Verification

So it is seen that there is no error in DRC verification.

Layout Optimization: We optimized layout and density by varying the die size area.

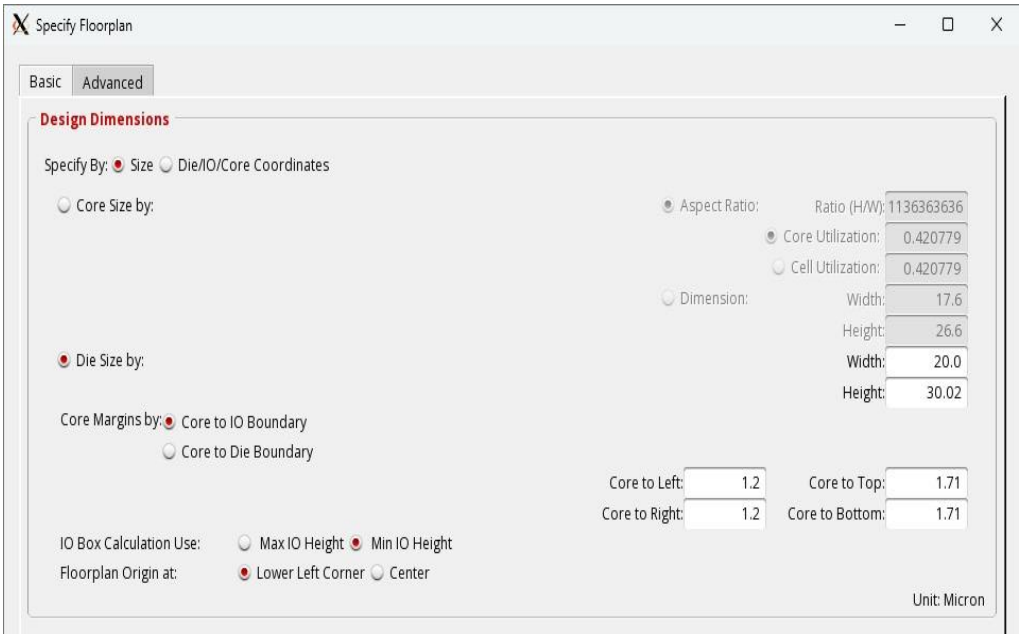


Fig: Previous Die Size

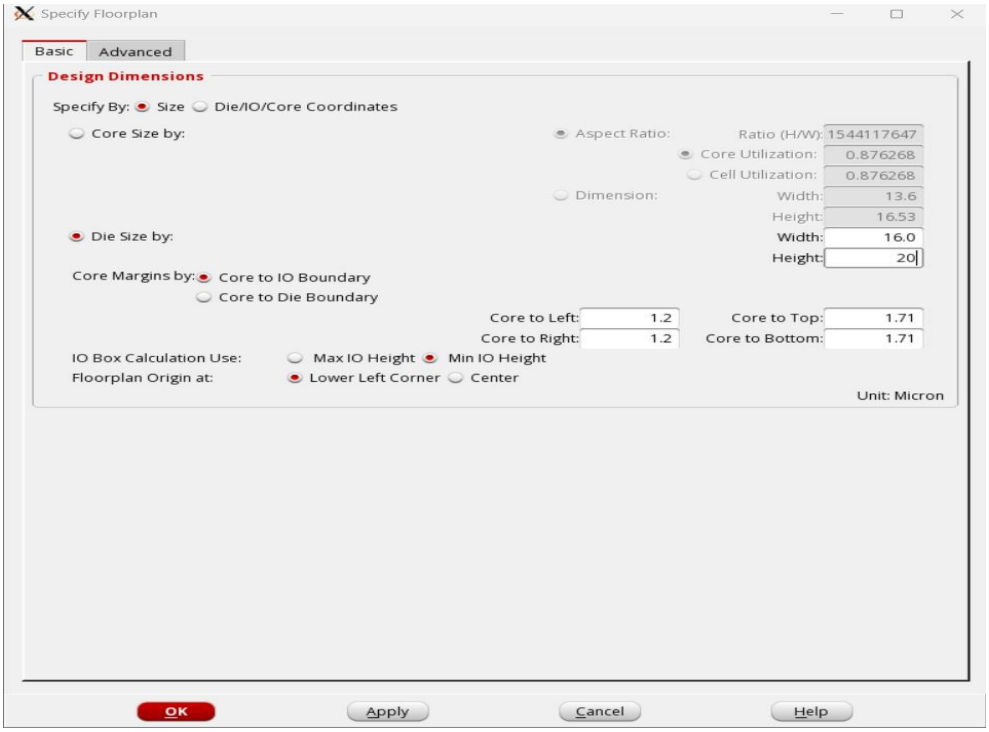


Fig: Optimized Die Size

We optimized layout by varying die size area. At first we used a die size with width as 20nm and height 30nm. Then we used another die size with width as 16nm and height 20nm. However we did not change core to left and right and core to top and bottom.

Spacing Variation:

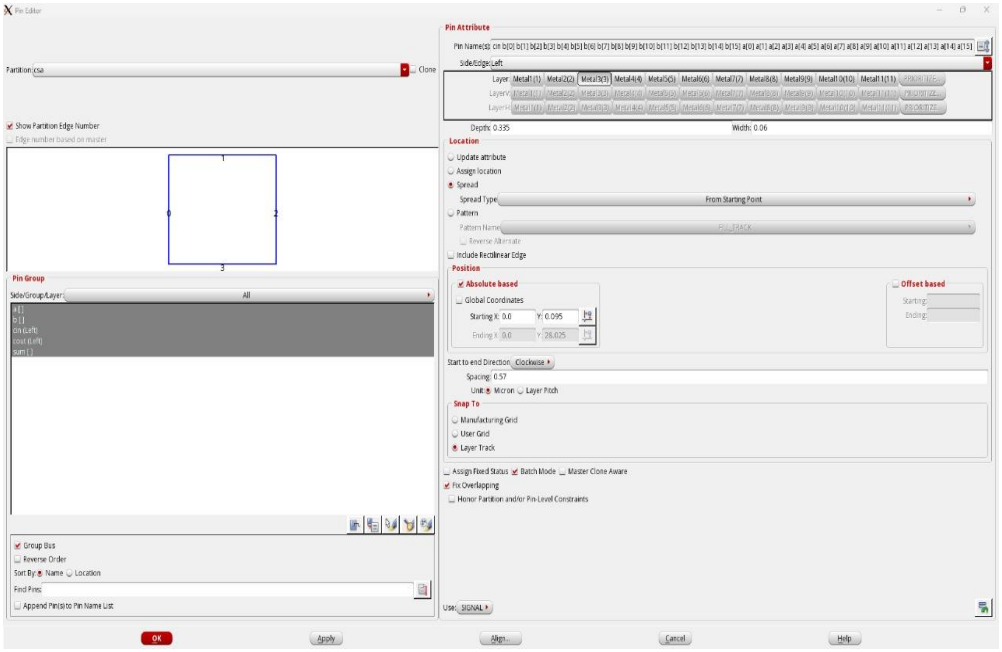


Fig: Previous Spacing

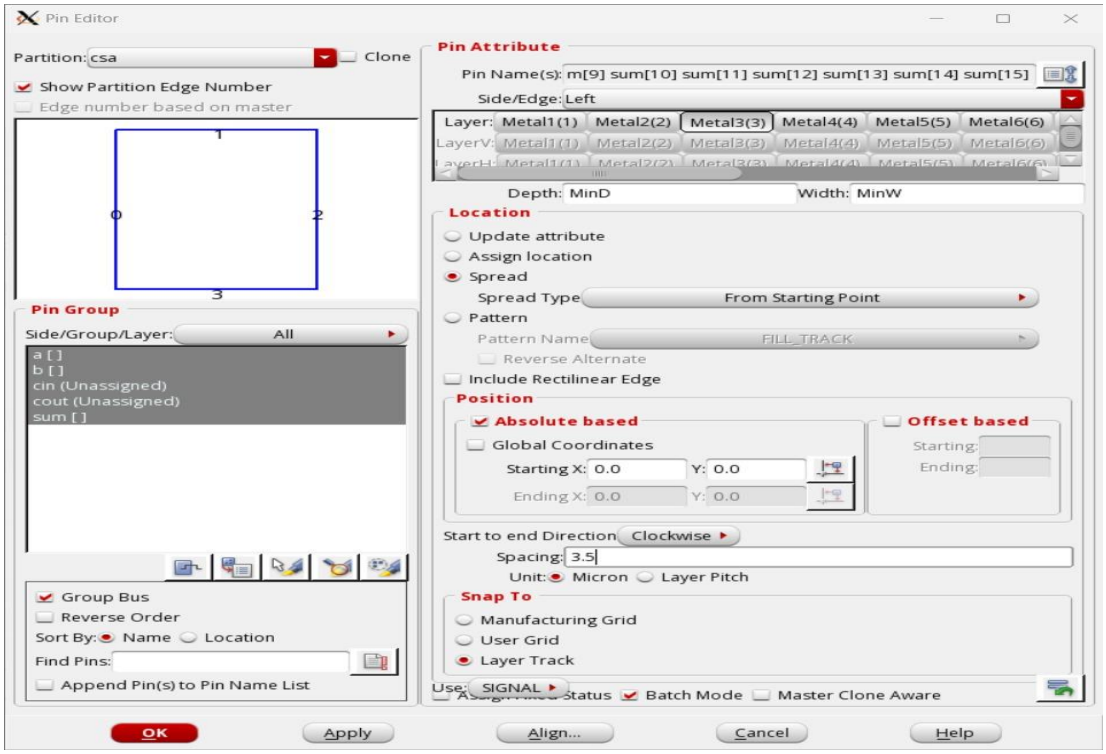


Fig: Optimized Spacing

Density Variation:

DRVs	Real		Total
	Nr nets(terms)	Worst Vio	Nr nets(terms)
max_cap	0 (0)	0.000	0 (0)
max_tran	0 (0)	0.000	0 (0)
max_fanout	0 (0)	0	0 (0)
max_length	0 (0)	0	0 (0)

Density: 43.636%
Routing Overflow: 0.00% H and 0.00% V

Fig: Previous Density

DRVs	Real		Total
	Nr nets(terms)	Worst Vio	Nr nets(terms)
max_cap	0 (0)	0.000	0 (0)
max_tran	0 (0)	0.000	0 (0)
max_fanout	0 (0)	0	0 (0)
max_length	0 (0)	0	0 (0)

Density: 94.118%
Routing Overflow: 0.00% H and 0.00% V

Fig: Optimized Density

The previously designed chip has 43.636% density while the optimized density is 94.118%.

Resource Analysis: Resource analysis in Very Large Scale Integration (VLSI) plays a critical role in the successful design and implementation of microchips. This process entails a thorough evaluation and management of various essential resources required for chip design. Among these resources are routing tracks, which are the pathways that connect different components on the chip; cell area, which refers to the space occupied by functional blocks; and power supply, which is vital for powering the chip's operations. Additionally, it encompasses other physical and computational elements that significantly impact the design's ability to meet functional requirements, timing constraints, and manufacturability standards. By carefully analyzing these resources, designers can optimize performance and ensure that the final product effectively meets its intended specifications.

```
[11/17 12:45:38    149] #  Resource Analysis:
[11/17 12:45:38    149] #
[11/17 12:45:38    149] #               Routing  #Avail      #Track     #Total     %Gcell
[11/17 12:45:38    149] #  Layer      Direction   Track      Blocked      Gcell    Blocked
[11/17 12:45:38    149] #  ---------------------------------------------------------------
[11/17 12:45:38    149] #  Metal 1        H         158          0          77     38.96%
[11/17 12:45:38    149] #  Metal 2        V         100          0          77      0.00%
[11/17 12:45:38    149] #  Metal 3        H         158          0          77      0.00%
[11/17 12:45:38    149] #  Metal 4        V         100          0          77      0.00%
[11/17 12:45:38    149] #  Metal 5        H         158          0          77      0.00%
[11/17 12:45:38    149] #  Metal 6        V         100          0          77      0.00%
[11/17 12:45:38    149] #  Metal 7        H         158          0          77      0.00%
[11/17 12:45:38    149] #  Metal 8        V         100          0          77      0.00%
[11/17 12:45:38    149] #  Metal 9        H         158          0          77      0.00%
[11/17 12:45:38    149] #  Metal 10       V          40          0          77      0.00%
[11/17 12:45:38    149] #  Metal 11       H          63          0          77      0.00%
[11/17 12:45:38    149] #  ---------------------------------------------------------------
[11/17 12:45:38    149] #  Total                   1293       0.00%         847      3.54%
```

Fig: Resource for Previous Design

Here total number of avail track is 1293 and total number of Gcell is 847. %Gcell is 3.54%

```
[12/05 22:52:19    24] #  Resource Analysis:
[12/05 22:52:19    24] #
[12/05 22:52:19    24] #               Routing  #Avail     #Track     #Total     %Gcell
[12/05 22:52:19    24] #  Layer      Direction   Track     Blocked     Gcell    Blocked
[12/05 22:52:19    24] #  ---------------------------------------------------------
[12/05 22:52:19    24] #  Metal 1        H        105          0          42    66.67%
[12/05 22:52:19    24] #  Metal 2        V         80          0          42     0.00%
[12/05 22:52:19    24] #  Metal 3        H        105          0          42     0.00%
[12/05 22:52:19    24] #  Metal 4        V         80          0          42     0.00%
[12/05 22:52:19    24] #  Metal 5        H        105          0          42     0.00%
[12/05 22:52:19    24] #  Metal 6        V         80          0          42     0.00%
[12/05 22:52:19    24] #  Metal 7        H        105          0          42     0.00%
[12/05 22:52:19    24] #  Metal 8        V         80          0          42     0.00%
[12/05 22:52:19    24] #  Metal 9        H        105          0          42     0.00%
[12/05 22:52:19    24] #  Metal 10       V         32          0          42     0.00%
[12/05 22:52:19    24] #  Metal 11       H         42          0          42     0.00%
[12/05 22:52:19    24] #  ---------------------------------------------------------
[12/05 22:52:19    24] #  Total                   919       0.00%        462     6.06%
```

Fig: Resource for Optimized Design

Here it is evident that the total number of available track is reduced but %GCell has increased to double.

Congestion Analysis: Congestion analysis in Very-Large-Scale Integration (VLSI) is a detailed evaluation process that focuses on identifying specific regions within a chip design where the available routing resources, including metal layers and vias, fall short of connecting all the necessary electrical nets. This analysis is essential in the physical design workflow, as it plays a pivotal role in ensuring that the integrated circuit can be routed efficiently. By pinpointing areas of potential traffic congestion, designers can make informed decisions to address timing, power consumption, and manufacturability issues, ultimately leading to a more reliable and high-performing chip. This proactive approach helps to avoid routing failures, ensuring that the final design meets all operational specifications and constraints.

```
[11/17 12:45:38    149] #  Congestion Analysis: (blocked Gcells are excluded)
[11/17 12:45:38    149] #
[11/17 12:45:38    149] #                OverCon       OverCon       OverCon
[11/17 12:45:38    149] #                 #Gcell        #Gcell        #Gcell    %Gcell
[11/17 12:45:38    149] #    Layer          (1)           (2)           (3)   OverCon
[11/17 12:45:38    149] #  ------------------------------------------------------------
[11/17 12:45:38    149] #   Metal 1     0(0.00%)      0(0.00%)      0(0.00%)   (0.00%)
[11/17 12:45:38    149] #   Metal 2     1(1.30%)      3(3.90%)      1(1.30%)   (6.49%)
[11/17 12:45:38    149] #   Metal 3     0(0.00%)      0(0.00%)      0(0.00%)   (0.00%)
[11/17 12:45:38    149] #   Metal 4     0(0.00%)      0(0.00%)      0(0.00%)   (0.00%)
[11/17 12:45:38    149] #   Metal 5     0(0.00%)      0(0.00%)      0(0.00%)   (0.00%)
[11/17 12:45:38    149] #   Metal 6     0(0.00%)      0(0.00%)      0(0.00%)   (0.00%)
[11/17 12:45:38    149] #   Metal 7     0(0.00%)      0(0.00%)      0(0.00%)   (0.00%)
[11/17 12:45:38    149] #   Metal 8     0(0.00%)      0(0.00%)      0(0.00%)   (0.00%)
[11/17 12:45:38    149] #   Metal 9     0(0.00%)      0(0.00%)      0(0.00%)   (0.00%)
[11/17 12:45:38    149] #  Metal 10     0(0.00%)      0(0.00%)      0(0.00%)   (0.00%)
[11/17 12:45:38    149] #  Metal 11     0(0.00%)      0(0.00%)      0(0.00%)   (0.00%)
[11/17 12:45:38    149] #  ------------------------------------------------------------
[11/17 12:45:38    149] #     Total     1(0.12%)      3(0.36%)      1(0.12%)   (0.59%)
[11/17 12:45:38    149] #
[11/17 12:45:38    149] #  The worst congested Gcell overcon (routing demand over resource in number of tracks) = 3
[11/17 12:45:38    149] #  Overflow after GR: 0.00% H + 1.30% V
```

Fig: Congestion for Previous Design

```
[12/05 22:52:19     24] #  Congestion Analysis: (blocked Gcells are excluded)
[12/05 22:52:19     24] #
[12/05 22:52:19     24] #                OverCon       OverCon
[12/05 22:52:19     24] #                 #Gcell        #Gcell    %Gcell
[12/05 22:52:19     24] #    Layer          (1)           (2)   OverCon
[12/05 22:52:19     24] #  --------------------------------------------
[12/05 22:52:19     24] #   Metal 1     0(0.00%)      0(0.00%)   (0.00%)
[12/05 22:52:19     24] #   Metal 2     8(19.0%)      1(2.38%)   (21.4%)
[12/05 22:52:19     24] #   Metal 3     0(0.00%)      0(0.00%)   (0.00%)
[12/05 22:52:19     24] #   Metal 4     0(0.00%)      0(0.00%)   (0.00%)
[12/05 22:52:19     24] #   Metal 5     0(0.00%)      0(0.00%)   (0.00%)
[12/05 22:52:19     24] #   Metal 6     0(0.00%)      0(0.00%)   (0.00%)
[12/05 22:52:19     24] #   Metal 7     0(0.00%)      0(0.00%)   (0.00%)
[12/05 22:52:19     24] #   Metal 8     0(0.00%)      0(0.00%)   (0.00%)
[12/05 22:52:19     24] #   Metal 9     0(0.00%)      0(0.00%)   (0.00%)
[12/05 22:52:19     24] #  Metal 10     0(0.00%)      0(0.00%)   (0.00%)
[12/05 22:52:19     24] #  Metal 11     0(0.00%)      0(0.00%)   (0.00%)
[12/05 22:52:19     24] #  --------------------------------------------
[12/05 22:52:19     24] #     Total     8(1.80%)      1(0.22%)   (2.02%)
[12/05 22:52:19     24] #
[12/05 22:52:19     24] #  The worst congested Gcell overcon (routing demand over resource in number of
tracks) = 2
[12/05 22:52:19     24] #  Overflow after GR: 0.00% H + 4.29% V
```

Fig: Congestion for Optimized Design

%Gcell overcon has increased to double for optimized design.

Optimization: Number of Metal Layers and Source to Sink Distance

```
[11/17 12:45:38    149] #----------------------
[11/17 12:45:38    149] #  Metal 1         294
[11/17 12:45:38    149] #  Metal 2         204
[11/17 12:45:38    149] #  Metal 3          33
[11/17 12:45:38    149] #  Metal 4           6
[11/17 12:45:38    149] #----------------------
[11/17 12:45:38    149] #                  537
[11/17 12:45:38    149] #
[11/17 12:45:38    149] #Total number of involved regular nets 27
[11/17 12:45:38    149] #Maximum src to sink distance  35.4
[11/17 12:45:38    149] #Average of max src_to_sink distance  18.7
[11/17 12:45:38    149] #Average of ave src_to_sink distance  16.8
[11/17 12:45:38    149] #Max overcon = 3 tracks.
[11/17 12:45:38    149] #Total overcon = 0.59%.
[11/17 12:45:38    149] #Worst layer Gcell overcon rate = 0.00%.
```

Fig: Previous Design

```
[12/05 22:52:19     24] #----------------------
[12/05 22:52:19     24] #  Metal 1         238
[12/05 22:52:19     24] #  Metal 2         145
[12/05 22:52:19     24] #  Metal 3          60
[12/05 22:52:19     24] #  Metal 4          22
[12/05 22:52:19     24] #----------------------
[12/05 22:52:19     24] #                  465
[12/05 22:52:19     24] #
[12/05 22:52:19     24] #Total number of involved regular nets 21
[12/05 22:52:19     24] #Maximum src to sink distance  28.8
[12/05 22:52:19     24] #Average of max src_to_sink distance  16.7
[12/05 22:52:19     24] #Average of ave src_to_sink distance  15.9
[12/05 22:52:19     24] #Max overcon = 2 tracks.
[12/05 22:52:19     24] #Total overcon = 2.02%.
[12/05 22:52:19     24] #Worst layer Gcell overcon rate = 0.00%.
```

Fig: Optimized Design

The previous design has total 537 metal whereas the optimized design has 465 metal. So the number of metal is reduced in optimized design. The source to sink distance has also decreased approximately 20% in the optimized design than the previous design.

Optimization: Track Assignment

```
[11/17 12:45:38    149] #Track assignment summary:
[11/17 12:45:38    149] #layer  (wire length)   (overlap)       (long ovlp)
[11/17 12:45:38    149] #-------------------------------------------------
[11/17 12:45:38    149] #M1          0.00        0.00%          0.00%
[11/17 12:45:38    149] #M2        349.71        0.24%          0.00%
[11/17 12:45:38    149] #M3        508.09        0.47%          0.00%
[11/17 12:45:38    149] #M4        178.44        0.00%          0.00%
[11/17 12:45:38    149] #M5         37.27        0.00%          0.00%
[11/17 12:45:38    149] #M6          0.00        0.00%          0.00%
[11/17 12:45:38    149] #M7          0.00        0.00%          0.00%
[11/17 12:45:38    149] #M8          0.00        0.00%          0.00%
[11/17 12:45:38    149] #M9          0.00        0.00%          0.00%
[11/17 12:45:38    149] #M10         0.00        0.00%          0.00%
[11/17 12:45:38    149] #M11         0.00        0.00%          0.00%
[11/17 12:45:38    149] #-------------------------------------------------
[11/17 12:45:38    149] #All      1073.50        0.30%          0.00%
```

Fig: Track Assignment for Previous Design

```
[12/05 22:52:19     24] #Track assignment summary:
[12/05 22:52:19     24] #layer  (wire length)   (overlap)       (long ovlp)
[12/05 22:52:19     24] #-------------------------------------------------
[12/05 22:52:19     24] #M1          21.72        0.00%          0.00%
[12/05 22:52:19     24] #M2         170.48        0.09%          0.00%
[12/05 22:52:19     24] #M3         356.53        0.45%          0.00%
[12/05 22:52:19     24] #M4         143.65        0.22%          0.00%
[12/05 22:52:19     24] #M5         104.10        0.00%          0.00%
[12/05 22:52:19     24] #M6           0.00        0.00%          0.00%
[12/05 22:52:19     24] #M7           0.00        0.00%          0.00%
[12/05 22:52:19     24] #M8           0.00        0.00%          0.00%
[12/05 22:52:19     24] #M9           0.00        0.00%          0.00%
[12/05 22:52:19     24] #M10          0.00        0.00%          0.00%
[12/05 22:52:19     24] #M11          0.00        0.00%          0.00%
[12/05 22:52:19     24] #-------------------------------------------------
[12/05 22:52:19     24] #All        796.47        0.26%          0.00%
```

Fig: Track Assignment for Optimized Design

The wire length has been reduced significantly in the optimized design.

Complete Track Assignment:

```
[11/17 12:45:38    149] #Total wire length = 1165 um.
[11/17 12:45:38    149] #Total half perimeter of net bounding box = 1306 um.
[11/17 12:45:38    149] #Total wire length on LAYER Metal1 = 54 um.
[11/17 12:45:38    149] #Total wire length on LAYER Metal2 = 347 um.
[11/17 12:45:38    149] #Total wire length on LAYER Metal3 = 548 um.
[11/17 12:45:38    149] #Total wire length on LAYER Metal4 = 177 um.
[11/17 12:45:38    149] #Total wire length on LAYER Metal5 = 39 um.
[11/17 12:45:38    149] #Total wire length on LAYER Metal6 = 0 um.
[11/17 12:45:38    149] #Total wire length on LAYER Metal7 = 0 um.
[11/17 12:45:38    149] #Total wire length on LAYER Metal8 = 0 um.
[11/17 12:45:38    149] #Total wire length on LAYER Metal9 = 0 um.
[11/17 12:45:38    149] #Total wire length on LAYER Metal10 = 0 um.
[11/17 12:45:38    149] #Total wire length on LAYER Metal11 = 0 um.
[11/17 12:45:38    149] #Total number of vias = 537
[11/17 12:45:38    149] #Up-Via Summary (total 537):
```

Fig: Complete Track Assignment for Previous Design

```
[12/05 22:52:19     24] #Complete Track Assignment.
[12/05 22:52:19     24] #Total wire length = 863 um.
[12/05 22:52:19     24] #Total half perimeter of net bounding box = 1015 um.
[12/05 22:52:19     24] #Total wire length on LAYER Metal1 = 65 um.
[12/05 22:52:19     24] #Total wire length on LAYER Metal2 = 168 um.
[12/05 22:52:19     24] #Total wire length on LAYER Metal3 = 380 um.
[12/05 22:52:19     24] #Total wire length on LAYER Metal4 = 143 um.
[12/05 22:52:19     24] #Total wire length on LAYER Metal5 = 106 um.
[12/05 22:52:19     24] #Total wire length on LAYER Metal6 = 0 um.
[12/05 22:52:19     24] #Total wire length on LAYER Metal7 = 0 um.
[12/05 22:52:19     24] #Total wire length on LAYER Metal8 = 0 um.
[12/05 22:52:19     24] #Total wire length on LAYER Metal9 = 0 um.
[12/05 22:52:19     24] #Total wire length on LAYER Metal10 = 0 um.
[12/05 22:52:19     24] #Total wire length on LAYER Metal11 = 0 um.
[12/05 22:52:19     24] #Total number of vias = 465
[12/05 22:52:19     24] #Up-Via Summary (total 465):
```

Fig: Complete Track Assignment for Optimized Design

Placement Density:

```
Begin checking placement ... (start mem=1244.2M, init mem=1244.2M)
*info: Placed = 72
*info: Unplaced = 0
Placement Density:43.64%(197/451)
Placement Density (including fixed std cells):43.64%(197/451)
Finished checkPlace (cpu: total=0:00:00.0, vio checks=0:00:00.0; mem=1244.2M)
```

Fig: Placement Density for Previous Design

```
[12/05 22:52:17     22] Core basic site is CoreSite
[12/05 22:52:17     22] Layer info - lib-1st H=1, V=2.  Cell-FPin=1. Top-pin=2
[12/05 22:52:17     22] Begin checking placement ... (start mem=1158.4M, init mem=1158.4M)
[12/05 22:52:17     22] *info: Placed = 72
[12/05 22:52:17     22] *info: Unplaced = 0
[12/05 22:52:17     22] Placement Density:94.12%(197/209)
[12/05 22:52:17     22] Placement Density (including fixed std cells):94.12%(197/209)
[12/05 22:52:17     22] Finished checkPlace (cpu: total=0:00:00.0, vio checks=0:00:00.0; mem=1158.4M)
```

Fig: Placement Density for Optimized Design

The chip density after placement was 43.64% but after optimization the placement density is 94.12%.

DRC Check:

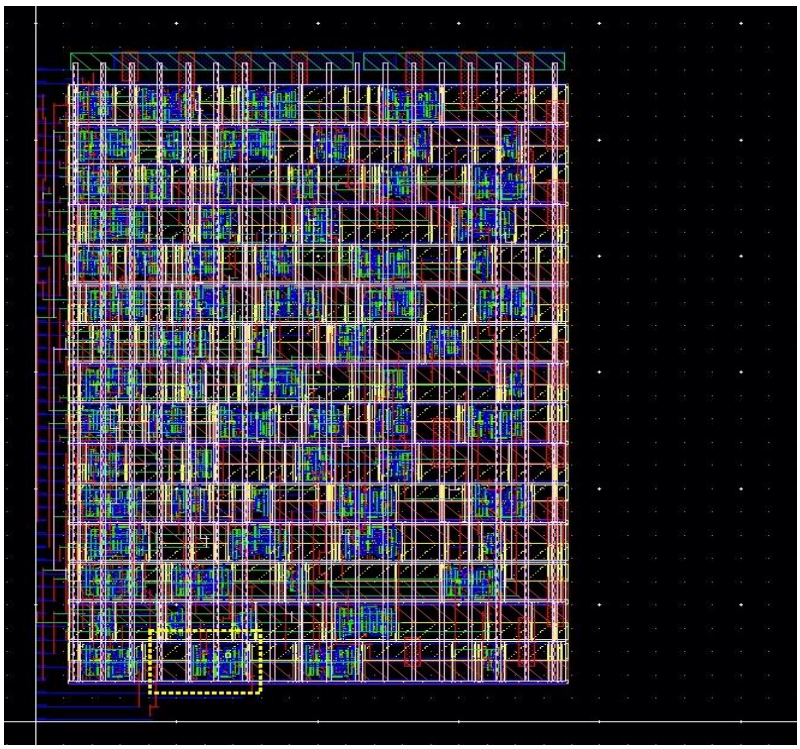


Fig: Layout View for Previous Design

```
Total CPU Time                  : 1(s)
Total Real Time                 : 2(s)
Peak Memory Used                : 20(M)
Total Original Geometry         : 1824(15608)
Total DRC RuleChecks            : 562
Total DRC Results               : 0 (0)
Summary can be found in file csa.sum
ASCII report database is /home/vlsi31/Carry_Skip_Adder_Project/CSA_Layout/csa.drc_errors.ascii
Checking in all SoftShare licenses.


Design Rule Check Finished Normally. Tue Dec  3 15:36:35 2024
```

Fig: DRC Verification for Previous Design

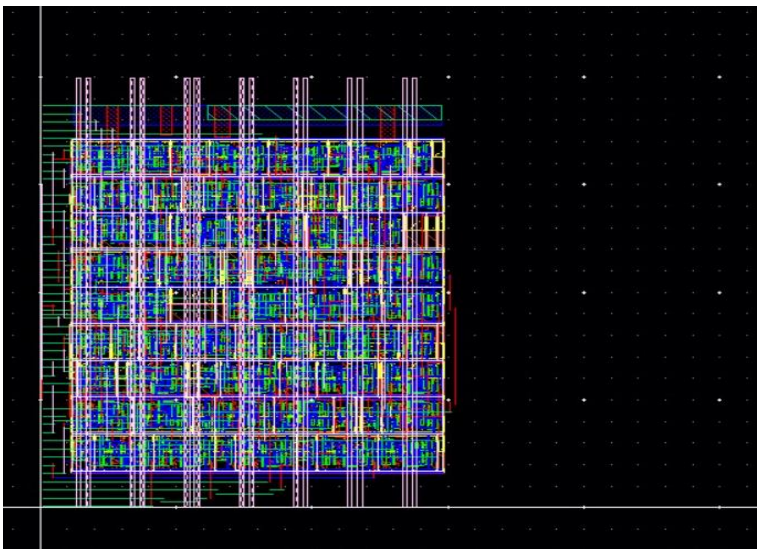


Fig: Layout View of Optimized Design

```
Total CPU Time                  : 1(s)
Total Real Time                 : 1(s)
Peak Memory Used                : 20(M)
Total Original Geometry         : 1381(8772)
Total DRC RuleChecks            : 562
Total DRC Results               : 0 (0)
Summary can be found in file csa.sum
ASCII report database is /home/vlsi31/Carry_Skip_Adder_Project/Optimized_CSA_Layout/csa.drc_errors.ascii
Checking in all SoftShare licenses.


Design Rule Check Finished Normally. Sat Dec  7 02:34:59 2024
```

Fig: DRC Verification for Optimized Design

5 Design Analysis and Evaluation

5.1 Novelty

In this experiment, we aimed to enhance our design by maximizing area density, a crucial factor for improving overall efficiency and performance. Initially, we discovered that the area density was a modest 27%. To address this, we implemented a series of design modifications and optimizations that successfully elevated the density to 43%. Building on these initial improvements, we further refined our approach by exploring the effects of varying the die size. This critical adjustment allowed us to increase the area density significantly, achieving an impressive 94%.

The optimization process involved careful calculations and analyses to select die sizes that would yield the best performance while maintaining structural integrity. In addition to optimizing area density, we focused on reducing the lengths of both metal interconnects and wiring within the design. This reduction not only streamlined the physical layout but also minimized resistance and signal delay, leading to improved electrical performance.

We also prioritized the reduction of the distance between the source and sink in our design, which is essential for enhancing data transfer speeds and overall responsiveness. Overall, this comprehensive optimization process not only achieved our goals for area density but also contributed to a more efficient and effective design, ready for practical applications.

6. Reflection on Individual and Team work

6.1 Individual Contribution of Each Member

All members contributed equally to project implementation. They actively tackled issues and met specifications with their best efforts

6.2 Log Book of Project Implementation

Date	**Progress**	
October 28,2024	Designing the Circuit	
November 13,2024	System Verilog and Testbench	Finally succeeded to meet specifications but still with deviations
November 25,2024	Layered Testbench	
November 26,2024	Synthesis	
November 28,2024	Layout	
December 3,2024	Optimization	

7 Conclusion

In this experiment, we successfully designed and implemented a 16-bit carry skip adder, which is a type of digital circuit that enhances the performance of arithmetic operations by effectively managing carry propagation. To validate our design, we created a comprehensive testbench that allows us to thoroughly evaluate its functionality. In addition to the standard testbench, we also constructed a layered testbench to assess the adder's performance under various conditions and scenarios. We proceeded with RTL synthesis, translating our high-level design into a format suitable for logic implementation, followed by place and route (PnR) using Innovus. This allowed us to map the synthesized design onto the physical layout of the chip. To optimize the area density of our design, we experimented with different die sizes, carefully balancing the trade-offs between area and performance. Throughout the process, we conducted design rule checks (DRC) and geometric verification, ensuring that our layout met all required specifications. We are pleased to report that no errors or violations were encountered during these checks. The waveform produced by our evaluation process demonstrates that the designed adder operates effectively and meets our expectations. However, it is important to note that we did not conduct any synthesis optimization, as our circuit design did not incorporate a clock signal. This omission limited certain optimizations that typically enhance performance.

8 References

www.Wikipedia.com