PROJECT ON RTL TO GDS2 FLOW

Submitted for requirements for the practical coursework

of

M Tech (VDN)

by

Akash Pandey (2302102035)

Shivam Vaish (2302102034)

DISCIPLINE OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

15 Nov. 2023

INDIAN INSTITUTE OF TECHNOLOGY INDORE

I hereby certify that the work which is being presented in the report entitled **Project on RTL to GDS flow** submitted for requirements of the practical coursework of **M Tech (VDN)** and submitted in the Discipline of Electrical Engineering**, Indian Institute of Technology Indore**, is an authentic record of our group work carried out during the time period from Oct 2023 to Nov 2023 under the supervision of Dr. Santosh Kumar Vishwakarma. The matter presented in this report has been cited properly wherever necessary.

Akash Pandey (2302102035)
Shivam Vaish (2302102034)

This is to certify that the above statement made by the candidate is correct to the best of my/our knowledge.

(Dr. Santosh Kumar Vishvakarma)

ACKNOWLEDGEMENTS

First of all, We would like to thank my Supervisor Dr. Santosh Kumar Vishwakarma whose constant motivation, guidance and suggestion helped me to accomplish the task. I would also like to thank him for providing required software resources associated with Nanoscale Devices, VLSI Circuit and System Design (NSDCS) Lab. I Would Like To Thank NSDCS Phd Scholar Mr. Mukul, Ms. Vasundhara and all other Phd scholars for guidance and support and fellow MTech VDN students for constant motivation towards achieving the task.

2. ABSTRACT

RTL to GDS2 flow(ASIC) design flow is a mature and silicon-proven IC design process which includes various steps like design conceptualization, chip optimization, logical/ physical implementation, and design validation and verification.

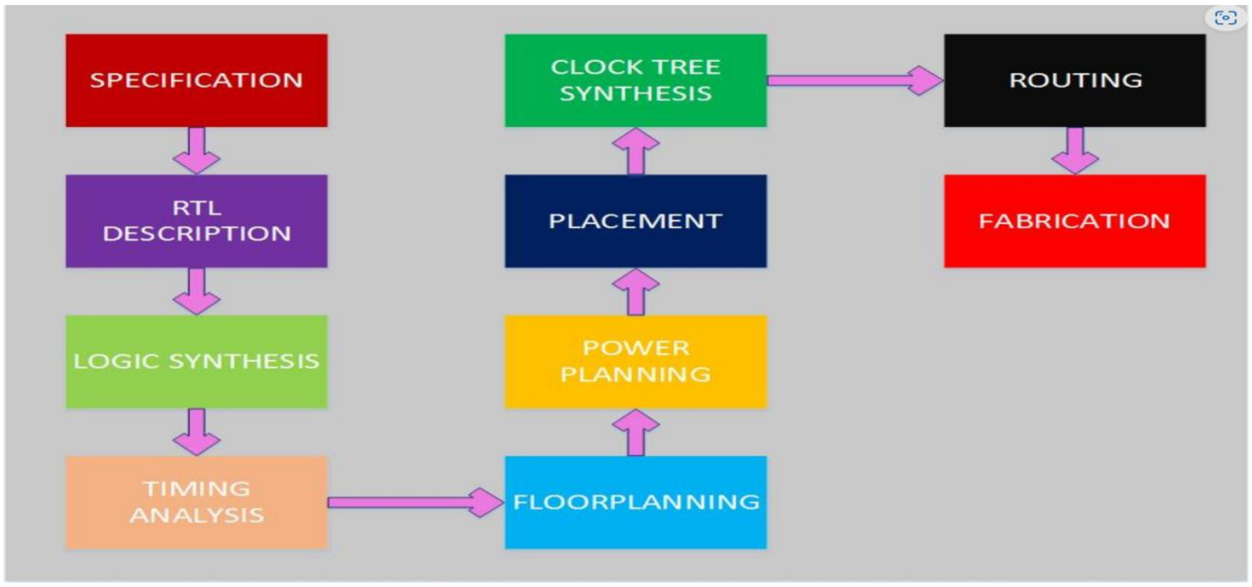


Fig.: RTL to GDS2 flow Block diagram

In this project we will implement rtl to gds2 flow for a 1bit full adder. This project will involve various steps such as RTL design(Verilog/VHDL code in Vivado – XILINX), Logic synthesis (converting the code into a circuit), Floorplanning and place & route (determining the arrangement of logical blocks, placement of I/O pins and routing tracks on the silicon chip) , Clock tree synthesis (distributing the clock equally among all sequential parts of a VLSI design), Timing analysis (evaluating the design's timing performance), Power analysis (calculates (estimates of) the active and static leakage power dissipation of the SoC design), Design verification.

In logic synthesis section we have compared the designs of Ripple carry adder, carry look ahead adder, carry skip adder, carry select adder. In this section we have compared performance metrices of these adder designs such as timing report, utilization report, area reports etc. In conclusion we have summarized the all the key observations and findings. Further we have also discussed on various future scopes in this direction.

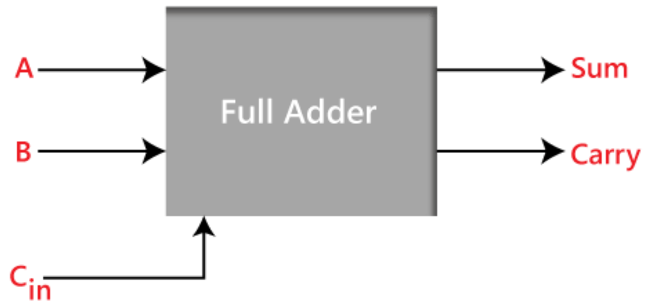
3. TABLE OF CONTENTS

1. INTRODUCTION

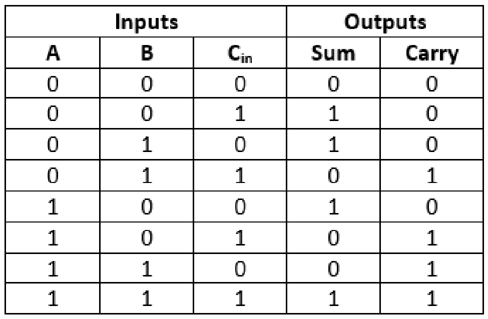
Full Adder

The half adder is used to add only two numbers. To overcome this problem, the full adder was developed. The full adder is used to add three 1-bit binary numbers A, B, and carry C. The full adder has three input states and two output states i.e., sum and carry.

Block diagram



Truth Table



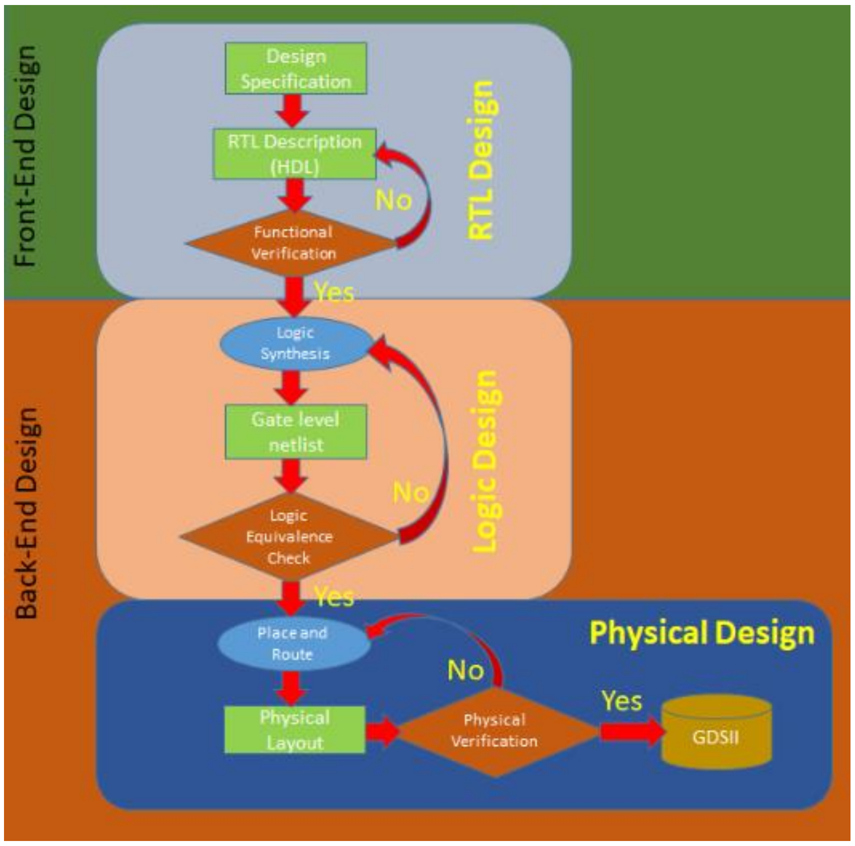
Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

In the above table,

1. 'A' and' B' are the input variables. These variables represent the two significant bits which are going to be added
2. 'C_{in}' is the third input which represents the carry. From the previous lower significant position, the carry bit is fetched.
3. The 'Sum' and 'Carry' are the output variables that define the output values.
4. The eight rows under the input variable designate all possible combinations of 0 and 1 that can occur in these variables.

2) OVERVIEW OF RTL TO GDS - II DESIGN FLOW

Physical Design is a process of converting the RTL netlist into a layout that is manufacture-able (GDS). This process of converting a netlist into a manufacture-able layout involves multiple steps. The flow starts with RTL coding and ends with GDS (Graphic Data Stream) file which is the final output of back end design, so this complete flow is also known as RTL to GDS (RTL2GDS) flow. A Simple flow diagram has been described here.



The complete design process can be divided into two parts.

- Front End Design
- Back End Design

Front End Design:

Front end design process starts with the specification received from the customer end. RTL (Register Transfer Level) design engineer converts the specification into an RTL code using the

HDL (Hardware Description Language) generally either in Verilog or VHDL. Once the RTL code is written, RTL designer simulates the code in RTL Simulator and check the functionality of the design. Once the functionality of code is correct and verified by the verification engineers and if there is no bug found, RTL code received from the front end engineer is technology independent, now the next step is Logic synthesis.

Back End Design:

- **Logic Synthesis:** In logic synthesis, a high-level description of the design (RTL Code) is converted into an optimized gate-level representation of a given standard cell library and certain design constraints. Now the code is in the form of a gate-level netlist of a particular standard cell library.

  LEC (Logic Equivalence Check is must in this stage to make sure that there are not logical changes occurred during the synthesis. During logical Synthesis, we also get various reports on timing power and area of design. We also get an SDC (Synopsys Design Constraint) file in this stage which is used in the next stage.

  DFT (Design For Testability) Insertion is also done in this stage to verify the chip after fabrication is done.

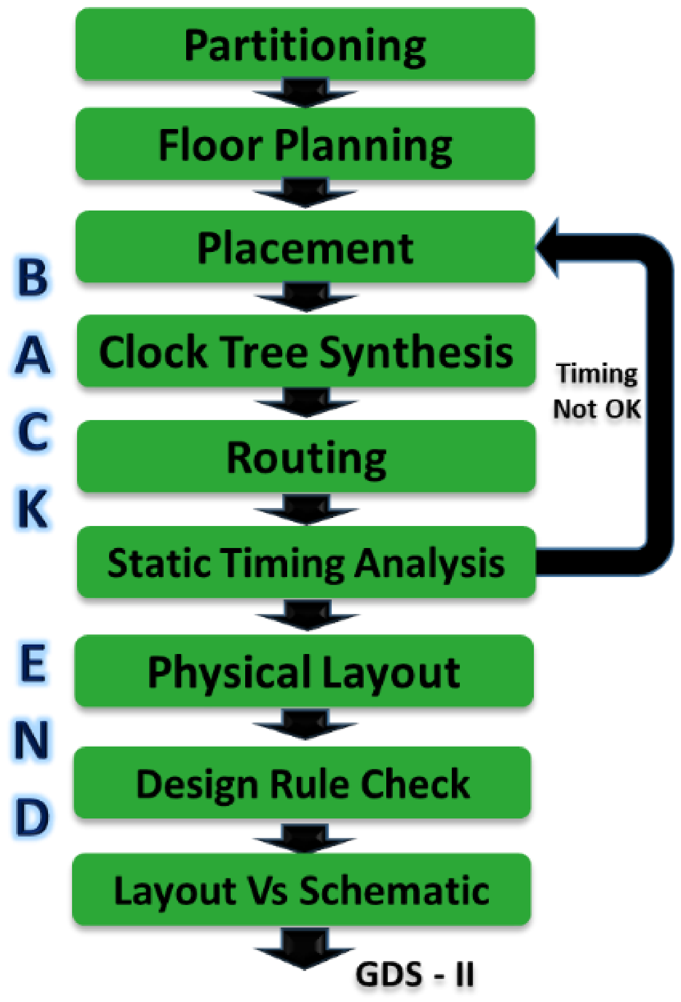
- **Place and Route (PnR):** Gate level netlist after DFT Insertion and SDC file is taken as input for the PnR and based on standard cells library, PnR starts. The goal of PnR stage is to place all the standard cells, Macros and I/O pads with minimal area, with minimal delay and Route them together in such a way that there is no DRC (Design Rule Check) error. The final output of this stage is the layout of design in the form of GDSII file which is defacto standard of layout file in the industry.

  PnR stage is a very challenging stage with large design cycle time depending on the complexity of a chip. This stage is further divided into various sub-stages. The main stages are starting from Design Import, followed by FloorPlan, Power Plan, Placement, CTS (Clock Tree Synthesis), and Routing.

  After routing we expect the design has met the timing and all DRC, but in the modern chip, it’s not easy to close the design in this stage. So Further we go to Signoff stage.

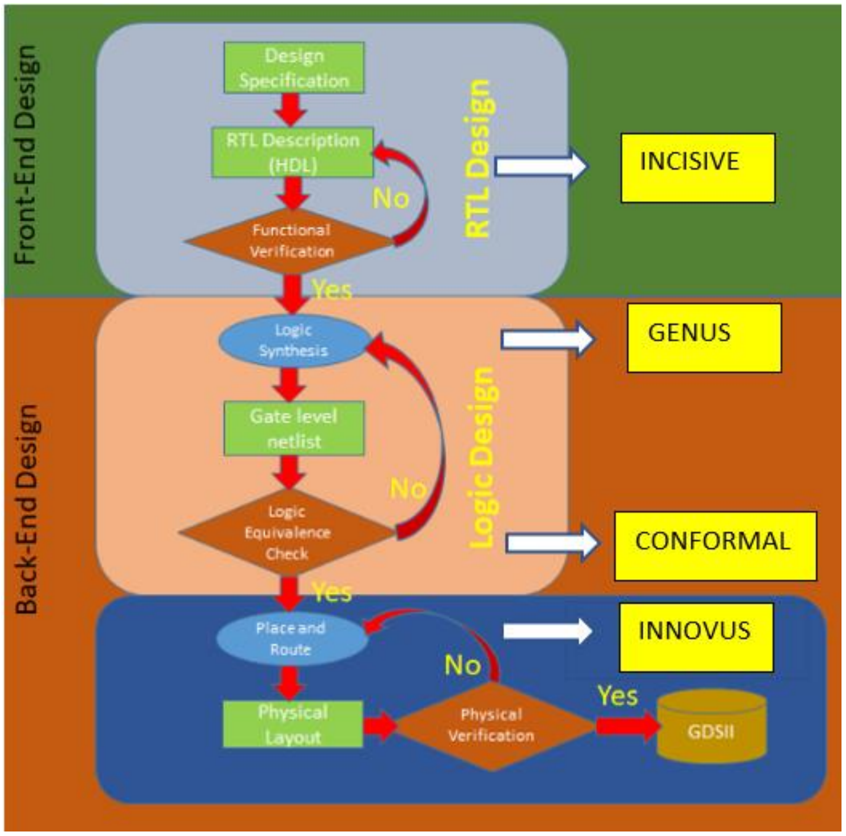
- **Signoff:** If there are some timing violations in post route design, we have a further stage called ECO (Engineering Change Order) where we can fix the timing violations. Apart from timing violation, there may be issues like IR Drop, DRC Violations all these are fixed in this stage and a final layout file free from all the violation is streamed out in GDSII format. This process is known as tapeout in design flow. This is the final design stage and GDSII file is sent to fabrication lab for the fabrication of chip.

In the subsequent sections we will elaborate on following steps of back end designing:



2.1) TOOLS USED IN THE PROJECT

The flow typically proceeds as follows: RTL design and verification in Incisive, synthesis with Genus, static timing analysis with Modus, formal verification with Conformal, and physical design using Innovus. The output of this flow is a GDS-II file that contains the detailed layout of the chip, ready for manufacturing. Each tool plays a crucial role in ensuring that the final design meets its functional, timing, and physical requirements.



Incisive (SimVision):

- **Role:** Incisive is a simulation and verification tool used primarily in the RTL phase of the design flow.
- **Functionality:** It allows designers to simulate and verify the functionality and correctness of the RTL code, ensuring that the design meets its functional requirements.
- **Features:** Supports various simulation methodologies, including event-driven and transaction-level modeling. Provides robust debugging capabilities, coverage analysis, and testbench development features.

Genus:

- **Role:** Genus is a synthesis tool used during the synthesis phase of the RTL to GDS-II flow.
- **Functionality:** It takes RTL code and converts it into gate-level logic that can be used for further physical design steps.
- **Features:** Genus is known for its high-performance and efficient synthesis algorithms. It supports optimizations for area, power, and timing, making it a crucial tool in optimizing the design for power consumption and performance.

Modus:

- **Role:** Modus is Cadence's static timing analysis tool used to ensure that the design meets its timing requirements.
- **Functionality:** It performs static analysis to identify timing violations, such as setup and hold time violations, and helps designers understand and resolve these issues.
- **Features:** Modus can generate timing reports and constraints to guide designers in achieving timing closure for their designs.

Conformal:

- **Role:** Conformal is a formal verification tool used to verify the equivalence of different RTL descriptions or check for design properties.
- **Functionality:** It ensures that the RTL design and its corresponding gate-level representation are functionally equivalent, which is essential for maintaining design correctness.
- **Features:** Conformal can be used for equivalence checking, property checking, and sequential formal verification.

Innovus:

- **Role:** Innovus is Cadence's place-and-route tool used in the physical design phase of the RTL to GDS-II flow.
- **Functionality:** Innovus takes synthesized gate-level netlists and performs placement, clock tree synthesis, and routing to create a physical layout that meets area, performance, and power requirements.
- **Features:** It offers advanced placement and routing algorithms, and it can optimize the design for power, timing, and manufacturability. Innovus generates the GDS-II file format for mask data preparation.

3. RTL DESIGN

Different types of adders along with their RTL design, specifications and architecture are discussed and compared in this section :

3.1) RTL DESIGN PHASE

➢ Full Adder :

- **Description**: A Full Adder is a combinational logic circuit that can add two binary digits (bits) and a carry bit, and produces a sum bit and a carry bit as output.

- **RTL Design:** At the RTL level, a FA is designed using two Half_Adders cells and an OR gate. Each half-adder cell takes two inputs (A, B) and produces two outputs (sum and carry).

- **Operation:** For a full adder, two half adder cells are cascaded together to perform binary addition and sum is obtained as it is. The carry outputs of the two half adders are given to an OR gate as inputs, the output gives the final carry expression.

- **Verilog Code :**

  1. **Main Code :**

```
`timescale 1ns/1ps
module clkFullAdder(
input a,b,cin,clk,
output reg sum,cout);
always@(posedge clk)
begin
sum<=a^b^cin;
cout<=a&b | b&cin | cin&a;
end
endmodule
```

2. Testbench Code :

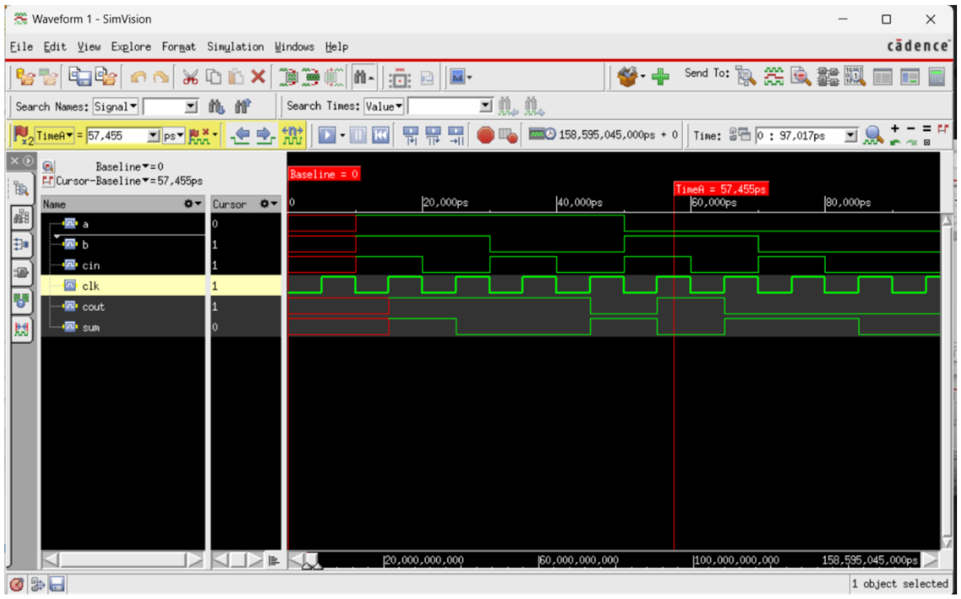
```
`timescale 1ns/1ps
module tb_clkFullAdder();
reg a,b,cin,clk;
wire cout,sum;
clkFullAdder tm(a,b,cin,clk,sum,cout);
initial begin
 clk=0;
#10 {a,b,cin}=3'b111;
#10 {a,b,cin}=3'b110;
#10 {a,b,cin}=3'b101;
#10 {a,b,cin}=3'b100;
#10 {a,b,cin}=3'b011;
#10 {a,b,cin}=3'b010;
#10 {a,b,cin}=3'b001;
#10 {a,b,cin}=3'b000;

end

always #5 clk=~clk;
endmodule
```

3.2) FUNCTIONAL VERIFICATION

- Full Adder testbench is simulated using simvision tool to verify the functionality of the Full Adder circuit .



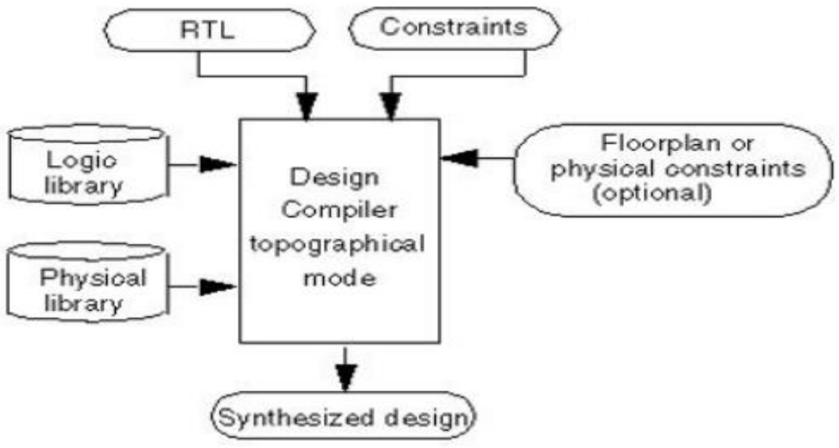
4.SYNTHESIS

4.1) SYNTHESIS PROCESS

Synthesis is the process of transferring higher level of abstraction (RTL) to lower level of abstraction. It is the process of transforming RTL to gate-level netlist(represents the circuit using gates, contains all the gate level information and the connection between these gates). During synthesis, the tools optimizes the design ensuring that the functionality remains unchanged. Synthesis optimization focuses on several key areas, including:

- Timing: Ensuring that the design meets the required performance by optimizing the critical paths and minimizing setup and hold violations.
- Power: Reducing the overall power consumption by optimizing the switching activity and implementing techniques such as clock gating and power gating.
- Area: Minimizing the silicon area consumed by the design, which can help reduce the manufacturing cost and improve yield.

  - **Inputs for Synthesis**
    1. RTL : Verilog/HDL files
    2. Libraries
    3. Constraints (SDC file)
    4. TCL script
  - **Outputs for Synthesis**
    1. Netlist
    2. Reports (QOR, Area, Power, Timing etc.)


RTL
Constraints
Logic library
Design Compiler topographical mode
Floorplan or physical constraints (optional)
Physical library
Synthesized design

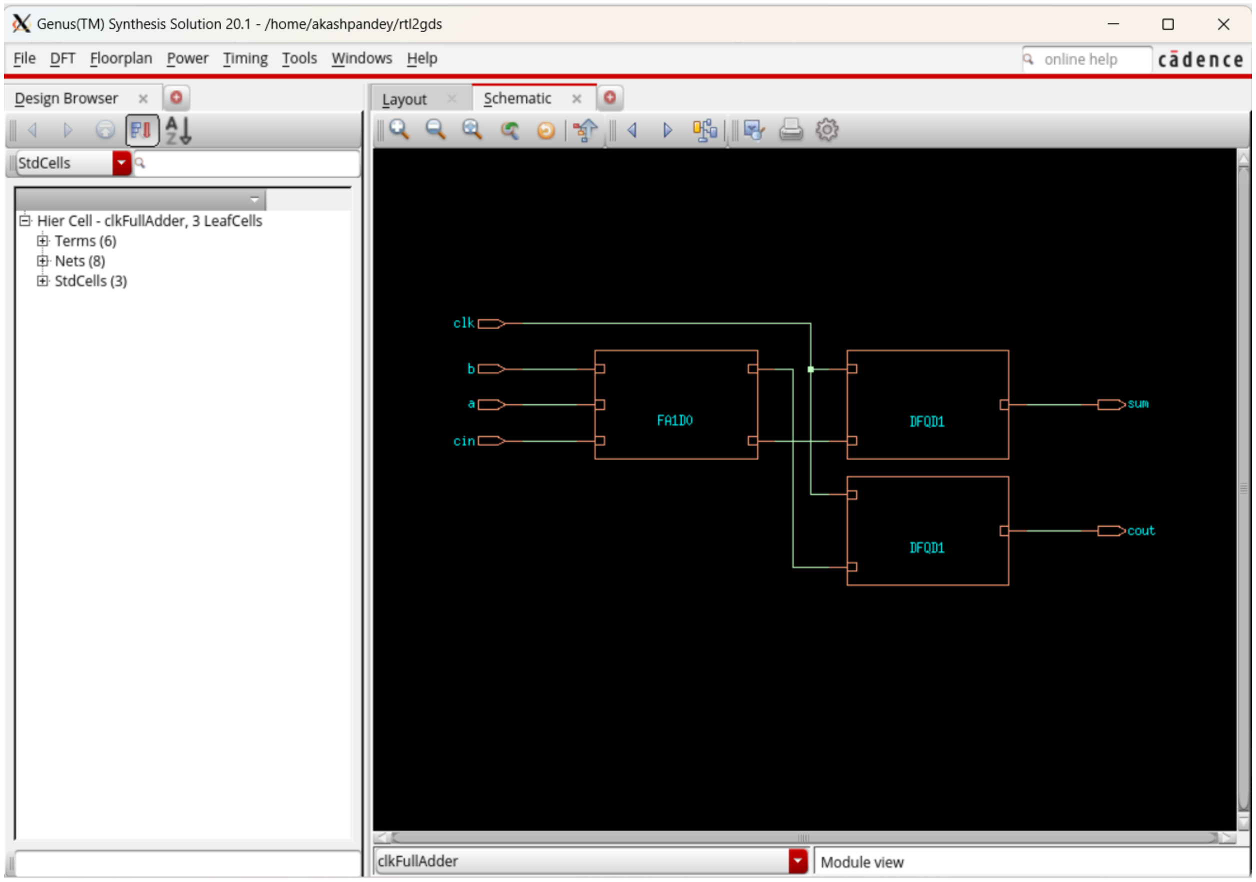

Genus(TM) Synthesis Solution 20.1 - /home/akashpandey/rtl2gds
File DFT Floorplan Power Timing Tools Windows Help
online help
cadence
Design Browser
Layout
Schematic
StdCells
Hier Cell - clkFullAdder, 3 LeafCells
Terms (6)
Nets (8)
StdCells (3)
clk
b
a
cin
FA1D0
DFQD1
DFQD1
sum
cout
clkFullAdder
Module view

Fig. : Synthesis of Full Adder RTL using Cadence-Genus

Synthesized Gate Netlist:

```
module clkFullAdder(a, b, cin, clk, sum, cout);
  input a, b, cin, clk;
  output sum, cout;
  wire a, b, cin, clk;
  wire sum, cout;
  wire n_0, n_1;
  DFQD1 sum_reg(.CP (clk), .D (n_1), .Q (sum));
  DFQD1 cout_reg(.CP (clk), .D (n_0), .Q (cout));
  FA1D0 g155__2398(.A (b), .B (a), .CI (cin), .CO (n_0), .S (n_1));
endmodule
```

Synthesized Synopsis Design Constraints:

```
# ####################################################################
#  Created by Genus(TM) Synthesis Solution 20.11-s111_1 on Sun Oct 20 14:26:42 IST 2024
# ####################################################################
set sdc_version 2.0
set_units -capacitance 1000fF
set_units -time 1000ps
# Set the current design
current_design clkFullAdder
create_clock -name "clk" -period 10.0 -waveform {0.0 5.0} [get_ports clk]
set_clock_gating_check -setup 0.0
set_input_delay -clock [get_clocks clk] -add_delay 0.0001 [get_ports clk]
set_input_delay -clock [get_clocks clk] -add_delay 0.0001 [get_ports cin]
set_input_delay -clock [get_clocks clk] -add_delay 0.0001 [get_ports b]
set_input_delay -clock [get_clocks clk] -add_delay 0.0001 [get_ports a]
set_wire_load_mode "segmented"
~
```

4.2) DESIGN FOR TESTABILITY (DFT)

In VLSI design, Design for Testability (DFT) is an approach that aims to make digital circuits easier to test during the manufacturing and debugging process. DFT in VLSI design involves incorporating additional circuitry and design features such as scan chains, built-in self-test (BIST) circuits, and boundary scan cells into the chip design to facilitate testing. Design for testability in VLSI design is essential to ensure that the fabricated chips are free from any kind of manufacturing defects. It also reduces the overall test time and thereby the cost of testing, and debugging. By incorporating DFT techniques into the chip design, it becomes easier to test the structural correctness of the chip, leading to higher-quality products and faster time-to-market.

Design for Testability (DFT) is essential in VLSI (Very Large Scale Integration) design because:

1. By designing a chip with testability in mind, it becomes easier to identify the structural defects in the chip and fix design errors before the product is shipped to customers.

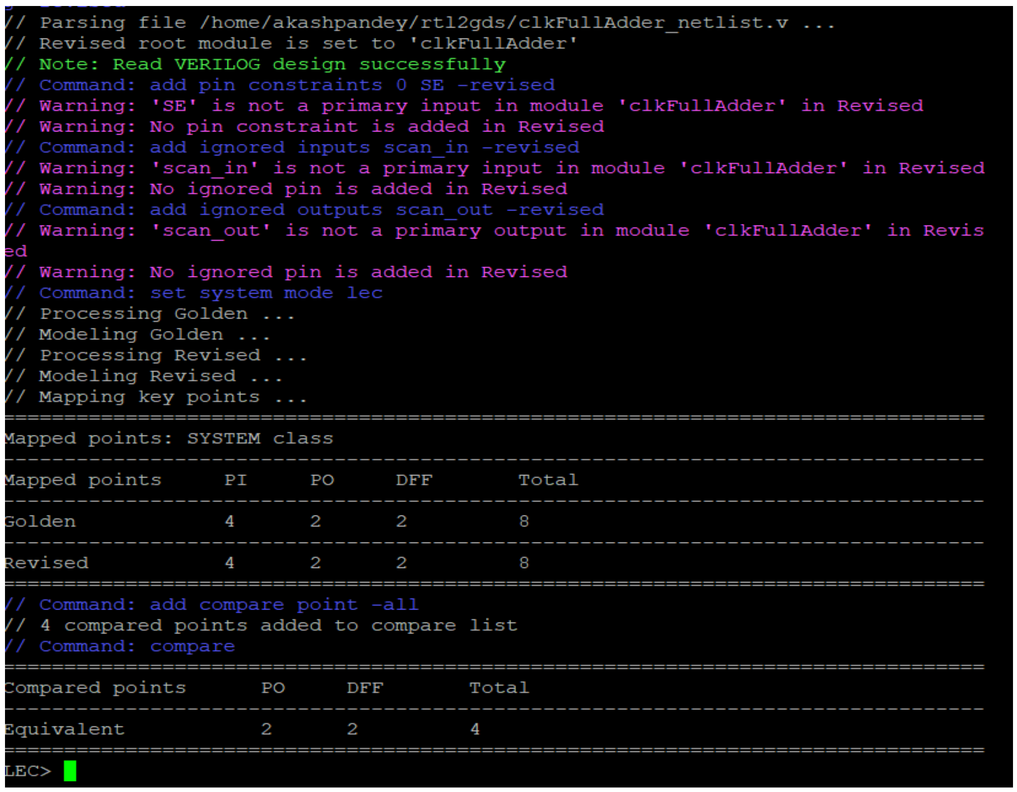
2. Designing a chip with built-in testability features can make the testing process more efficient, reducing the cost and time required for testing. This can lead to significant cost savings during the manufacturing process.
3. By incorporating DFT features into the chip design can help to identify any defects early in the process, allowing for faster repairs and improvements in production yield.
4. Design for testability can help accelerate the development cycle by reducing the time and effort required for testing and debugging.
5. DFT can make it easier to diagnose and fix problems in the chip design, reducing the effort required for maintenance and updates.

5. Logic Equivalence Check

In physical design, logic equivalence checking (LEC) ensures that the gate-level netlist produced during RTL synthesis is functionally equivalent to the original RTL description1. This process is crucial because it verifies that the transformations and optimizations applied during synthesis, place and route, and other stages do not alter the intended functionality of the design.

Key Steps in Logic Equivalence Check

1. Model Extraction: Extract models of the RTL and the gate-level netlist.
2. Mapping: Identify corresponding points (equivalence points) in the two models.
3. Comparison: Use formal methods to compare the Boolean functions at the equivalence points.
4. Debugging: If discrepancies are found, debug the circuits to identify and correct the source of the mismatch.



```
// Parsing file /home/akashpandey/rtl2gds/clkFullAdder_netlist.v ...
// Revised root module is set to 'clkFullAdder'
// Note: Read VERILOG design successfully
// Command: add pin constraints 0 SE -revised
// Warning: 'SE' is not a primary input in module 'clkFullAdder' in Revised
// Warning: No pin constraint is added in Revised
// Command: add ignored inputs scan_in -revised
// Warning: 'scan_in' is not a primary input in module 'clkFullAdder' in Revised
// Warning: No ignored pin is added in Revised
// Command: add ignored outputs scan_out -revised
// Warning: 'scan_out' is not a primary output in module 'clkFullAdder' in Revis
ed
// Warning: No ignored pin is added in Revised
// Command: set system mode lec
// Processing Golden ...
// Modeling Golden ...
// Processing Revised ...
// Modeling Revised ...
// Mapping key points ...
================================================================================
Mapped points: SYSTEM class
--------------------------------------------------------------------------------
Mapped points     PI     PO     DFF       Total
--------------------------------------------------------------------------------
Golden            4      2      2         8
--------------------------------------------------------------------------------
Revised           4      2      2         8
================================================================================
// Command: add compare point -all
// 4 compared points added to compare list
// Command: compare
================================================================================
Compared points      PO     DFF       Total
--------------------------------------------------------------------------------
Equivalent           2      2         4
================================================================================
LEC>
```

Fig. : LEC of Gate level Netlist vs RTL Design using Conformal tool

6. PHYSICAL DESIGN

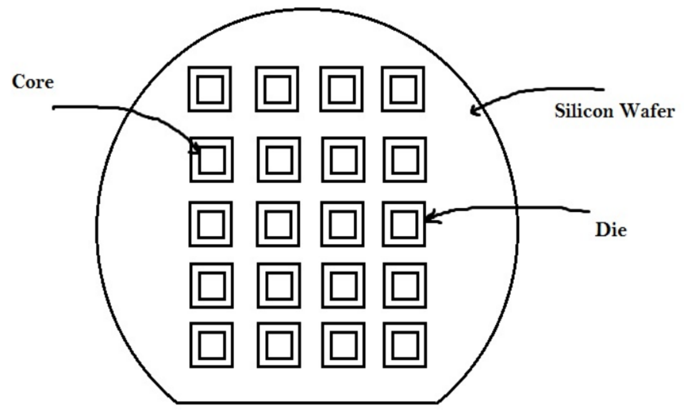
6.1) FLOORPLANNING

Floorplanning involves determining the locations, shape, size of modules/logical blocks (i.e. multiplexer, AND, OR gates, buffers) in a chip and as such it estimates the chip area, delay and the wiring congestion, thereby providing a ground work for layout. A good floorplan can make implementation process a cake walk. On similar lines, a bad floorplan can create all kinds of issues in the design (congestion, noise, routing issues). A bad floorplan will blow up the area, power and affects reliability, life of the IC and also it can increase overall IC cost.

- **Objectives of floorplan :**
    - Minimize the area
    - Minimize the timing
    - Reduce the wire length
    - Make routing easy
    - Reduce IR drop

- **Floorplan control parameters**

  A 'core' is the section of the chip where the fundamental logic of the design is placed. A die, which consists of core, is small semiconductor material specimen on which the fundamental circuit is fabricated. IC's are fabricated on a single 9 inch or 12 inch diameter silicon wafer, which contains hundreds of mirror images of the fundamental logic. This wafer is then cut into small pieces, each piece has similar functionality of the fundamental logic. This is called 'die'.

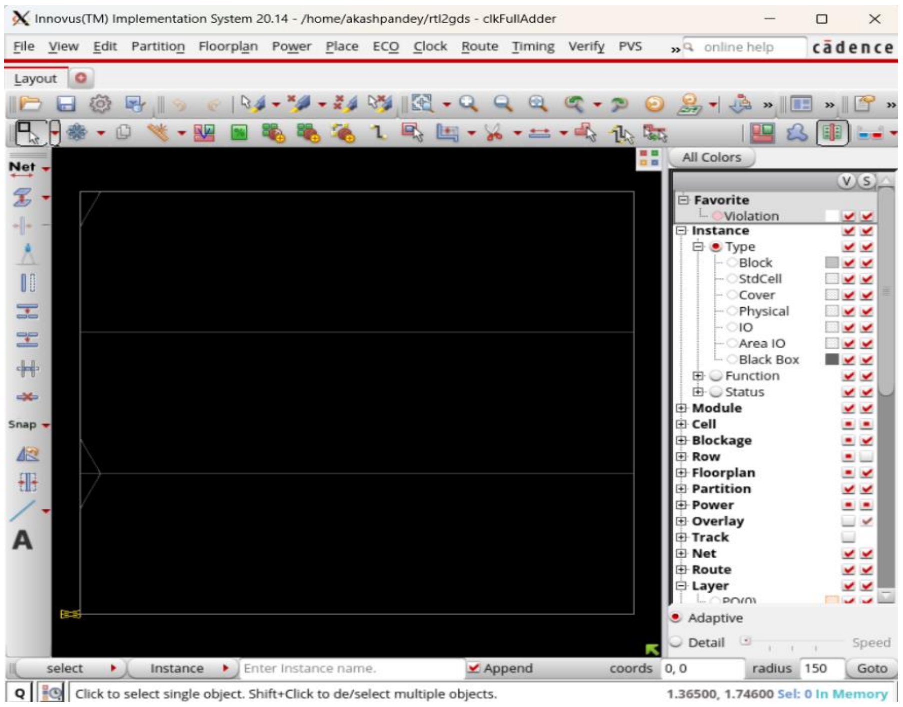


1. **<u>Aspect ratio</u>**: Aspect ratio will decide the size and shape of the chip. It is the ratio between horizontal routing resources to vertical routing resources (or) ratio of height and width.

$$\text{Aspect Ratio} = \frac{\text{Horizontal Routing Resource}}{\text{Vertical Routing Resource}}$$

2. **Core utilization**: Utilization will define the area occupied by the standard cells, macros, and other cells. If core utilization is 0.8 (80%) that means 80% of the core area is used for placing the standard cells, macros, and other cells, and the remaining 20% is used for routing purposes.

$$\text{Utilization} = \frac{\text{Total Standard Cell Area+Macro Area}}{\text{Total Core Area}} \times 100\ \%$$


Innovus(TM) Implementation System 20.14 - /home/akashpandey/rtl2gds - clkFullAdder

Click to select single object. Shift+Click to de/select multiple objects.

1.36500, 1.74600 Sel: 0 In Memory

Fig. : Core area of Chip

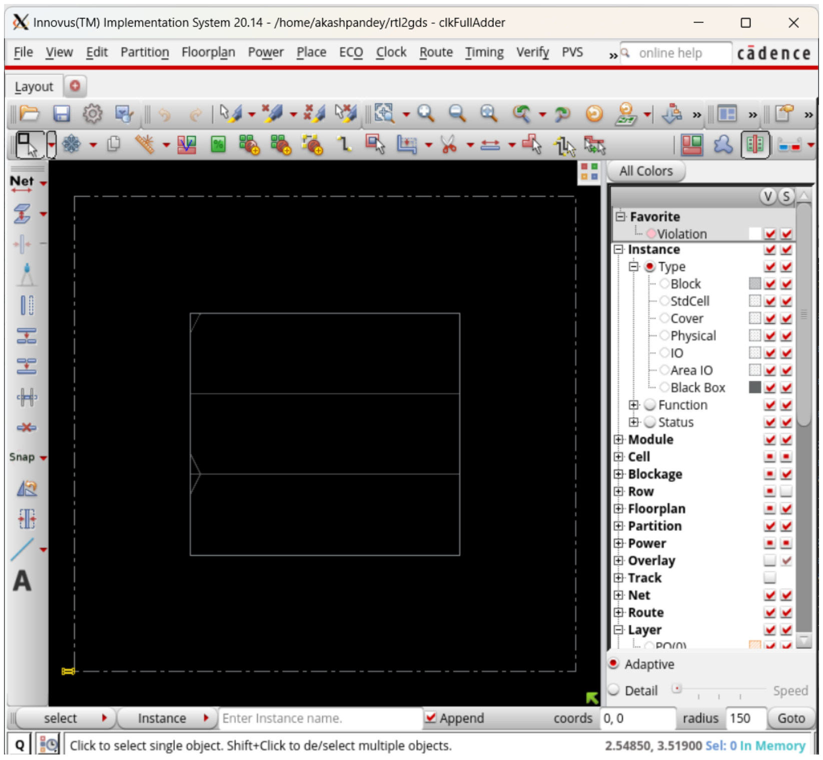


Fig. : Floorplanning of Core chip area

6.2) PLACEMENT

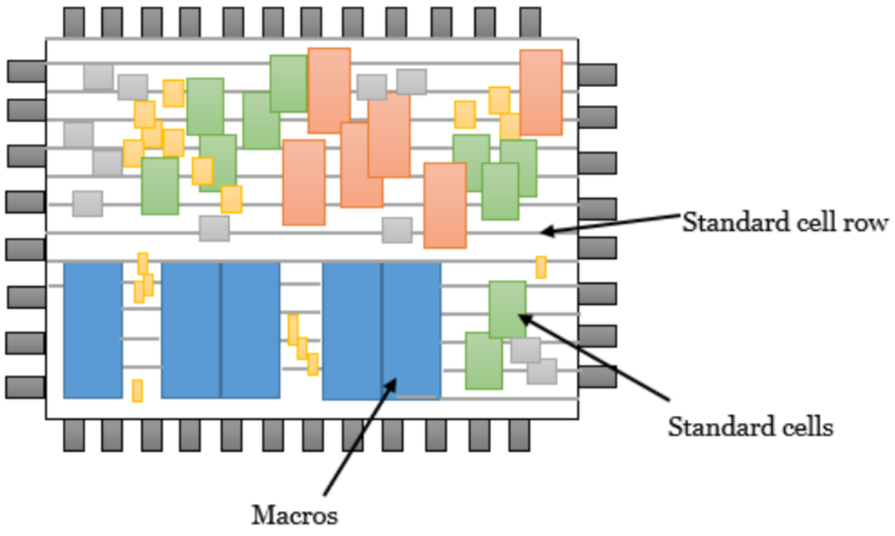
Placement is the process of finding a suitable physical location for each cell in the block. The tool only determine the location of each standard cell on the die. Placement does not just place the standard cell available in the synthesized netlist, it also optimized the design.

The tool determines the location of each of the standard cell on the core. Various factors come into play like the timing requirement of the system, the interconnect length and hence the connections between cells, power dissipation, etc. the interconnect length depends on the placement solution used, and it is very important in determining the performance of the system as the geometries shrink. Placement will be driven based on different criteria like timing driven, congestion driven, power optimization.

Placement is performed in two stages: coarse placement and legalization.

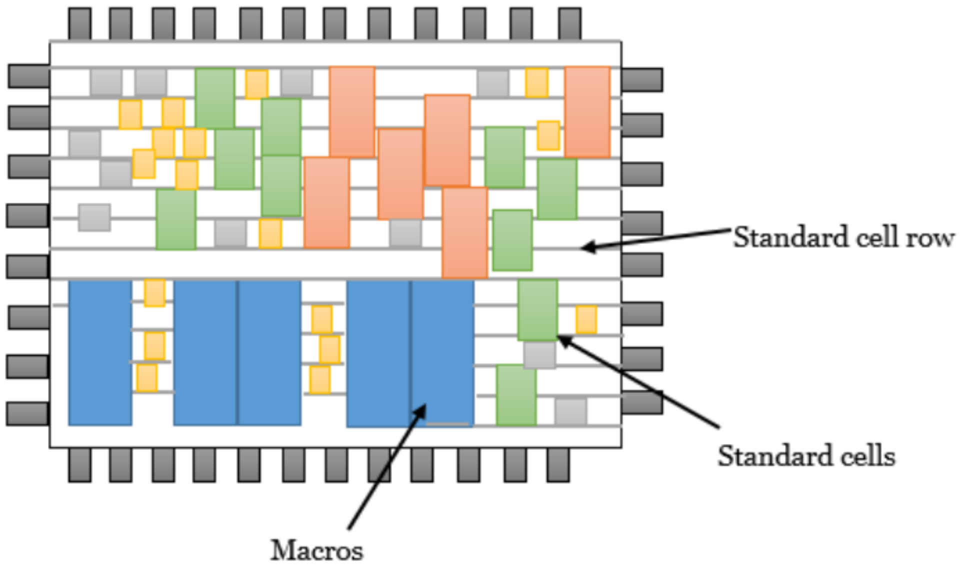
- **Coarse Placement**

  During the coarse placement, the tool determines an approximate location for each cell according to the timing, congestion and multi-voltage constraints. The placed cells don’t fall on the placement grid and might overlap each other. Large cells like RAM and IP blocks act as placement blockages for standard cells. Coarse placement is fast and sufficiently accurate for initial timing and congestion analysis**.**



- **Legalization**

  During legalization, the tool moves the cells to legal locations on the placement grid and eliminate any overlap between cells. These small changes to cell location cause the lengths of the wire connections to change, possibly causing new timing violations. Such violations can often be fixed by incremental optimization, for example: by resizing the driving cells.



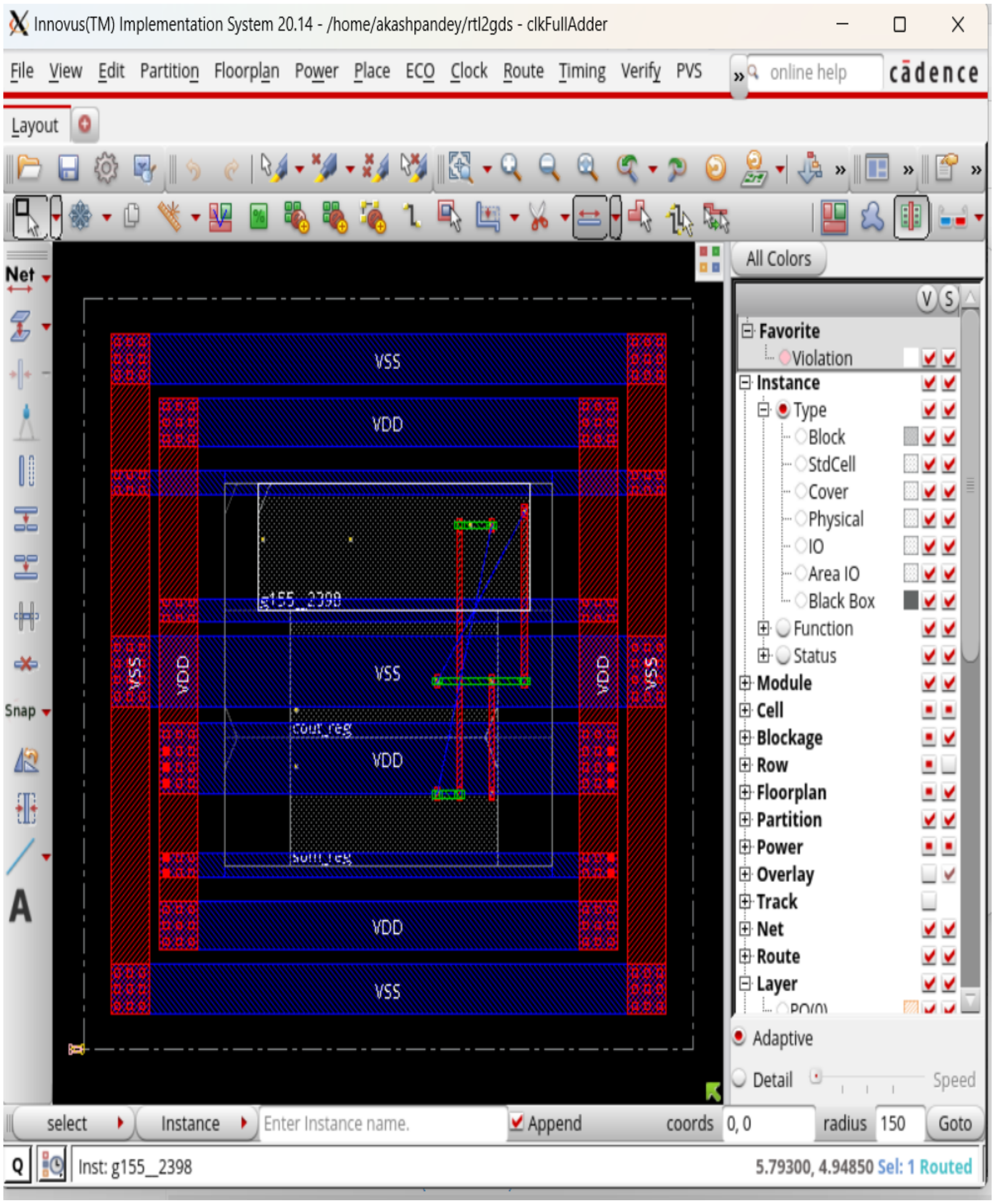


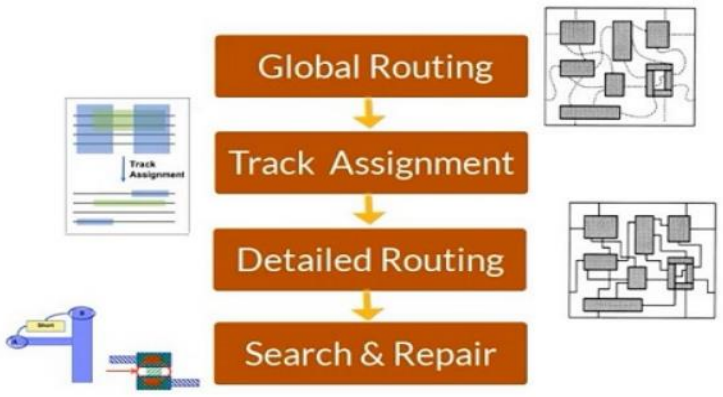
Fig. : Standard cell placement with power ring and strips routed in Innovus tool

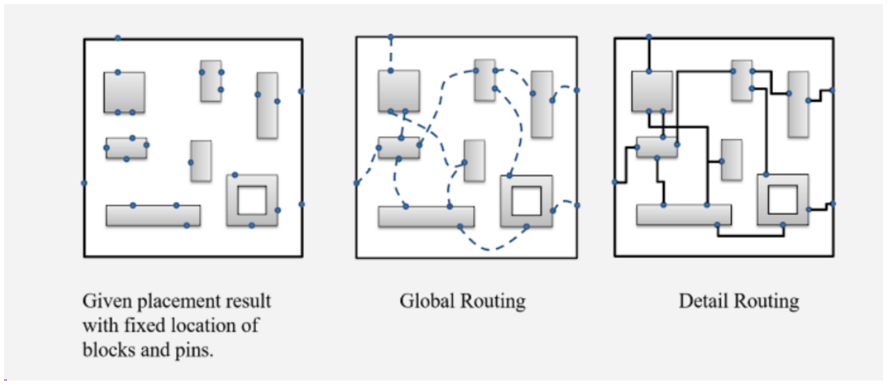
6.3) ROUTING

Routing in VLSI is making physical connections between signal pins, including power (VDD and VSS) and clock using metal layers. Following Clock Tree Synthesis (CTS) and optimization, the routing step determines the exact pathways for interconnecting standard cells, macros, and I/O pins. The layout creates electrical connections using metals and vias that are determined by the logical connections in the netlist (i.e; logical connectivity converted as physical connectivity).
CTS has information on all the cells, blockages, clock trees, buffers, inverters, and I/O pins that have been put in. The Routing program uses this data to electrically complete all of the connections defined in the netlist, ensuring that there are no DRC violations. The tool makes all the connections defined in the netlist in a way that:

a. The design is completely routed
b. There are minor LVS violations and SI breaches
c. There should be no or few congestion hotspots
d. The timing DRCs and QOR are met

- **Routing Flow :**




Given placement result with fixed location of blocks and pins.
Global Routing
Detail Routing

- **<u>Global Routing</u>**
  - first partitions the routing region into tiles/rectangles called global routing cells (gcells) and decides tile-to-tile paths for all nets while attempting to optimize some given objective function (e.g., total wire length and circuit timing), but doesn't make actual connections or assign nets to specific paths within the routing regions.
  - By default, the width of a gcells is same as the height of a standard cell and is aligned with the standard cell rows
  - Every gcell having the a number of horizontal routing resources and vertical routing resources.
  - Global routing assigns nets (logical connectivity not metal connectivity) to specific metal layers and global routing cells.

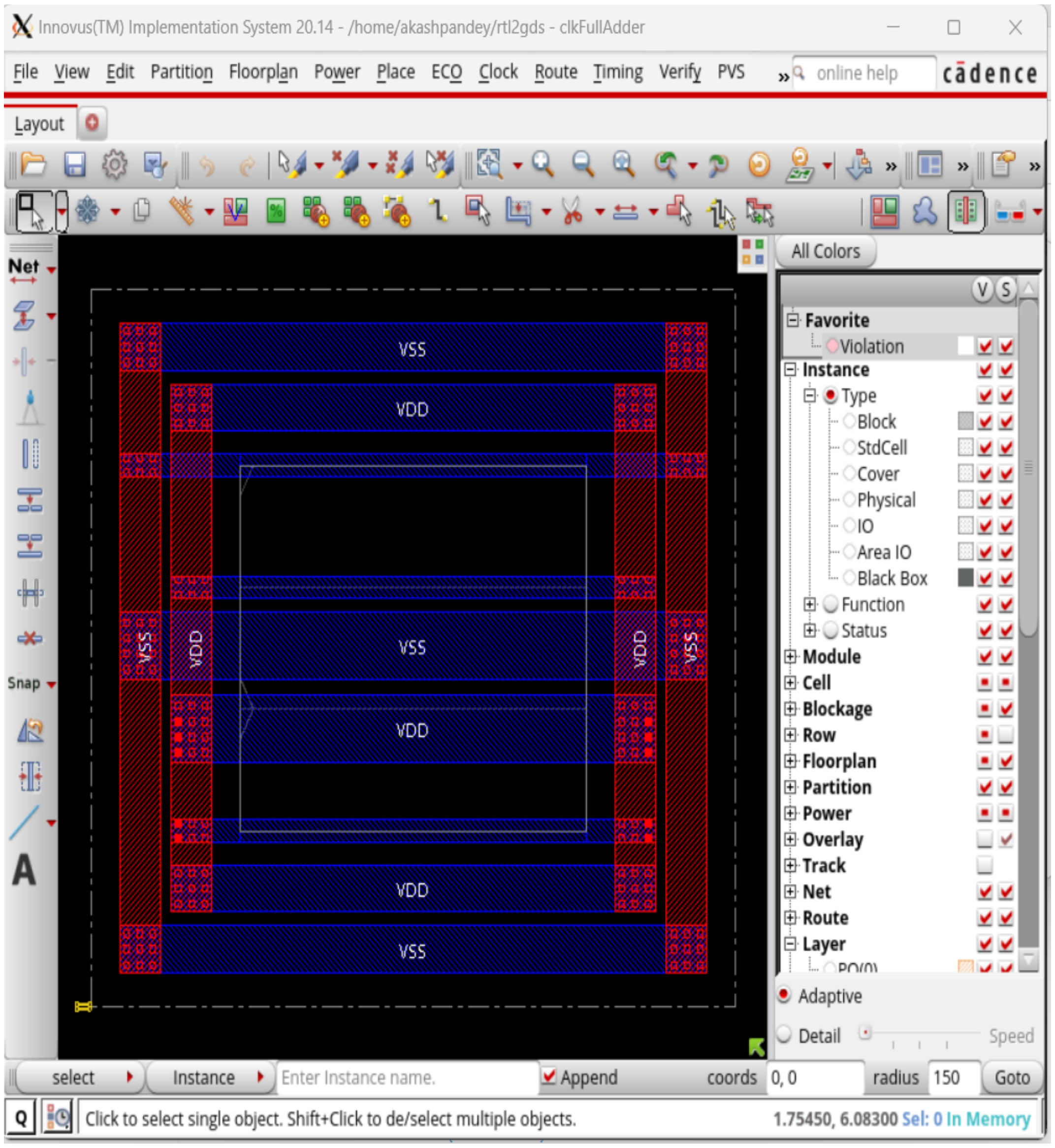
- **<u>Track Assignment</u>**
  - Track assignment is a stage wherein the routing tracks are assigned for each global routes. The tasks that are performed during this stage are as follows
    1. Assigning tracks in horizontal and vertical partitions.
    2. Rerouting all overlapped wires.
  - Track Assignment replaces all global routes with actual metal layers.
  - Although all nets are routed (not very carefully), there will be many DRC, SI and timing related violations, especially in regions where the routing connects the pins. These violations are fixed in the succeeding stages.

- **<u>Detail Routing</u>**
  - The detailed router uses the routing plan laid by the router during the Global Routing and Track Assignment and lays actually metal to logically connect pins with nets and other pins in the design.
  - The violations that were created during the Track Assignment stage are fixed through multiple iterations in this stage.

- The main goal of detailed routing is to complete all the required interconnect without leaving shorts or spacing violations.
- The detailed routing starts with the router dividing the block into specific areas called switch boxes or Sbox, which are generally expressed in terms of gcells.

➢ **<u>Search and Repair</u>**

- The search-and-repair stage is performed during detailed routing after the first iteration. In search-and-repair, shorts and spacing violations are located and rerouting of affected areas to fix all possible violation is executed.



Click to select single object. Shift+Click to de/select multiple objects.

1.75450, 6.08300 Sel: 0 In Memory

Fig.: Power Ring and Power strip Routing

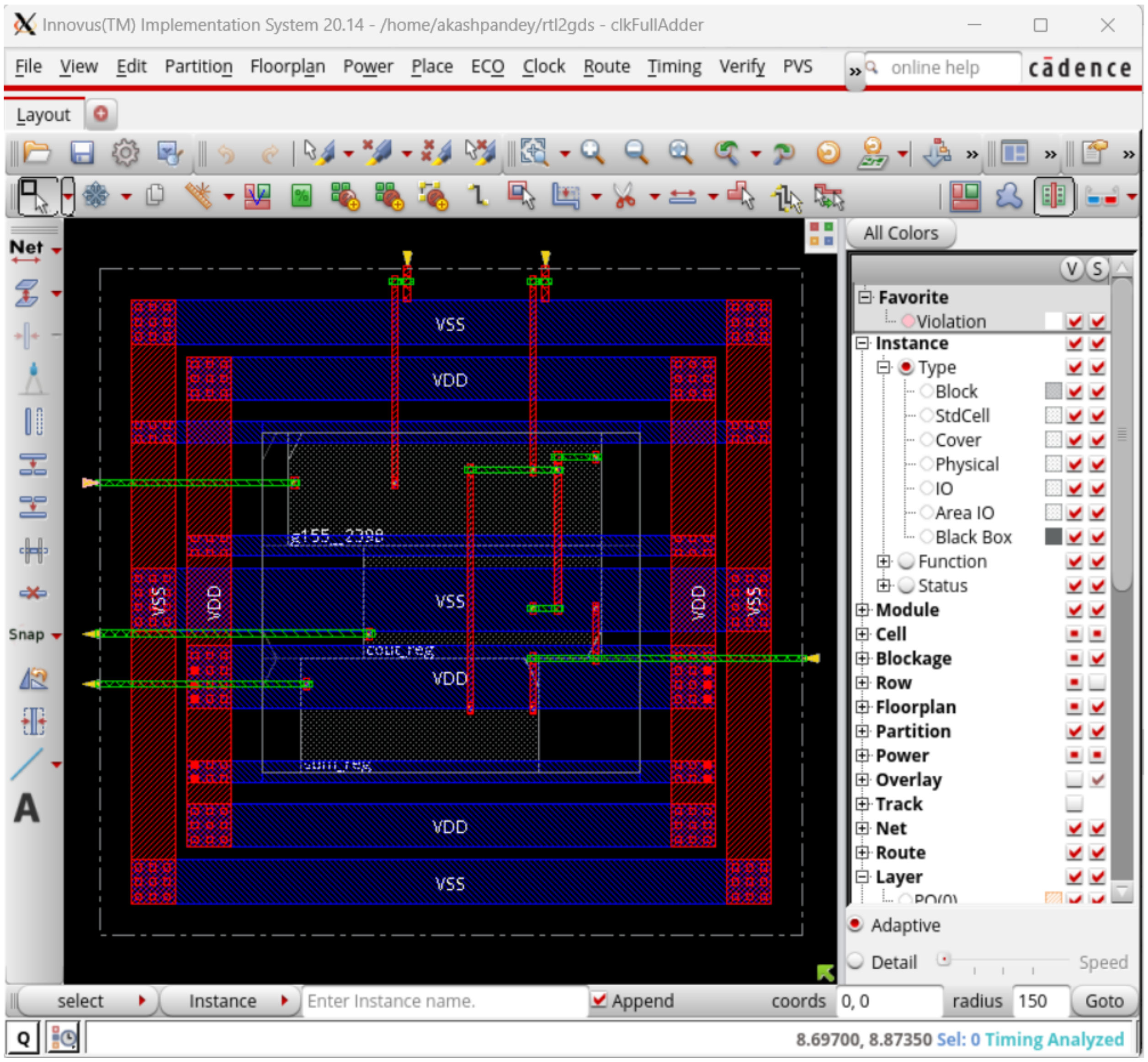
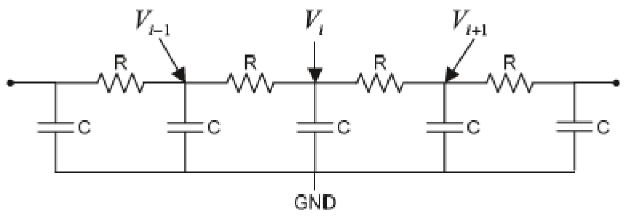


Fig.: Detailed Routing of signal and power nets in Innovus tool

6.4) CLOCK TREE SYNTHESIS

The concept of clock tree synthesis (CTS) is the automatic insertion of buffers/inverters along the clock paths of the ASIC design to balance the clock delay to all clock inputs. Naturally, clock signals are considered global (or ideal) nets.

These nets exhibit high resistance and capacitance due to their being very long wires. The principle of CTS is to reduce the RC delay associated with these long wires. These long wires can be modeled as distributed networks such that



As the number of segments increases, delay can be expressed as

$$t = \frac{rcL^2}{2},$$

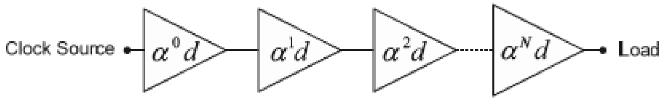
where *L* is the length of the wire, and *r* and are resistance and capacitance per unit length.

Delay after buffer insertion:

$$t = rc(\frac{L}{N})^2 + (N-1)t_b,$$

Interconnect is segmented into *N* equal sections, the wire propagation will be reduced quadratically, which is sufficient to offset the extra buffer delay that is introduced by repeaters/buffers.

to obtain optimal propagation delay, these buffers may be selected to monotonically increase in their drive strength *d* by a factor α for each level of clock tree path

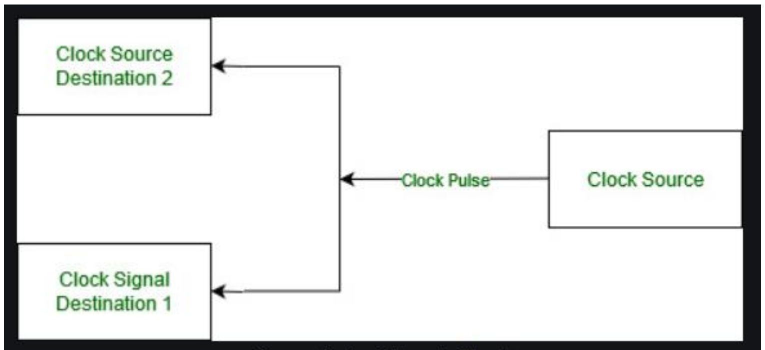


Cascaded Buffer

CLOCK SKEW:

In Synchronous circuits where all the logic elements share the same clock signal, it becomes imperative to design these elements as close to the clock source as possible because a system-on-chip, FPGA, CPLD contain Billions of transistors. Even though these distances are minute due to their sheer number there is a propagation delay which leads to the clock signal arriving at different parts of the chip at different times. This is called **Clock Skew**.

In Digital Circuit Design a “Sequentially Adjacent” circuit is one where if a pulse emitted from a common source is supposed to arrive at the same time. Using this definition, we can write a mathematical expression for clock skew as



Sequentially Adjacent Circuit

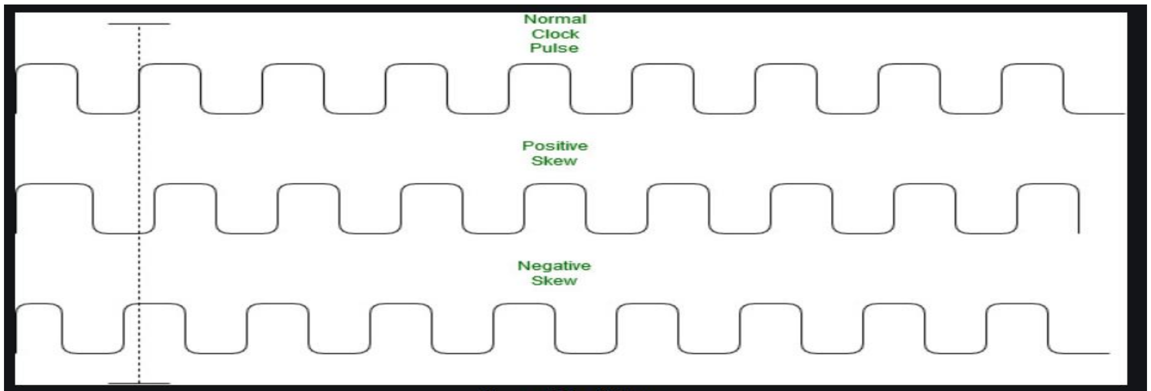
Clock skew Ts = Ta – Tb

Ta (Time of arrival of clock pulse at component a)

Tb (Time of arrival of clock pulse at component b)

Types of Clock Skew:

- **Positive Skew –**
  This occurs when the receiving register receives the clock pulse later than it is required.
- **Negative Skew –**
  This occurs when the receiving register receives the clock pulse earlier than required.



Types of Clock Skews

Factors causing Clock Skew:

- Interconnect Length
- Temperature Variations
- Capacitive Coupling
- Material Imperfections
- Differences in input capacitance on the clock inputs

7. REFERENCES

1. Chipedge - https://chipedge.com/
2. Wikipedia - https://en.wikipedia.org/wiki/
3. Digital Integrated Circuits: A Design Perspective
   Book by Anantha P. Chandrakasan, Borivoje Nikolic, and Jan M. Rabaey
4. CMOS digital integrated circuits: Analysis and design. 3rd. Edition. Sung-Mo Kang
   By: Kang, S.
   Contributor(s): Pebblelike, Y.
5. Area, Delay and Power Comparison of Adder Topologies
   International Journal of VLSI design & Communication Systems (VLSICS) Vol.3, No.1, February 2012