PROJECT ON RTL TO GDS2 FLOW

Submitted for requirements for the practical coursework

of

M Tech (VDN)

by

Akash Pandey (2302102035)
Ashreta Sahay (2302102033)
Rambabu Kumar (2302102024)
Sapan Kushwaha (2302102025)
Shivam Chaudhary (2302102032)
Shivam Vaish (2302102034)
Suvirn Upadhyay (2302102028)
Sushil Chaudhary (2302102027)

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 Oct2023 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)
Ashreta Sahay (2302102033)
Rambabu Kumar (2302102024)
Sapan Kushwaha (2302102025)
Shivam Chaudhary (2302102032)
Shivam Vaish (2302102034)
Suvirn Upadhyay (2302102028)
Sushil Chaudhary (2302102027)

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.

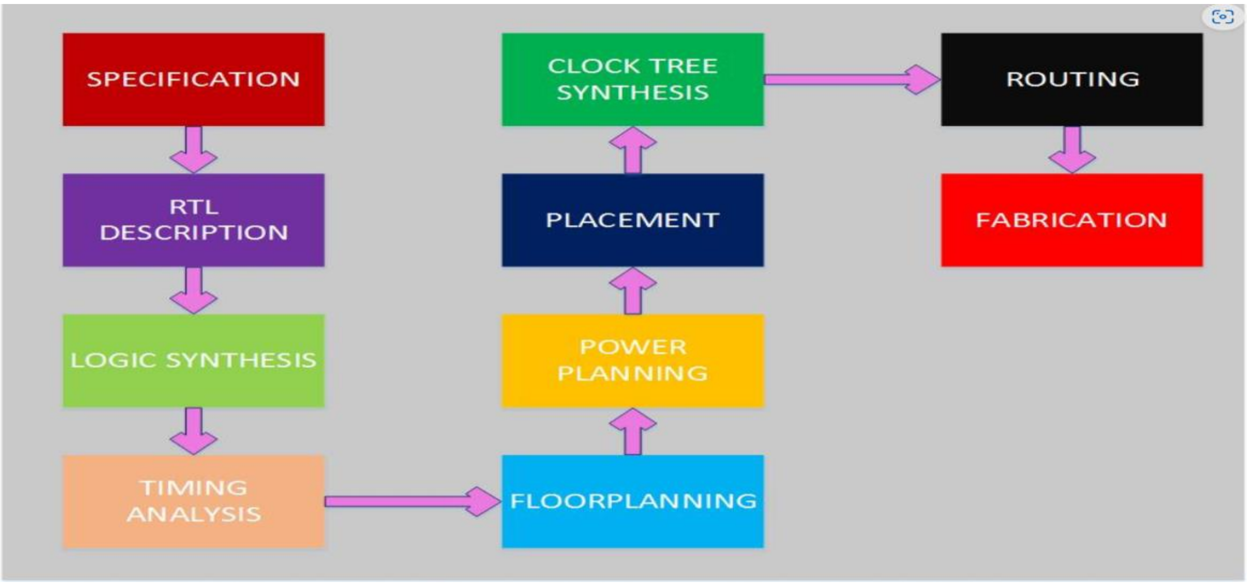


Fig.: RTL to GDS2 flow Block diagram

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.

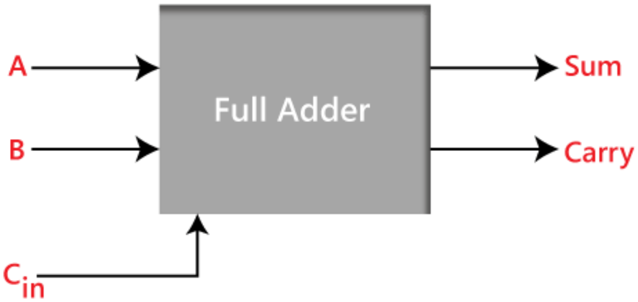
3. TABLE OF CONTENTS

1. INTRODUCTION

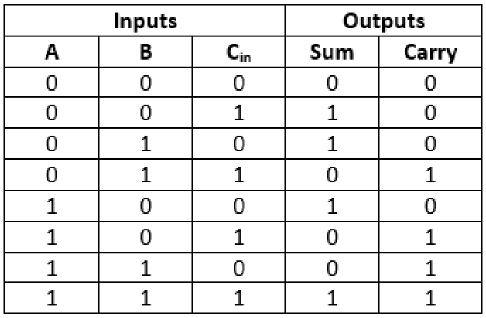
Full Adder

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



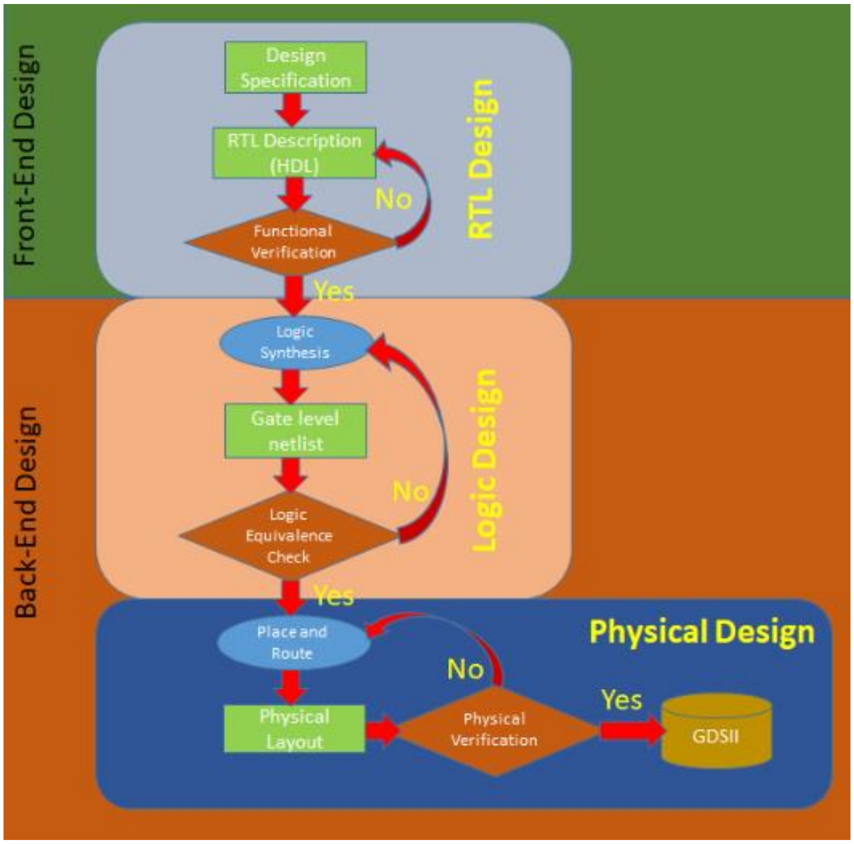
Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

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.

5.2) 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.

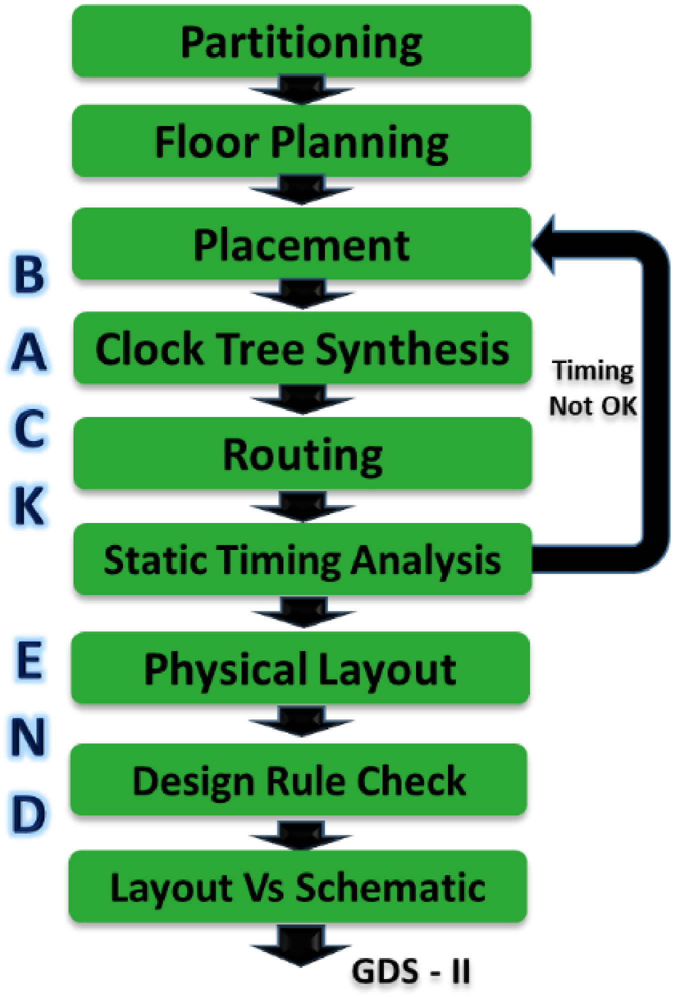
- **Place and Route (PnR):** Gate level netlist after DFT Insertion and SDC file is taken as input for the PnR and based on standard cells library, PnR starts. The goal of PnR stage is to place all the standard cells, Macros and I/O pads with minimal area, with minimal delay and Route them together in 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.

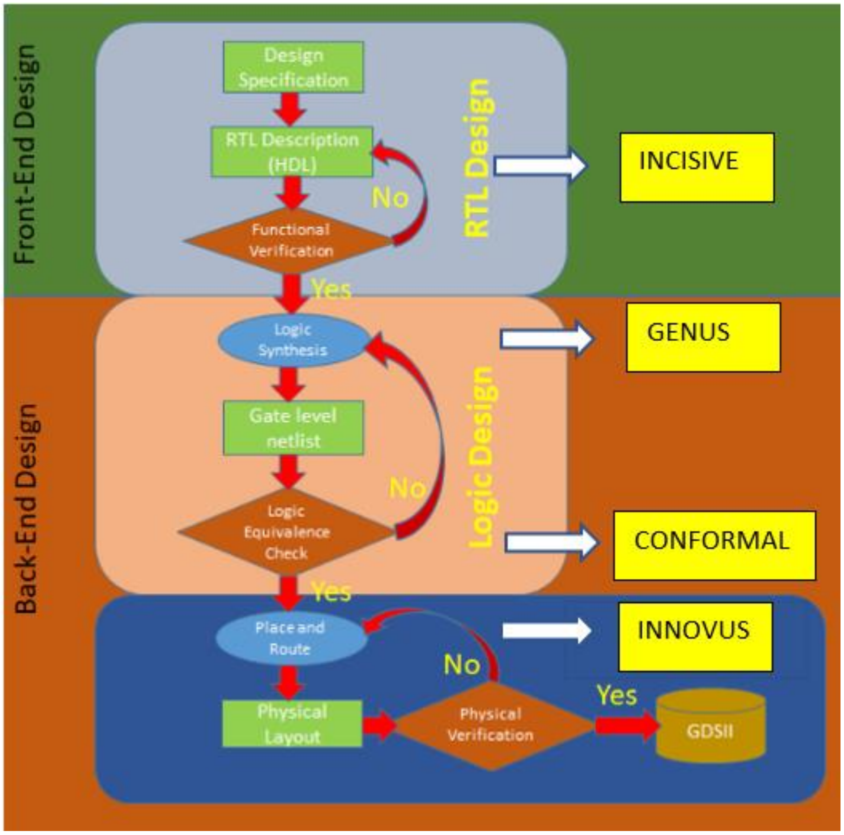
- **Signoff:** If there are some timing violations in post route design, we have a further stage called ECO (Engineering Change Order) where we can fix the timing violations. Apart from timing violation, there may be issues like IR Drop, DRC Violations all these are fixed in this stage and a final layout 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:



5.3) 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.

6. RTL DESIGN

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

6.1) 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 :**

```
module FA(sum,cout,a,b,cin,clk);
input a,b,cin,clk;
output reg cout;
output sum;
wire s1,c1,c2;
HA H1(s1,c1,a,b);
HA H2(sum,c2,s1,cin);
always @(posedge clk)
begin
cout =c1+c2;
end
endmodule
```

2. **Half adder code :**

```
module HA(s,c,a,b);
input a,b;
output reg s=0,c=0;
always @(*)
begin
```

```
s=a^b;
c=a&b;
end
endmodule
```

➢ Ripple Carry Adder :

- **Description**: The Ripple Carry Adder is a combinational circuit that can perform addition operation of two n-bit binary numbers. The carry signal produces “ripple” effect, that is the carry-out of each full adder is the carry of the succeeding next most significant full adder.

- **RTL Design:** At the RTL level, a RCA is designed using a series of full-adder cells. Each full-adder cell takes three inputs (A, B, and the carry-in) and produces two outputs (sum and carry-out).

- **Operation:** For an n-bit RCA, n full-adder cells are cascaded together to perform binary addition. It has a high propagation delay because of the sequential carry propagation.

- **Verilog Code :**

  1. **Main Code**

```
`timescale 1ns / 1ps
module ripple_carry_adder(x,y,s,cout,clk);
input [3:0] x,y;
input clk;
output [3:0] s;
output cout;
wire c1,c2,c3;

fulladder a1(x[0],y[0],1'b0,s[0],c1,clk);
fulladder a2(x[1],y[1],c1,s[1],c2,clk);
fulladder a3(x[2],y[2],c2,s[2],c3,clk);
fulladder a4(x[3],y[3],c3,s[3],cout,clk);
endmodule
```

2. **Full Adder code**

```
`timescale 1ns / 1ps
module fulladder(x,y,cin,s,co,clk);
input x,y,cin,clk;
output reg s,co;
always @(posedge clk)
begin
s= x^y^cin;
co= ((x^y)&cin)|(x&y);
end
endmodule
```

➢ Carry Look-Ahead Adder :

- **Description**: A Carry Look-Ahead Adder is designed to reduce the carry propagation delay seen in a RCA by precomputing carry signals.

- **RTL Design:** At the RTL level, a CLA is designed using blocks that generate carry signals for groups of bits based on their input values without having to wait for carries to propagate.

- **Operation:** It utilizes "generate" and "propagate" functions to generate carry signals for each group of bits independently, reducing the overall delay compared to a Ripple Carry Adder.

- **Verilog Code :**

1. **Main Code :**

```
module cla_adder(a,b,cin1,clk,carry,sum);
input[3:0] a,b;
input cin1,clk;
output [3:0] sum;
output carry;
wire [3:0] s,x,y;
wire cin,cout,p0,p1,p2,p3,g0,g1,g2,g3;
wire c1,c2,c3,c4;
```

```
 d_flip_flop A1(a[0],clk,x[0]);
 d_flip_flop A2(a[1],clk,x[1]);
 d_flip_flop A3(a[2],clk,x[2]);
 d_flip_flop A4(a[3],clk,x[3]);
 d_flip_flop A5(b[0],clk,y[0]);
 d_flip_flop A6(b[1],clk,y[1]);
 d_flip_flop A7(b[2],clk,y[2]);
 d_flip_flop A8(b[3],clk,y[3]);
 d_flip_flop A9(cin1,clk,cin);
 assign p0=x[0]^y[0],
        p1=x[1]^y[1],
        p2=x[2]^y[2],
        p3=x[3]^y[3];
 assign g0=x[0]&y[0],
        g1=x[1]&y[1],
        g2=x[2]&y[2],
        g3=x[3]&y[3];
assign c1=g0|p0&cin,
       c2=g1|p1&g0|p1&p0&cin,
       c3=g2|p2&g1|p2&p1&g0|p2&p1&p0&cin,
       c4=g3|p3&g2|p3&p2&g1|p3&p2&p1&g0|p3&p2&p1&p0&cin;
assign s[0]=p0^cin,
       s[1]=p1^c1,
       s[2]=p2^c2,
       s[3]=p3^c3;
assign cout=c4;
d_flip_flop B1(s[0],clk,sum[0]);
d_flip_flop B2(s[1],clk,sum[1]);
d_flip_flop B3(s[2],clk,sum[2]);
d_flip_flop B4(s[3],clk,sum[3]);
d_flip_flop B5(cout,clk,carry);
endmodule
```

2. D – Flipflop Code

```
module d_flip_flop(d,clk,q);
input d,clk;
output reg q;
always @(posedge clk)
begin
q=d;
end
endmodule
```

➢ Carry Skip Adder :

- **Description**: A Carry Skip Adder is designed to further reduce the delay by allowing certain groups of bits to skip the carry generation.

- **RTL Design:** At the RTL level, it combines CLA and additional logic to determine which groups of bits can safely skip the carry propagation.

- **Operation:** The CSKA employs CLA within groups of bits while also having a skip logic that can bypass carry calculations for groups of bits where the carry chain can be safely skipped, further reducing delay

➢ Carry Select Adder :

- **Description**: A Carry Select Adder is designed to provide two parallel adders with their carry chains, reducing delay but increasing area.

- **RTL Design:** At the RTL level, a CSA consists of two parallel adders and a multiplexer that selects the correct result based on a carry signal.

- **Operation:** It provides faster addition by allowing two addition operations to occur simultaneously and then selecting the result based on the carry signal generated in parallel.

- **Verilog Code :**

  1. **Main Code :**

```
module Carry_Select_Adder(
output [3:0] S,
output Cout,
input [3:0] a, b,
input Cin);
supply0 gnd;
supply1 vdd;
wire [3:0] c0, c1, S0, S1;
full_adder L00(a[0], b[0], gnd, S0[0], c0[0]);
full_adder L01(a[1], b[1], c0[0], S0[1], c0[1]);
full_adder L02(a[2], b[2], c0[1], S0[2], c0[2]);
full_adder L03(a[3], b[3], c0[2], S0[3], c0[3]);
full_adder L10(a[0], b[0], vdd, S1[0], c1[0]);
full_adder L11(a[1], b[1], c1[0], S1[1], c1[1]);
full_adder L12(a[2], b[2], c1[1], S1[2], c1[2]);
full_adder L13(a[3], b[3], c1[2], S1[3], c1[3]);
mux_2to1 m1(Cin, S0[0], S1[0], S[0]);
mux_2to1 m2(Cin, S0[1], S1[1], S[1]);
mux_2to1 m3(Cin, S0[2], S1[2], S[2]);
mux_2to1 m4(Cin, S0[3], S1[3], S[3]);
mux_2to1 m5(Cin, c0[3], c1[3], Cout);
endmodule
```

2. Full Adder Code

```
module Full_adder(input a, b, cin, output sum, carry);
wire x, y, z;
Half_adder h1 (a, b, x, y);
Half_adder h2 (cin, x, sum, z);
or (carry, y, z);
endmodule
```

3. Half Adder Code

```
module Half_adder(input a, b, output sum, carry);
assign sum = a ^ b;
assign carry = a &amp; b;
endmodule
```

4. 2X1 Multiplexer Code

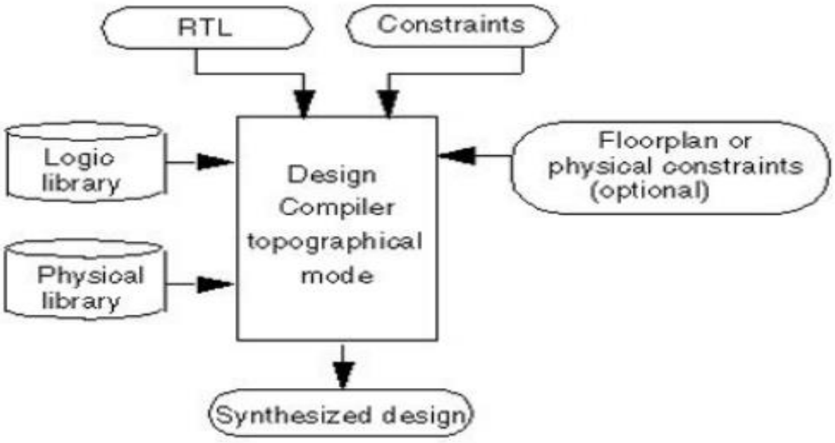
```
module mux_2to1(input sel,
input i0, i1,
output y);
assign y= sel? i0 : i1;
endmodule
```

7. SYNTHESIS

7.1) 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
Physical library
Design Compiler topographical mode
Floorplan or physical constraints (optional)
Synthesized design

7.2) 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.

8. FLOORPLANNING

8.1) PARTITIONING

A chip may contain several million transistors. Due to the limitations of memory space and computation power available it may not be possible to layout the entire chip (or generically speaking any large circuit) in the same step. Therefore, the chip (circuit) is normally partitioned into sub-chips (sub-circuits). These sub-partitions are called blocks.

Partitioning is a process of dividing the chip into small blocks. This is done mainly to separate different functional blocks and also to make placement and routing easier. Partitioning can be done in the RTL design phase when the design engineer partitions the entire design into sub-blocks and then proceeds to design each module. These modules are linked together in the main module called the TOP LEVEL module. The actual partitioning process considers many factors such as the size of the blocks, number of blocks, and number of interconnections between the blocks

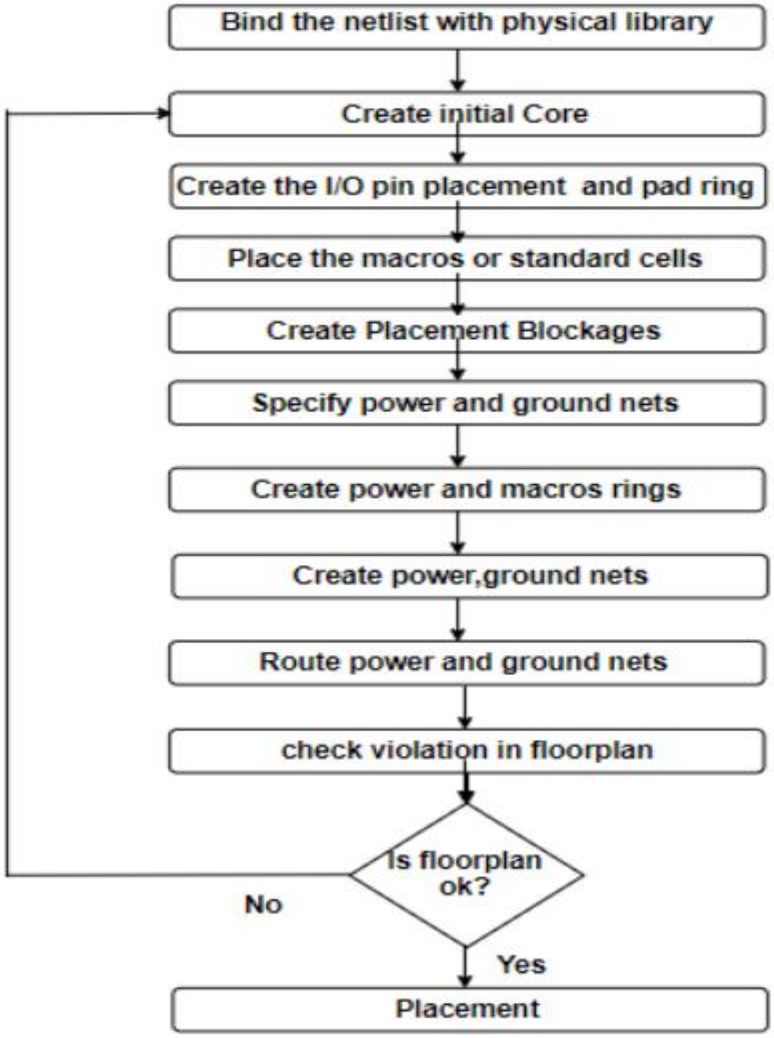
The output of partitioning is a set of blocks and the interconnections required between blocks. The goal of partitioning is to split the circuit such that the number of connections between partitions is minimized. . In large circuits, the partitioning process is hierarchical and at the topmost level a chip may have 5 to 25 blocks. Each block is then partitioned recursively into smaller blocks.

8.2) 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

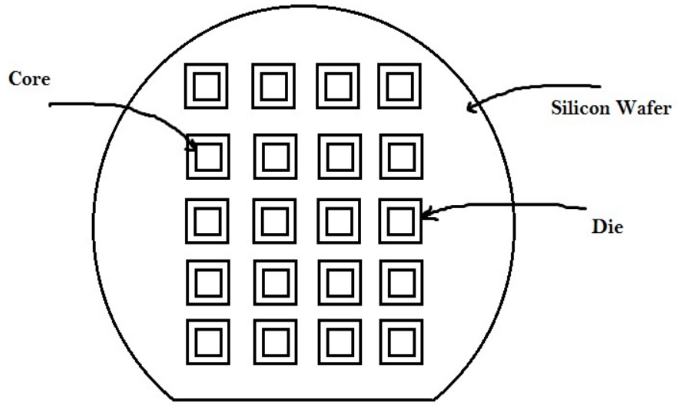
The following flowchart lists the steps involved in floorplanning process, will be explained in detail in subsequent sub-sections :


Bind the netlist with physical library
Create initial Core
Create the I/O pin placement and pad ring
Place the macros or standard cells
Create Placement Blockages
Specify power and ground nets
Create power and macros rings
Create power,ground nets
Route power and ground nets
check violation in floorplan
Is floorplan ok?
No
Yes
Placement

-

- **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'.



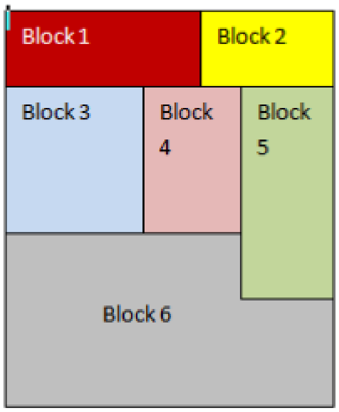
1. **Aspect ratio**: Aspect ratio will decide the size and shape of the chip. It is the ratio between horizontal routing resources to vertical routing resources (or) ratio of height and width.

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

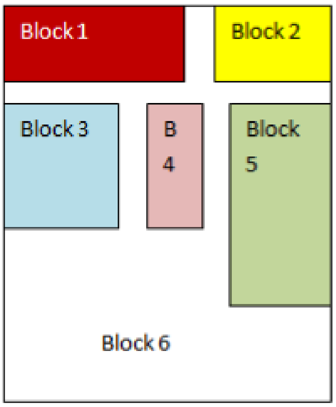
2. **Core utilization**: Utilization will define the area occupied by the standard cells, macros, and other cells. If core utilization is 0.8 (80%) that means 80% of the core area is used for placing the standard cells, macros, and other cells, and the remaining 20% is used for routing purposes.

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

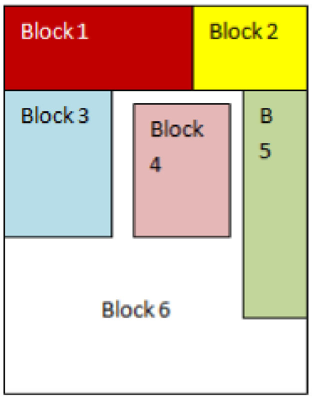
- **Types of Floorplan Techniques:**
    - Abutted floorplan : Channel less placement of blocks.
    - Non – Abutted Floorplan : Channel based placement of blocks.
    - Mix of both : Partially abutted with some channels.



Abutted



Non abutted



Mixed

- **Placement of macros**

A Macro cell is a cell without pre-defined dimensions. This term may also refer to a large physical layout, possibly containing millions of transistors, e.g., an SRAM or CPU core, and possibly having discrete dimensions, that can be incorporated into the IC physical design.
Macros and standard cells are placed and fixed at different stages in physical design flow. Macros are placed at floorplan stage where as standard cells are placed in placement stage. Some guidelines and recommendations for macro placement are :

1. Place macros around chip periphery
2. Consider connections to fixed cells when placing macros
3. Orient macros to minimize the distance between pins
4. Reserve enough room around macros (Minimize congestion)
5. Reduce open fields as much as possible (remove dead space to increase the area for random logic)
6. Reserve space for power grid

8.3) PLACEMENT

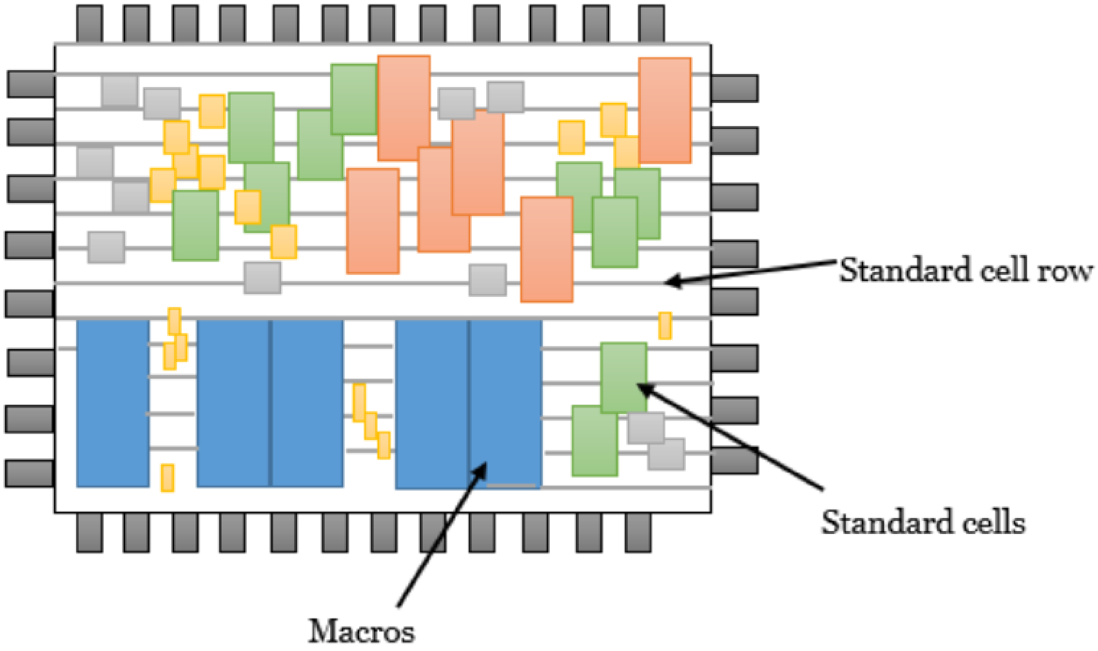
Placement is the process of finding a suitable physical location for each cell in the block. The tool only determine the location of each standard cell on the die. Placement does not just place the standard cell available in the synthesized netlist, it also optimized the design.

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.

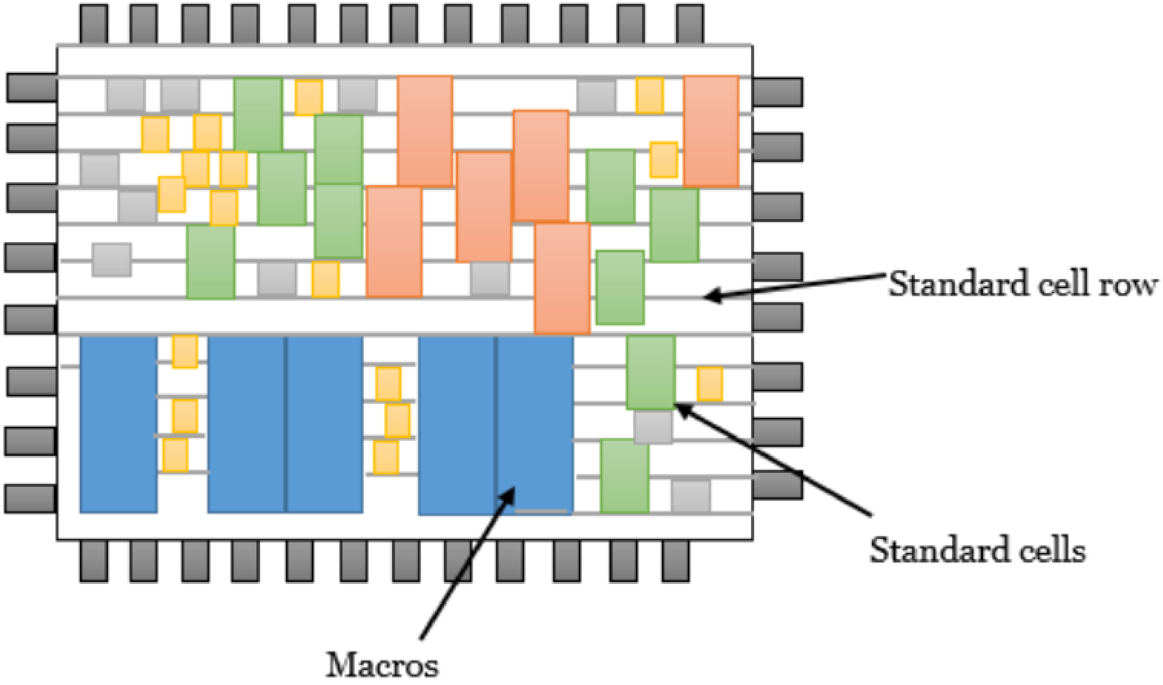
- **Coarse Placement**

  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.



- **Placement constraints**

  Placement constraints provide guidance during placement and placement optimization and legalization so that congestion and timing violations will be reduced.

  - **<u>Placement blockages</u>**

    It is the area where the cells must avoid during placement, optimization and legalization.

    It can be hard and soft.

    ICC tools supports two types of placement blockages

    a. <u>Keep-out margin</u> : it is a region around the boundary of fixed cells in a block in which no other cells are placed. The width of the keep-out margin on each side of the fixed cell can be the same or different. Keeping the placement of cells out of such regions avoids congestion and net detouring and produces better QOR (quality of results).

    b. <u>Area-based placement blockage:</u> It is a rectangular region in which cells can be placed or not or we can limit the number of cells.

Hard blockages prevent the placement of standard cells being placed in the blockage area. Soft blockages allow buffer/inv to be placed in that blockage area. Partial placement blockages limits the cell density in a particular region. For example, 40 % area is blocked for placement of standard cells and rest of the 60% available for placement of standard cells)

- **Placement bounds :**
  It is a constraint that controls the placement of groups of leaf cells and hierarchical cells. It allows you to group cells to minimize wire length and place the cells at most appropriate locations. When our timing is critical during placement then we create bounds in that area where two communicating cells are sitting far from another. It is a fixed region in which we placed a set of cells. It comprises of one or more rectangular or rectilinear shapes which can be abutted or disjoint. In general we specify the cells and ports to be included in the bound. If a hierarchical cell is included, all cells in the sub-design belong to the bound.

  a. Soft move bound
     In this tool tries to place the cells in the move bound within a specified region, however, there is no guarantee that the cells are placed inside the bounds.

  b. Hard move bound
     In this tool must place the cells in the move bound within a specified region.

  c. Exclusive move bound
     In this tool tries to place the cells in the group bound within a floating region, however, there is no guarantee that the cells are placed inside the bounds

- **Density constraint**
  It means how the density of cells can be packed. We can control the overall placement density for the block or the cell density for specific regions. To control the cell density for specific regions we can also use partial placement blockages.

8.4) 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).

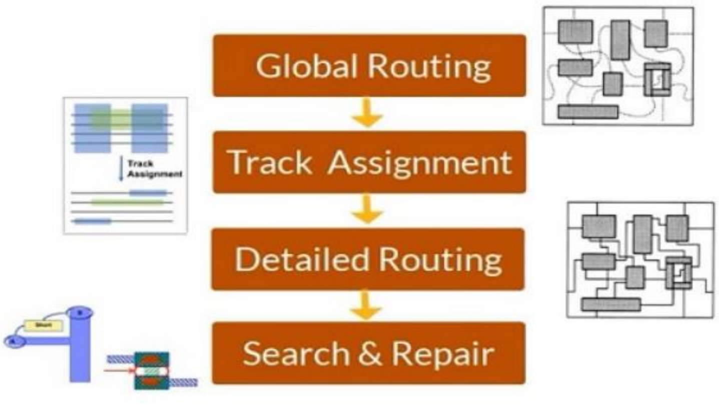
CTS has information on all the cells, blockages, clock trees, buffers, inverters, and I/O pins that have been put in. The Routing program uses this data to electrically complete all of the connections defined in the netlist, ensuring that there are no DRC violations. The tool makes all the connections 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

- **Inputs of Routing :**
  1. Netlist with position of blocks and location of pins (CTS completed)
  2. Timing budget for critical net
  3. Technology file
  4. TLU+ file
  5. SDC

- **Outputs of Routing :**
  1. Geometric layout of all nets (.GDS)
  2. .spef file
  3. .sdc file

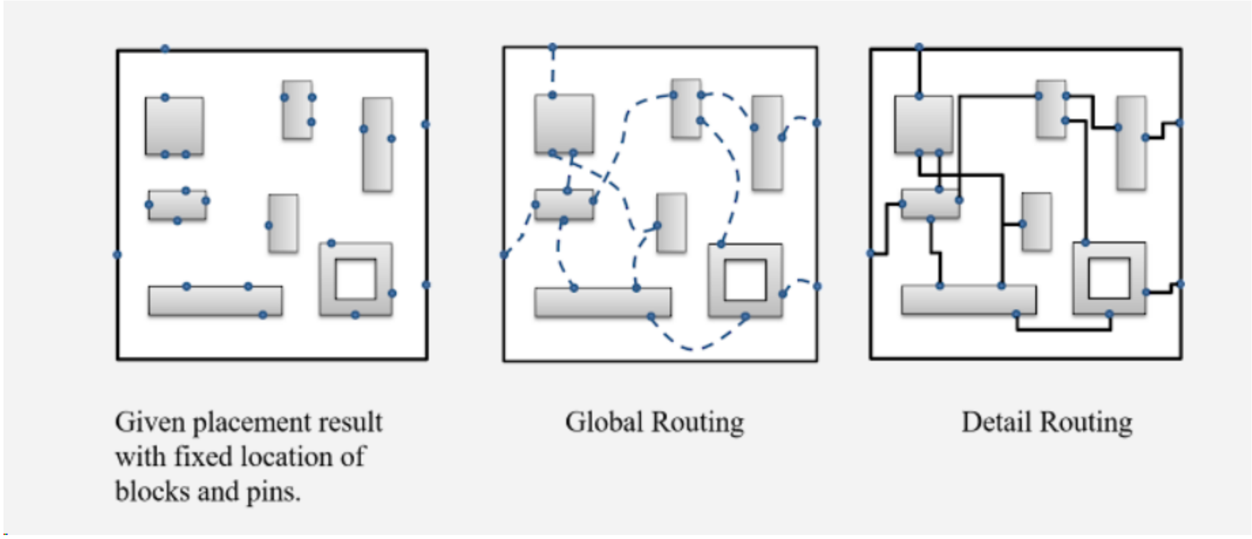
- **Routing Flow :**



- **Routing operation stages**

  Each metal layer in a grid-based routing system has its tracks and preferred routing direction, which are described in a unified cell in the standard cell library. Routing activities are divided into four steps:

  1. Global route
  2. Track Assignment
  3. Detail Routing
  4. Search and repair


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.

➢ **Track Assignment**

- 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.

➢ **Detail Routing**

- 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.

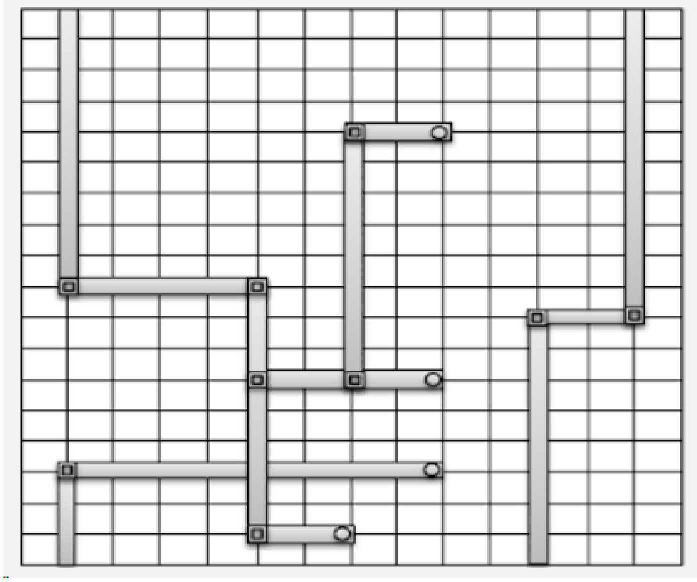
➢ **Search and Repair**

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.

- **Types of Detail Routing :**

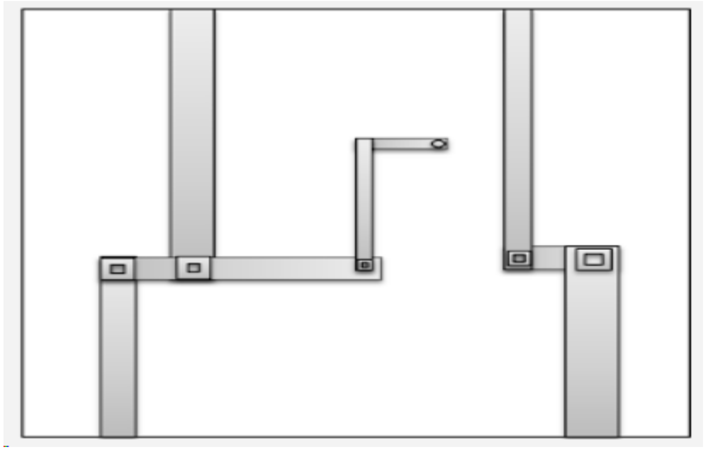
a. **Grid-Based Routing**

For grid-based routing, a routing grid is superimposed on the routing region, and then the detailed router finds routing paths in the grid. The space between adjacent grid lines is called wire pitch, which is defined in the technology file and is larger than or equal to the sum of the minimum width and spacing of wires. Switching from layer to layer is allowed only at the intersection of vertical and horizontal grid lines. In this way, the wires with the minimum width following the path in the grid would automatically satisfy the design rules. Therefore, grid-based detailed routing is much more efficient and easier for implementation.



b. **Grid-Less Routing**

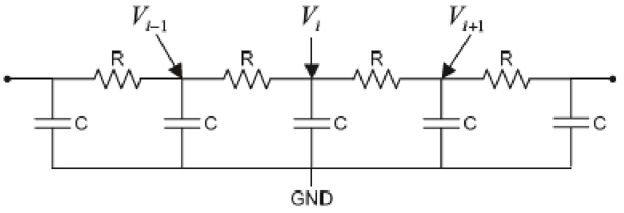
A gridless detailed router does not follow the routing grid and thus can use different wire widths and spacing.
It can handle variable widths and spacing for wires and is, thus, more suitable for interconnect tuning optimization, such as wire sizing.
However, gridless detailed routing is generally much slower than the grid-based one because of its higher complexity.



9.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

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

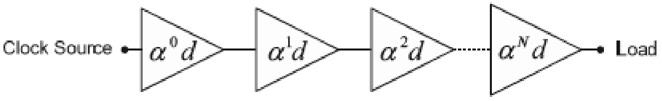
where *L* is the length of the wire, and *r* and are resistance and capacitance per unit length.

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

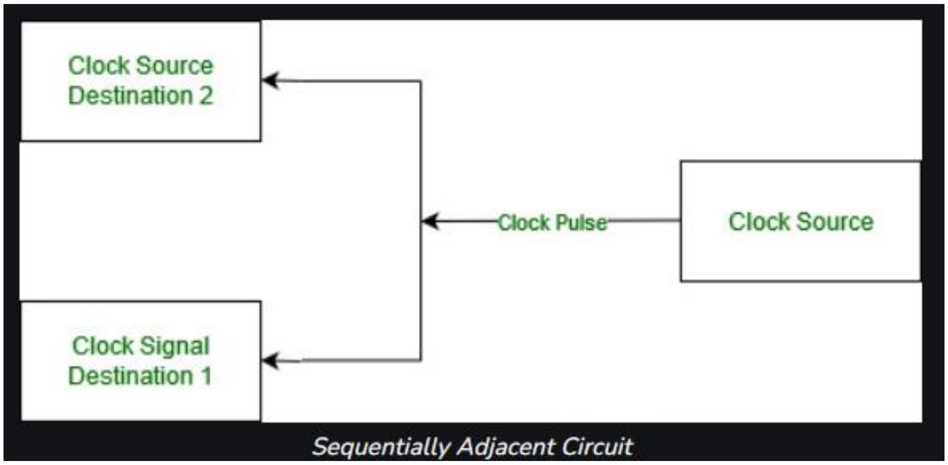


Cascaded Buffers

9.1 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

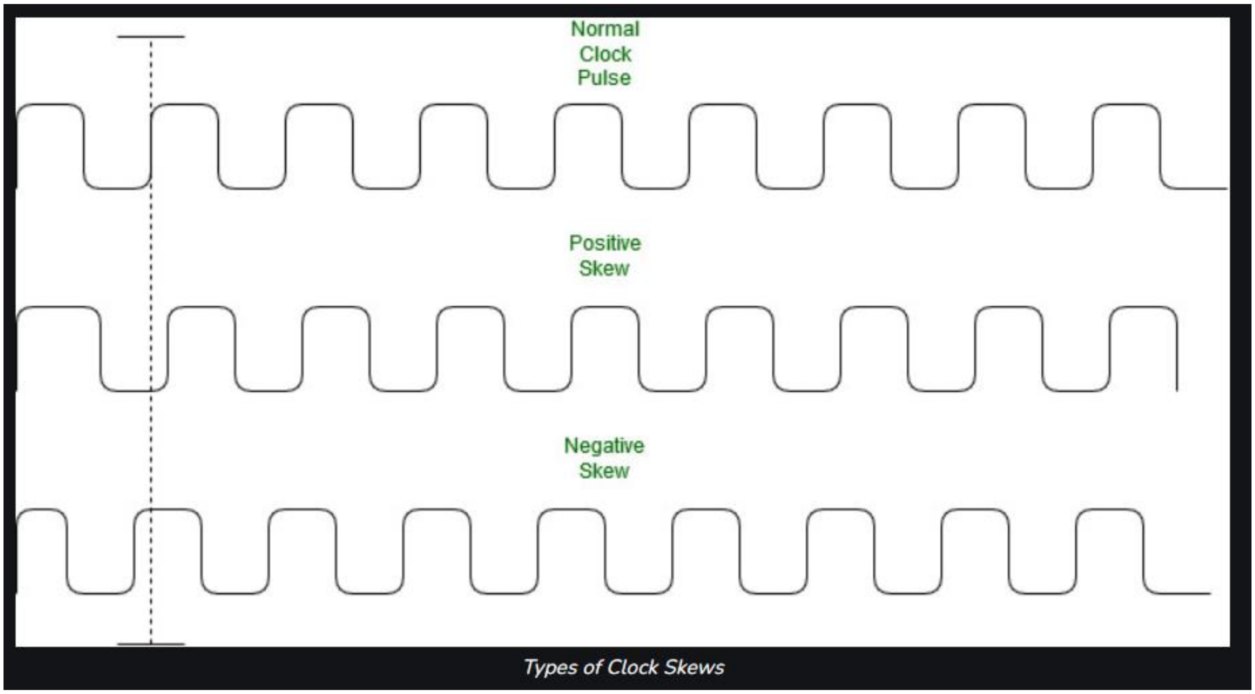
Ta (Time of arrival of clock pulse at component a)

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:

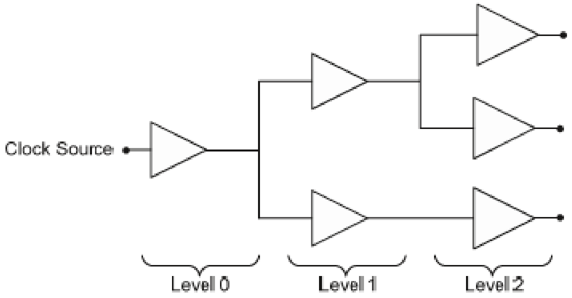
- Interconnect Length
- Temperature Variations
- Capacitive Coupling
- Material Imperfections
- Differences in input capacitance on the clock inputs

9.2 CLOCK TREE GENERATION ALGORITHMS

Most clock tree synthesis algorithms use either the Sum or Pi configuration during the clock buffer insertion along each clock path.

- **Sum configuration**

  In the Sum configuration, the total number of inserted buffers is the sum of all buffers per level. This type of structure exhibits unbalanced trees due to the different number of buffers and wire lengths. In this method, the systematic skew is minimized through delay matching along each clock path and, due to its non-symmetrical nature, is highly process-corner-dependent.

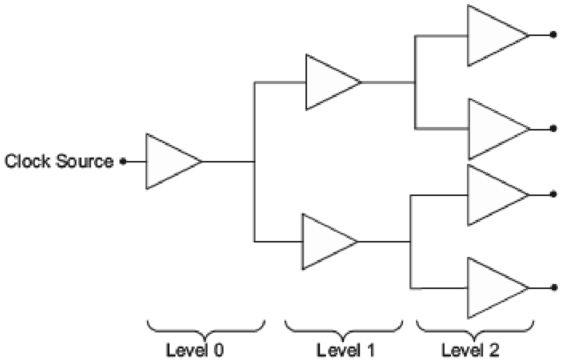


In this configuration, the total number of clock buffers is given by

$$N_{total} = n_{level0} + n_{level1} + n_{level2} + \ldots\ldots + n_{leveln} .$$

- **Pi configuration**

  In the Pi configuration, the total number of buffers inserted along clock paths is a multiple of the previous level. This type of structure uses the same number of buffers and geometrical wires and relies on matching the delay components at each level of the clock tree. The Pi structure clock tree is considered to be balanced, or symmetrical.



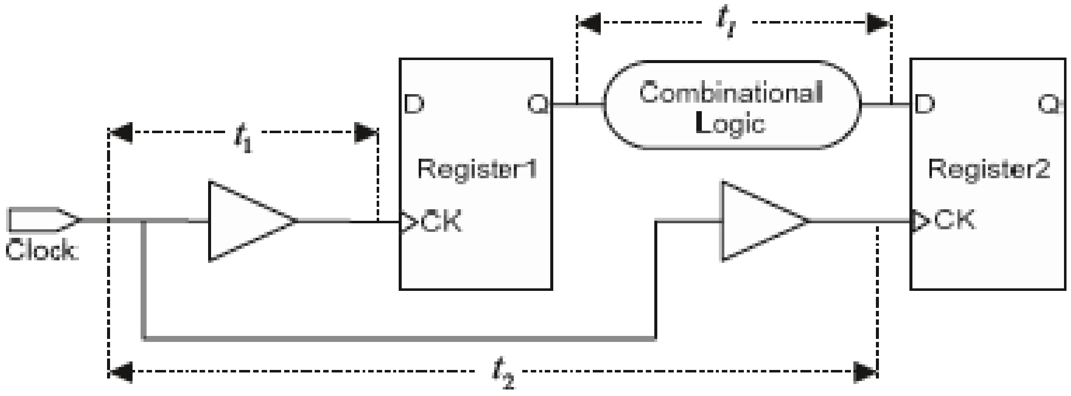
$$N_{total} = (n_{level0} \times n_{level1}) + (n_{level1} \times n_{level2}) + .. + (n_{level(n-1)} \times n_{leveln})$$

9.3 OPTIMIZATION IN BUFFER INSERTION

Regardless of which configuration is used to insert buffers during clock tree synthesis, the objective that governs the clock tree quality is achieving minimal skew while maintaining an acceptable clock signal propagation delay. The difference of the leaf registers' clock or skew with respect to the source of clock can be expressed as

$$\delta = t_2 - t_1,$$

where δ is the clock skew between two leaf registers (leaf registers are connected at the last level of clock tree) with the propagation delay clock of t1 and t2 along two different clock paths, as shown in Figure



To prevent any error in synchronization following condition must be satisfied:

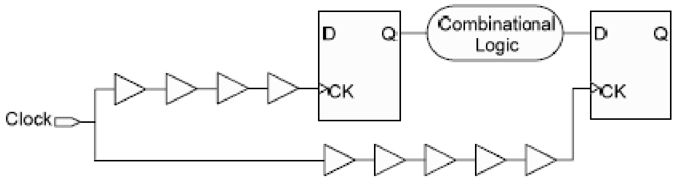
$$\delta \le t_l .$$

$$T \ge t_l - \delta ,$$

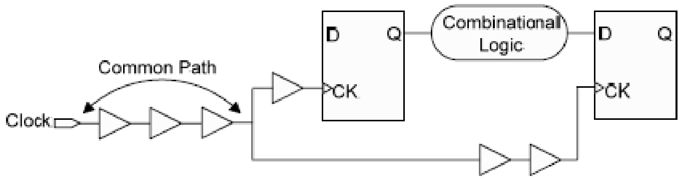
On Chip Variation Awareness:

One of the common issues that can be affected by on-chip variation on a clock tree is that if the clock tree synthesis algorithm branches the clock paths near the clock source rather than close to the leaf cells, the clock can be skewed.

This means that a CTS algorithm must be able to use a common path during buffer insertion as much as possible. In this situation, the delay differences along each clock path will be local and the delay through the common path will not contribute to clock skews due to on-chip variation.



Without On-chip Variation Awareness



With On-chip Variation Awareness

10. TIMING ANALYSIS

Static timing analysis (STA) is a method of validating the timing performance of a design by checking all possible paths for timing violations. STA breaks a design down into timing paths, calculates the signal propagation delay along each path, and checks for violations of timing constraints inside the design and at the input/output interface.

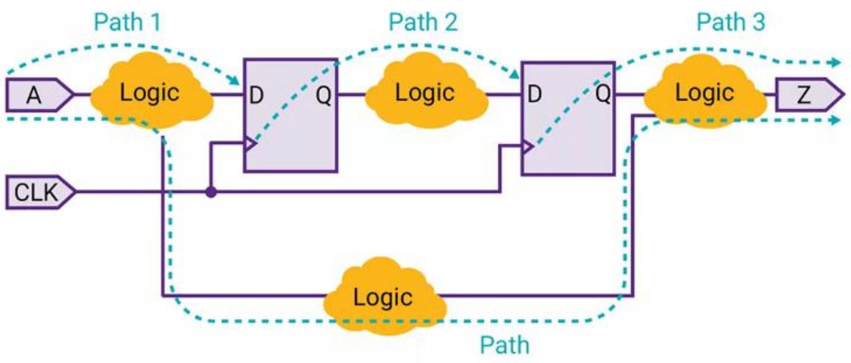
When performing timing analysis, STA first breaks down the design into timing paths. Each timing path consists of the following elements:

Start point: The beginning of a timing route in which data is launched by a clock edge or must be ready at a certain moment. Every start point must be a register clock pin or an input port.

Combinational logic network: It includes elements with no internal state or memory. AND, OR, XOR, and inverter elements are allowed in combinational logic, but flip-flops, latches, registers, and RAM are not.

Endpoint: When data is caught by a clock edge or when it needs to be provided at a specified moment, this is the end of a timing path. A register data input pin or an output port must be present at each endpoint.

Timing paths

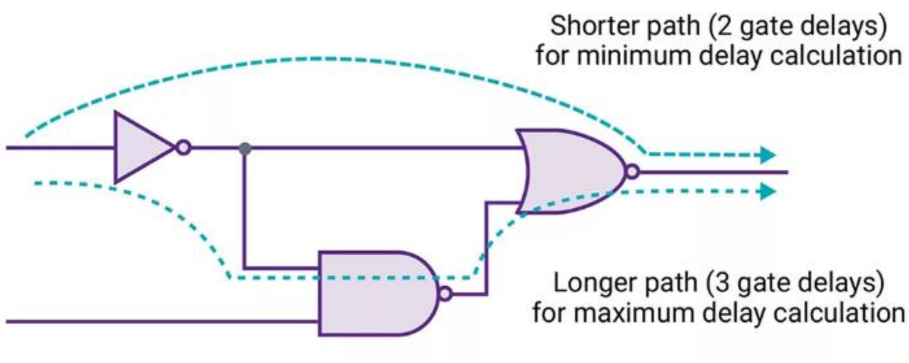


In the example, each logic cloud represents a combinational logic network. Each path starts at a data launch point, passes through some combinational logic, and ends at a data capture point.

Path	Startpoint	Endpoint
Path 1	Input port	Data input of a sequential element
Path 2	Clock pin of a sequential element	Data input of a sequential element
Path 3	Clock pin of a sequential element	Output port
Path 4	Input port	Output port

A combinational logic cloud might contain multiple paths, as shown in the following figure. STA uses the longest path to calculate a maximum delay and the shortest path to calculate a minimum delay.

Multiple paths through combined logic


Shorter path (2 gate delays)
for minimum delay calculation
Longer path (3 gate delays)
for maximum delay calculation

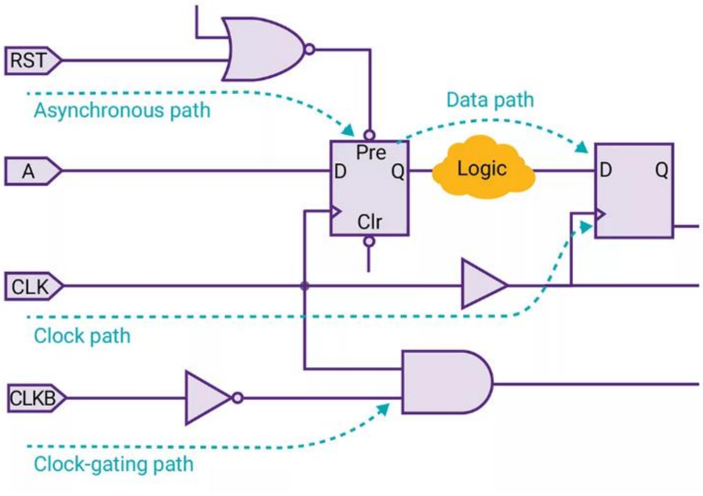
STA also considers the following types of paths for timing analysis:

- **Clock path.** A path from a clock input port or cell pin, through one or more buffers or inverters, to the clock pin of a sequential element; for data setup and hold checks.

- **Clock-gating path.** A path from an input port to a clock-gating element; for clock-gating setup and hold checks.

- **Asynchronous path.** A path from an input port to an asynchronous set or clear pin of a sequential element; for recovery and removal checks.

Types of paths considered for timing analysis



After breaking down a design into a set of timing paths, an STA tool calculates the delay along each path. The total delay of a path is the sum of all cell and net delays in the path.

- **Cell Delay:** It is the amount of delay from input to output of a logic gate in a path. In the absence of back-annotated delay information from an SDF file, the tool calculates the cell delay from delay tables provided in the logic library for the cell.

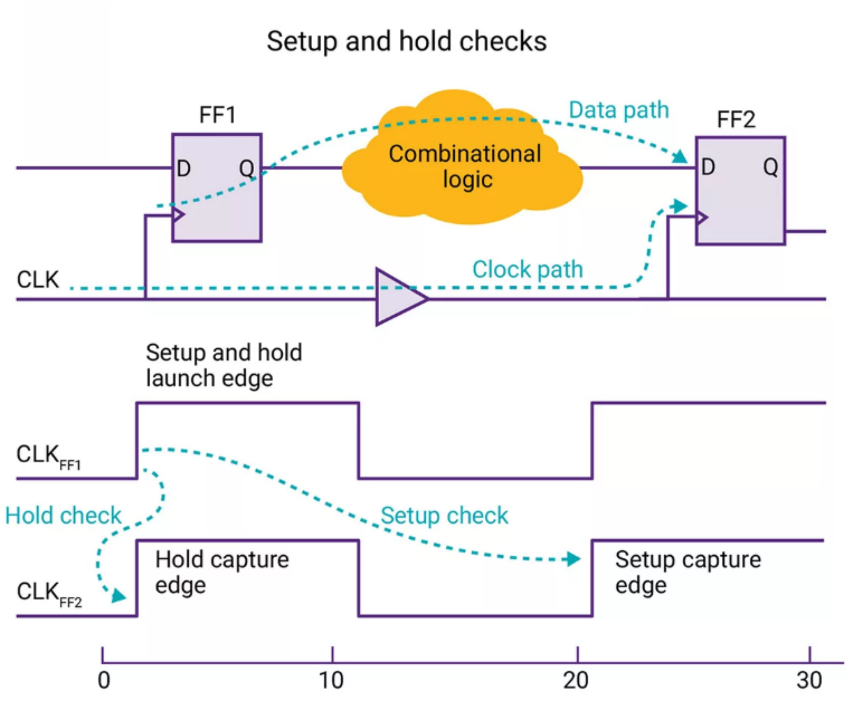
  Typically, a delay table lists the amount of delay as a function of one or more variables, such as input transition time and output load capacitance. From these table entries, the tool calculates each cell delay.

- **Net Delay:** It is the amount of delay from the output of a cell to the input of the next cell in a timing path. This delay is caused by the parasitic capacitance of the interconnection between the two cells, combined with net resistance and the limited drive strength of the cell driving the net.

  STA then checks for violations of timing constraints, such as **setup and hold constraints**:

- A setup constraint specifies how much time is necessary for data to be available at the input of a sequential device before the clock edge that captures the data in the device. This constraint enforces a maximum delay on the data path relative to the clock edge.
- A hold constraint specifies how much time is necessary for data to be stable at the input of a sequential device after the clock edge that captures the data in the device. This constraint enforces a minimum delay on the data path relative to the clock edge.

  The following example shows how STA checks setup and hold constraints for a flip-flop:



For this example, assume that the flip-flops are defined in the logic library to have a minimum setup time of 10-time units and a minimum hold time of 0.0-time units. The clock period is defined in the tool to be 10-time units. The time unit size, such as ns or ps, is specified in the logic library.

By default, the tool assumes that signals are propagated through each data path in one clock cycle. Therefore, when the tool performs a setup check, it verifies that the data launched from FF1 reaches FF2 within one clock cycle, and arrives at least 1.0-time unit before the data gets captured by the next clock edge at FF2. If the data path delay is too long, it is reported as a timing violation. For this setup check, the tool considers the longest possible delay along the data path and the shortest possible delay along the clock path between FF1 and FF2.

When the tool performs a hold check, it verifies that the data launched from FF1 reaches FF2 no sooner than the capture clock edge for the previous clock cycle. This check ensures that the data already existing at the input of FF2 remains stable long enough after the clock edge that captures data for the previous cycle. For this hold check, the tool considers the shortest possible delay along the data path and the longest possible delay along the clock path between FF1 and FF2. A hold violation can occur if the clock path has a long delay.

If certain paths are not intended to operate according to the default setup and hold behavior assumed by the STA tool, you need to specify those paths as timing exceptions. Otherwise, the tool might incorrectly report those paths as having timing violations.

An STA tool may let you specify the following types of exceptions:

- **False path.** A path that is never sensitized due to the logic configuration, expected data sequence, or operating mode.
- **Multicycle path.** A path designed to take more than one clock cycle from launch to capture.
- **Minimum or maximum delay path.** A path that must meet a delay constraint that you explicitly specify as a time value.

11. POWER ANALYSIS

Power analysis of design can be reconnoitered at several levels such as system level, architectural level, logic level, circuit level and device level.

11.1 BASICS OF POWER DISSIPATION

In CMOS circuits there are mainly three types of power dissipation namely average power dissipation, short circuit power dissipation and leakage power dissipation.

1. AVERAGE POWER DISSIPATION

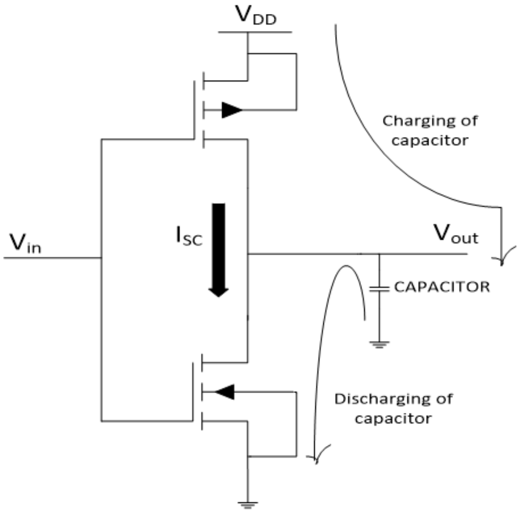
This power is also known as switching or dynamic power dissipation. The significant cause of this power dissipation is switching or we can say that it occurs during charging and discharging of capacitance as shown in “Fig. 3”. Dynamic power can be obtained from the formula as in equation 1.

$$P_{Dyn} = C_L V_{DD} \alpha f$$

Where C_L is load capacitor, V_{DD} is supply voltage, α is switching activities factor of a circuit and f is the clock frequency. To evaluate average switching power in charging and discharging of a capacitor we have to integrate the power required to charge the capacitor from V_{DD} through PMOS and discharging to ground through NMOS while applying the input voltage having zero rise and fall time of time period T.

2.SHORT CIRCUIT POWER DISSIPATION

During switching there is some small time when both PMOS and NMOS are ON and there is short circuit current flow from V_{DD} to ground as there exist a path between them as shown in “Fig. 3”. This power dissipation arises because practically input voltage pulse has some finite rise and fall time.

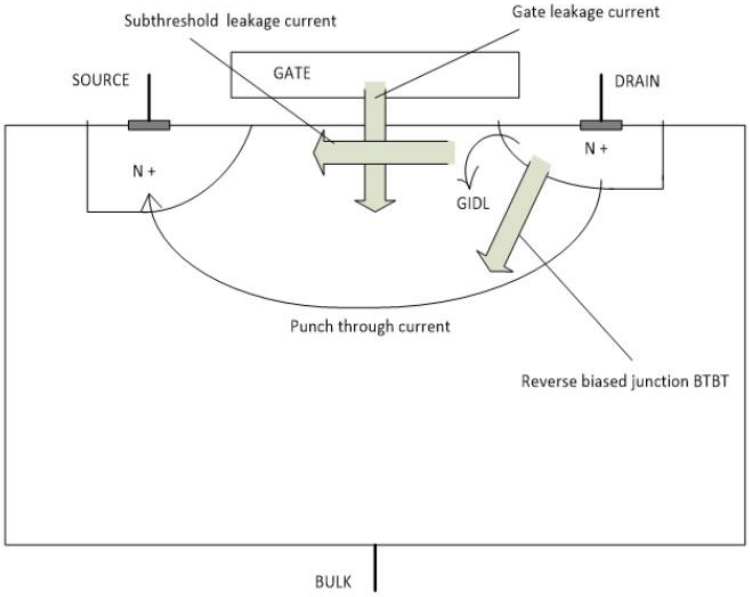


3. LEAKAGE POWER DISSIPATION

- **SUBTHRESHOLD LEAKAGE POWER DISSIPATION**
  When gate voltage is less than threshold voltage of MOSFET then even a current flow from drain to source through channel because of drain voltage (V_{DS}) this current is called as Subthreshold Leakage Current as shown in figure. It causes dissipation of power known as subthreshold leakage power dissipation.

- **GATE-OXIDE TUNNELING LEAKAGE POWER DISSIPATION**
  In short channel devices, resultant of electric field of gate voltage electric field and drain voltage electric field causes electrons to move in zig zag manner and some electrons which gain extra energy at pinch off tunnel to gate through insulating SiO_2 layer. This gives to gate leakage current towards substrate as shown in figure. It causes dissipation of power known as gate oxide tunneling leakage power dissipation.

- **REVERSE BIASED JUNCTION BTBT POWER DISSIPATION**
  It occurs when the source or drain of an NMOS is at V_{DD} or in case of PMOS is at ground. Reversed biased PN diode is formed at source or drain of transistors causing flow of current through substrate as shown in figure. In this process tunneling of electron of p type substrate from valence band to conduction band of n type drain occur. Hence, this current is known as Reverse Biased Junction BTBT Current. It causes dissipation of power known as reversed biased junction BTBT power dissipation.



11.2 LOW POWER DESIGN & TECHNIQUES

Low power design is a collection of techniques and methodologies aimed at reducing the overall dynamic and static power consumption of an integrated circuit (IC)

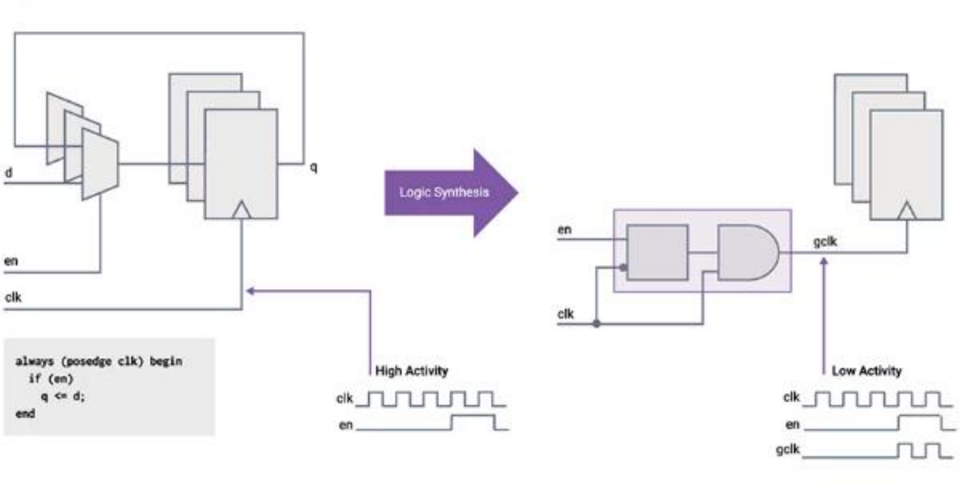
Looking at the individual components of power as illustrated by the equation in **Figure 1**, the goal of low power design is to reduce the individual components of power as much as possible, thereby reducing the overall power consumption. The power equation contains components for dynamic and static power. Dynamic power is comprised of switching and short-circuit power; whereas static power is comprised of leakage, or current that flows through the transistor when there is no activity. The value of each power component is related to any of the following factors:

- Activity
- Frequency
- Transition time
- Capacitive load
- Voltage
- Leakage current
- Peak current

For example, the higher the voltage, the higher the power consumed by each component, resulting in higher overall power. Conversely, the lower the voltage, the lower the overall power. To achieve the best performance with the lowest power consumption, tradeoffs for each of these different factors are tried and tested via various low power techniques and methodologies.

There are many low power design techniques available, some of which are very simple to use while others are more involved and complex.

- **Clock Gating**
  This technique is typically performed during logic synthesis where enable flops are optimized into a clock gating structure, thereby saving mux area and reducing the overall switching activity of the clock net (refer to **Figure 2**). With respect to the power equation, the goal is to reduce capacitive load (via area reduction) and activity factors which reduces the switching power component of dynamic power. This is a very simple and readily available technique to reduce power and area. However, it does rely on the logic synthesis tool to perform this optimization. Fortunately, this technique is well-known and well supported in most tools and flows.

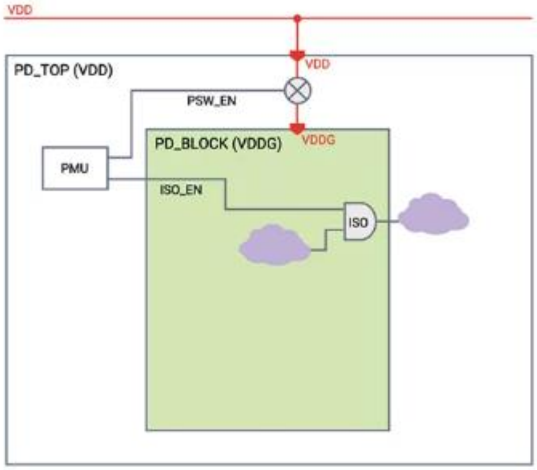


```
always (posedge clk) begin
  if (en)
    q <= d;
end
```

CLOCK GATING

- **Power Gating**
  This is a technique where functions on an IC are also partitioned, much like multi voltage, but this time the power supplies for the power domains are connected to power switches as shown in Figure 4. Power gating effectively shuts off the power completely for a block. In the power equation, this zeros out the voltage and shuts off power, resulting in both static and dynamic savings for the time that the block is turned off. Power gating typically offers the most aggressive power savings, and thus it’s an ideal goal to shut off as many domains as possible, as often as possible, while maintaining functionality. In order to achieve this power savings with power gating, power switches must be implemented in the design, which requires Isolation gates that clamp the boundaries of the power domain to known values when off.

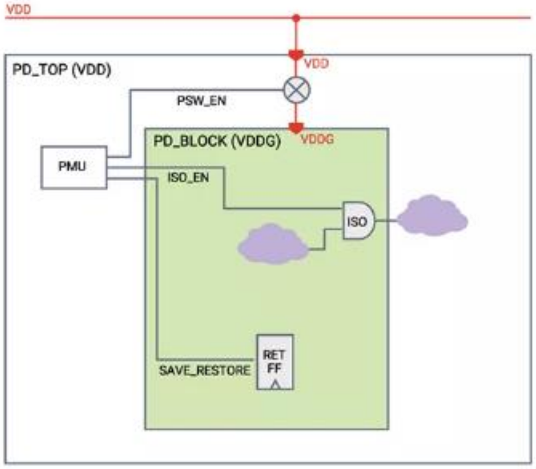
The power states of the design and what combination of ON/OFF states for given voltages must be considered. Lastly, a power management unit (PMU) that controls the power switch and isolation enable signals must be implemented. It is essential that the order of these signals are correct during power down and power up, such that the values during shutdown are clamped to the right values at the right time.



POWER GATING

- **Retention with Power Gating**

  Retention (or register retention) is a technique used along with power gating. Here in each shutdown block, when the block is OFF, either a subset of the flops or all the flops in the block have their previous values saved. When the block powers on, then the previously saved values will be restored. It is important to save the state of the block at the time it is powered off so that the block can quickly restore its previous state instead having to cycle through from an INIT state to the current state. This saves power by reducing the time and steps necessary to get the saved state, as well as improves the overall ramp up time to restore the previous functionality of the block. In addition to everything needed for power gating, retention flops must exist in the library that can map to the desired registers in the RTL. SAVE/RESTORE signals will need to be added to the PMU along with the control the sequence of these signals in addition to those done for power gating (refer to **Figure**).



Retention with Power Gating

12. COMPARATIVE STUDIES

- Ripple Carry Adder (RCA):

  ● Area: RCAs typically have a smaller area footprint than more complex adders, making them area-efficient for small bit-widths.

  ● Timing: RCAs generally have longer propagation delays due to the sequential carry propagation, making them slower for large bit-widths.

  ● Power: They consume less dynamic power but can have higher static power due to more active components.

- Carry Look-Ahead Adder (CLA):

  ● Area: CLAs tend to have a larger area footprint than RCAs due to the additional logic for generating carry lookahead signals.

  ● Timing: CLAs are faster than RCAs, especially for larger bit-widths, because they can calculate carries in parallel.

  ● Power: They consume more dynamic power due to increased logic, but their faster operation may reduce active time, leading to lower total energy consumption.

- Carry Skip Adder (CSKA):

  ● Area: CSKAs typically have a moderate area footprint. The skip logic adds some complexity, but it can reduce the overall area compared to CLAs for certain input patterns.

  ● Timing: They are faster than RCAs but may be slightly slower than CLAs in some cases. Timing improvements are achieved by skipping carry propagation in certain scenarios.

  ● Power: Power consumption depends on the number of skips, but CSKAs generally consume less power than RCAs for the same bit-width due to faster operation.

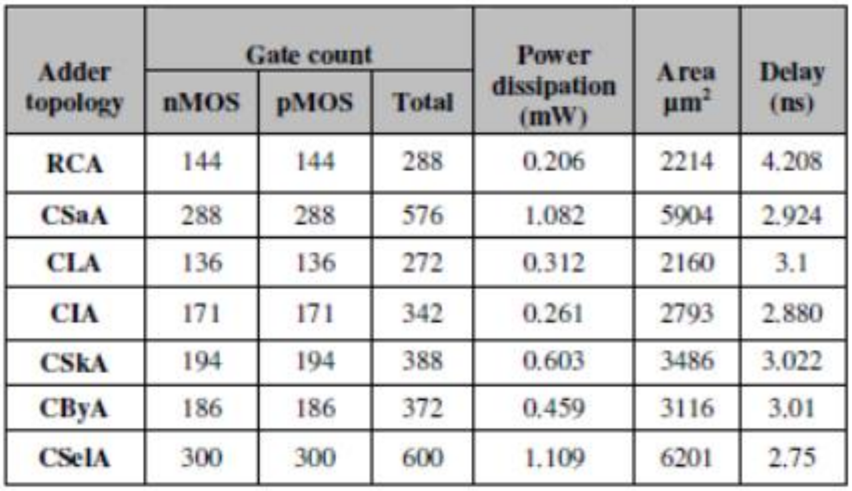
- Carry Select Adder (CSA):

  ● Area: CSAs have a larger area footprint than RCAs and CLAs due to the parallel adders and multiplexers.

  ● Timing: They are faster than RCAs and CLAs for large bit-widths due to parallelism but may be slower than CLAs for smaller bit-widths.

  ● Power: CSAs tend to consume more dynamic power due to the larger number of active components

Table 1 Area, Delay and Power Dissipation of Adders



Adder topology	Gate count			Power dissipation (mW)	Area µm^2	Delay (ns)
	nMOS	pMOS	Total			
RCA	144	144	288	0.206	2214	4.208
CSaA	288	288	576	1.082	5904	2.924
CLA	136	136	272	0.312	2160	3.1
CIA	171	171	342	0.261	2793	2.880
CSkA	194	194	388	0.603	3486	3.022
CByA	186	186	372	0.459	3116	3.01
CSelA	300	300	600	1.109	6201	2.75

15. FUTURE WORKS

- ## Advanced Technology Nodes:

Advanced technology nodes are crucial for the future of the semiconductor industry. The design of system-on-chips (SoCs) relies upon many pre-defined target parameters, including power consumption, voltage supply, clock frequency, data path timing, and desired physical area.

Advanced technology nodes, such as 7nm, 5nm, and beyond, enable the development of SoCs with higher performance, lower power consumption, and smaller physical footprint. These technology nodes allow for the integration of more transistors on a single chip, which in turn enables the creation of more complex and powerful SoCs.

The smaller feature sizes and improved transistor performance at advanced nodes also enable higher clock frequencies and faster data path timing, leading to better overall system performance. Additionally, advanced technology nodes often offer lower voltage supply requirements, which can contribute to reduced power consumption and improved energy efficiency.

Furthermore, the smaller physical area of advanced nodes allows for the creation of smaller, more compact SoCs, which is crucial for applications such as mobile devices and IoT devices where space is limited.

Overall, the continued advancement of technology nodes is essential for driving innovation and progress in the semiconductor industry, enabling the development of more powerful, energy-efficient, and compact SoCs that can meet the demands of modern and future electronic devices.

- ## Low-Power Design:

One approach to low-power design for adders is to explore novel circuit designs that prioritize energy efficiency. This could involve using alternative transistor structures, such as FinFETs or nanowires, which offer improved performance at lower power levels. Additionally, exploring different logic styles, such as adiabatic or reversible logic, can also contribute to reducing power consumption in adder circuits.

Power-gating strategies can also be employed to minimize power consumption in adder designs. By selectively shutting off power to unused portions of the circuit when they are not in use, power-gating techniques can significantly reduce overall energy consumption. Advanced low-power synthesis and optimization methods, such as clock gating, voltage scaling, and dynamic voltage and frequency scaling (DVFS), can also be applied to further minimize power usage in adder circuits.

In addition to these hardware-level approaches, software-based techniques such as algorithmic optimizations and data encoding schemes can also contribute to reducing power consumption in adder circuits. By optimizing the algorithms used in adder operations and encoding data in a more power-efficient manner, overall energy consumption can be further minimized.

Overall, exploring a combination of novel circuit designs, power-gating strategies, and advanced low-power synthesis and optimization methods can lead to significant reductions in power consumption in adder circuits, contributing to the development of more energy-efficient SoCs.

➢ **High-Performance Computing:**

In high-performance computing applications, adder designs can be optimized to maximize throughput and overall performance. One approach is to leverage parallelism by using multiple adders in parallel to perform addition operations on multiple sets of data simultaneously. This can significantly increase the overall throughput of the system.

Pipeline architectures can also be employed to optimize adder designs for high-performance computing. By breaking down the addition operation into smaller stages and processing them in a pipeline fashion, the overall latency of the adder can be reduced, allowing for faster computation of addition operations.

Custom adder designs tailored specifically for the requirements of high-performance computing applications can also be developed. These custom designs can incorporate specialized features such as reduced carry propagation delay, optimized for specific data types (e.g., floating-point or integer addition), and tailored for specific arithmetic operations commonly used in high-performance computing workloads.

Additionally, exploring novel circuit designs and advanced optimization methods, such as those mentioned in the low-power design approach, can also be beneficial for high-performance computing applications. By focusing on energy-efficient designs and optimizing power consumption, adder circuits can operate at maximum performance while minimizing energy consumption.

Overall, by leveraging parallelism, pipeline architectures, custom adder designs, and advanced optimization methods, adder designs can be optimized for high-performance computing applications to achieve maximum throughput and overall performance.

➢ **Machine Learning Integration:**

Machine learning and artificial intelligence techniques can be integrated into adder design to create adaptive or self-optimizing adders that can dynamically adjust their operation based on workload and data characteristics. This integration can involve using machine learning algorithms to analyze the patterns and characteristics of the input data and workload, and then using this information to optimize the operation of the adder in real-time.

For example, machine learning algorithms can be used to predict the types of data and arithmetic operations that are most commonly encountered in a specific high-performance computing workload. Based on these predictions, the adder can dynamically adjust its internal architecture, such as the number of parallel adders used or the pipeline stages employed, to maximize performance for the specific workload.

Furthermore, machine learning techniques can be used to continuously monitor the performance of the adder and identify areas for improvement. This can involve analyzing the latency, throughput, and power consumption of the adder under different operating conditions and using this information to optimize the adder's operation in real-time.

By integrating machine learning and artificial intelligence techniques into adder design, adaptive and self-optimizing adders can be created that continuously learn and adapt to the specific requirements of high-performance computing workloads, ultimately maximizing throughput and overall performance.

➢ Custom Adder Design:

Designing custom adders tailored to specific applications or processing tasks can offer several feasibility and benefits. By optimizing the adder for particular functions, efficiency and performance can be improved in various ways.

Firstly, custom adders can be designed to minimize power consumption and area usage by eliminating unnecessary features that are not required for the specific application. This can result in more efficient use of resources and lower overall system power consumption.

Secondly, custom adders can be optimized for specific arithmetic operations commonly used in a particular application. For example, if a specific application requires a large number of additions but very few multiplications, the custom adder can be designed to prioritize addition operations and minimize the hardware required for multiplication, leading to improved performance for the targeted tasks.

Additionally, custom adders can be optimized for specific data types commonly used in a particular application. For example, if an application primarily operates on fixed-point numbers with a specific range and precision, the custom adder can be designed to efficiently handle these data types, resulting in improved performance and accuracy for the targeted tasks.

Furthermore, custom adders can be designed with specialized features or optimizations to accelerate specific algorithms or processing tasks commonly found in a particular application. For example, if a specific application frequently performs matrix operations, the custom adder can be designed with specialized circuitry to accelerate matrix addition and multiplication, leading to improved performance for the targeted tasks.

Overall, designing custom adders tailored to specific applications or processing tasks can lead to improved efficiency, performance, and accuracy by optimizing the adder for the specific requirements of the targeted tasks. While this approach may require additional design effort and verification, the potential benefits in terms of improved performance and resource utilization make it a feasible and potentially advantageous strategy for certain high-performance computing applications.

➢ Design for Emerging Technologies:

As new technologies and materials continue to evolve, there is a growing opportunity to explore how adder designs can be adapted to harness their unique properties and improve performance. For example, memristors, which are non-volatile resistors with memory capabilities, offer the potential for ultra-low power and high-density computing. By designing custom adders that leverage the unique properties of memristors, such as their ability to retain state and perform logic functions, it may be possible to create adders with improved energy efficiency and reduced area usage.

Similarly, emerging technologies like spintronics, which utilize the spin of electrons to store and manipulate data, present new opportunities for adder design. By exploring how spin-based devices can be integrated into custom adders, it may be possible to achieve faster operation speeds and lower power consumption compared to traditional CMOS-based adders.

Furthermore, as quantum computing continues to advance, there is potential for custom adders to be designed specifically for quantum arithmetic operations. By tailoring adder designs to harness the unique properties of quantum bits (qubits) and quantum gates, it may be possible to create adders that are optimized for quantum algorithms, leading to improved performance in quantum computing tasks.

In conclusion, as new technologies continue to emerge, there is great potential for custom adders to be designed and optimized to take advantage of their unique properties. By exploring how adder designs can be adapted for emerging technologies such as memristors, spintronics, and quantum computing, it may be possible to achieve significant improvements in efficiency, performance, and functionality for future computing applications.

➢ Security and Cryptographic Adders:

In addition to exploring the potential of adder designs for emerging technologies, there is also a need to consider their application in cryptographic systems. Adders play a crucial role in cryptographic algorithms, such as encryption and decryption processes, and are therefore vulnerable to security threats and side-channel attacks.

To address these concerns, it is important to investigate how adder designs can be optimized for security and resilience against cryptographic attacks. This may involve developing custom adders with built-in security features, such as error detection and correction mechanisms, to prevent tampering or unauthorized access to sensitive data.

Furthermore, exploring the use of advanced cryptographic techniques, such as homomorphic encryption, in adder designs can help enhance their security and protect against side-channel attacks. By integrating these techniques into custom adders, it may be possible to ensure that sensitive information remains secure and protected from potential threats.

Overall, the development of custom adders with a focus on security and cryptographic resilience is essential for ensuring the integrity and confidentiality of data in modern computing systems. By investigating and implementing robust security measures in adder designs, it is possible to mitigate the risks associated with cryptographic attacks and enhance the overall security of computing applications.

- **Formal Verification of Adder Designs:**

Formal verification techniques involve mathematically proving that a design meets its specifications and requirements. This can help ensure the correctness of adder designs and reduce the need for extensive simulation-based verification, which can be time-consuming and resource-intensive.

By researching formal verification techniques for adder designs, it is possible to establish a rigorous methodology for verifying the functionality and security of adders. This may involve using formal methods such as model checking, theorem proving, and equivalence checking to verify that the adder design behaves as intended and is resilient against security threats.

Furthermore, formal verification can also help identify and eliminate potential vulnerabilities in adder designs that could be exploited in cryptographic attacks. By proving the correctness of the design, it is possible to ensure that the adder operates securely and reliably in cryptographic systems.

Overall, researching formal verification techniques for adder designs is crucial for enhancing their security and resilience in cryptographic applications. By establishing a formal methodology for verifying adder designs, it is possible to reduce the risk of security vulnerabilities and ensure the integrity of cryptographic systems.

16. 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