Analysis of ASIC Design Flow by performing Design and Verification of ALU Block and Lint using Spyglass

A
Dissertation
Submitted in the fulfilment of the requirements
For the award of degree

Of

MASTER OF TECHNOLOGY
In
VLSI Design and Embedded System

By
SHUBHAM GOYAL
(2K19/VLS/18)

Under the Guidance of

Dr. SONAL SINGH

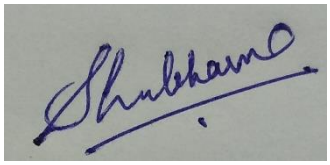
ELECTRONICS AND COMMUNICATION DEPARTMENT
DELHI TECHNOLOGICAL UNIVERSITY
DELHI-110042
SESSION 2019-2021

ELECTRONICS AND COMMUNICATION DEPARTMENT
DELHI TECHNOLOGICAL UNIVERSITY
DELHI-110042
SESSION 2019-2021

CANDIDATE'S DECLARATION

I hereby declare that the work being presented in this dissertation entitled **"*Analysis of ASIC Design Flow by performing Design and Verification of ALU Block and Lint using Spyglass*"** submitted towards the fulfilment of the Major project requirements for the award of degree, Master of Technology in VLSI Design and Embedded System to the Electronics and Communication Dept., Delhi Technological University, is an authentic record of my work carried out from January 2021 to June 2021, under the guidance of Dr. Sonal Singh, Electronics and Communication Dept., Delhi Technological University, Delhi.

I have not submitted the matter embodied in the dissertation for the award of any other degree.



Shubham Goyal
2K19/VLS/18
Electronics and Communication Department

Date: 24th June, 2021

ELECTRONICS AND COMMUNICATION DEPARTMENT
DELHI TECHNOLOGICAL UNIVERSITY
DELHI-110042
SESSION 2019-2021

CERTIFICATE

This is to certify that the dissertation entitled ***"Analysis of ASIC Design Flow by performing Design and Verification of ALU Block and Lint using Spyglass"*** is the authentic record of work done by **Shubham Goyal** under my guidance and supervision. This dissertation is being submitted to the ***Delhi Technological University, Delhi*** towards the fulfilment of the requirements for the award of ***degree of Master of Technology in VLSI Design and Embedded System.***

Date: 24th June, 2021

Dr. Sonal Singh
SUPERVISOR
Assistant Professor
Electronics and Communication Department
Delhi Technological University, Delhi

ACKNOWLEDGEMENT

I would like to express my deep gratitude and appreciation to all the people who have helped and supported me in the process of dissertation. Without their help and support, 1 would not have been able to reach this level of satisfaction with what 1 have learnt and accomplished during my Master's dissertation. First and foremost, I would like to express my deep sense of respect and gratitude towards my supervisor Dr. Sonal Singh, Assistant Professor, Electronics and Communication Dept., DTU, for giving me opportunity to do my Major project of master's dissertation under her guidance. I am very thankful for her for giving me the opportunity to choose such an interesting topic by my own. I would also like to thanks the NPTEL Lectures for their valuable thoughts and knowledge, which motivated me to do better. Finally, none of this would have been possible without incredible support of my friends. They were always supporting me and encouraging me with their best wishes.

Shubham Goyal
Roll No. 2K19/VLS/18
Electronics and Communication Dept.

ABSTRACT

We are living in the era of artificial intelligence where everything which we can imagine is in our hands with the help and emergence of the ongoing technology. There is a need for the semiconductor industries, as well, to make themselves comfortable with the growing pace of the world of technology. With the increase demand of the technologies and the evolution of the products, the big giants of semiconductor industries like Qualcomm, Intel, NXP Semiconductors and Western Digital are finding their way in a best appropriate manner to design the product which is user friendly.

There are various constraints which are implemented, which are imposed by these industries like functionality of the electronic device, power dissipation by the product, area occupied and also the reliability of the product. All these constraints, require some special attention and the measurements, which needs to be fulfilled by the design engineers, so that the reputation of the industry, and the competition in the products will be sustained.

There are various steps, followed by every Semiconductor industry to make their products the best one. Some of the basic VLSI design flow steps are described in this thesis, and a special focus has been done on the frontend part of the design flow, which includes the designing of the RTL code, and then the Lint process, which generally verifies the syntax and the functionality of the coding so that it can be synthesized properly. And then, verification environment has been created with the help of system Verilog, so that the verification of the RTL code could be done, along with a brief introduction of UPF unified power format is also been studied in this thesis, so that the power aware estimations can be done, along with the functionality checks of the RTL design.

The last set of these thesis is a study of the clock domain crossing as clock is one of the crucial nets in the design of complex SOCs, and there are many clock domains running from one part of the SOC to another. So, there must be the proper data transfer between the two clock domains.

To understand the basic concept of design and verification of the SOC, the thesis contains one example of ALU, the design part of which is written in the Verilog language, it contains all the proper syntax of the language and the code

is synthesizable. And to verify the functionality of this, a new environment is created with the help of system Verilog which includes the concepts of object-oriented programming and functional coverage measures. After the verification part of the design process, lint is done to verify the functional checks on the RTL design so that it can be synthesized properly to the gate level netlist.

TABLE OF CONTENT

CHAPTER	TITLE	PAGE NO.
	DECLARATION	i
	CERTIFICATE	ii
	ACKNOWLEDGEMENT	iii
	ABSTRACT	iv
	LIST OF FIGURES	vii
	LIST OF ABBREVIATIONS	xi
1	INTRODUCTION	1
2	LITERATURE REVIEW	15
3	LINT, CDC AND UPF	44
4	RESULT AND DISCUSSION	55
5	CONCLUSION AND FUTURE SCOPE	67
	REFERENCES	68
	APPENDIX	70

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
1.1	Memory Specification Chart	3
1.2	NAND Controller Architecture	4
1.3	RTL of function and showing data flow	5
1.4	Basic Architecture of the Testbench Environment	5
1.5	Conversion of RTL to Gate Level Netlist	6
1.6	DFT Insertion	7
1.7	Functional Verification of the logic	7
1.8	Data path for Timing Analysis	8
1.9	Conversion of Netlist to Layout is PD	8
1.10	Partitioning and Floor-Planning	9
1.11	Placement of the Blocks	10
1.12	Clock Nets connected to F/Fs	11
1.13	Routing	11
1.14	Design Rule Check and Layout Versus Schematic	12
1.15	CMOS Fabrication Process	13

2.1	Chip after Floor-planning	16
2.2	Chip after Placement	16
2.3	Chip after Clock Tree Synthesis	17
2.4	Chip after Clock Net Shielding	18
2.5	Chip after Routing	19
2.6	Chip after Parasitic Extraction	19
2.7	Width and Height of the core	20
2.8	Effective area calculation	21
2.9	Finding the utilization factor	21
2.10	interconnection of IPs	22
2.11	Black-Boxing the logic	22
2.12	Commonly used Macros	23
2.13	Cells without Decoupling Capacitor	23
2.14	Cells with Decoupling Capacitor	24
2.15	Placement of Decoupling capacitances	24
2.16	Power Planning	25
2.17	Mesh arrangement of Power Rails	26
2.18	Pin Placement	27
2.19	Logical Cell Placement Blockage	27
2.20	Mapping of Library cells	28
2.21	Placement of Logical Cells	29

2.22	Optimised placement with inserting Buffers	31
2.23	Setup time analysis for ideal clocks	32
2.24	Hold time analysis for ideal clocks	32
2.25	Clock available at CLK1	33
2.26	Clock Tree Synthesis	34
2.27	Clock path using H-tree	35
2.28	Clock net shielding	36
2.29	Setup time analysis for Real clocks	36
2.30	Hold time analysis for real clocks	37
2.31	Maze Algorithm for Routing	38
2.32	Routing	39
2.33	Routing the data path	39
2.34	Lambda rules	40
2.35	Steps of Physical Design Flow	43
3.1	Showing Synchronous Clocks	46
3.2	Asynchronous clock	47
3.3	Concept of Metastability	48
3.4	Advantages of 2 F/F Synchronizers	49
3.5	Data Copying Problem due to Metastability	50
3.6	3 F/F Synchronizer	50
3.7	Explaining Effect of Metastability	51

3.8	Metastability on Pulse delay	51
3.9	Metastability on Pulse Missed	52
3.10	Metastability on Glitch Captured	52
4.1	Spyglass error W18	55
4.2	Spyglass error W19	55
4.3	Spyglass error W69	56
4.4	Spyglass error W111	57
4.5	Spyglass error W123	57
4.6	Spyglass error W336	58
4.7	Spyglass error W391	59
4.8	Spyglass error W392	59
4.9	Spyglass error W414	60
4.10	Spyglass error W448	60
4.11	Error information in Spyglass	61
4.12	Report generation in Spyglass	61
4.13	Error reduction in Spyglass	62
4.14	Functional Coverage in QuastaSim with 25 test cases	62
4.15	Functional Coverage in QuastaSim with 25 test cases for particular inputs.	63
4.16	Waveform showing of the functionality of ALU with 25 testcases.	64
4.17	Functional coverage of ALU with 50 testcases.	64

4.18	Functional coverage of ALU with 150 testcases.	65
4.19	Functional coverage of ALU with 250 testcases.	65
4.20	Functional Coverage in QuastaSim with 250 test cases for particular inputs.	66

LIST OF ABBREVIATIONS

RTL	Register transfer level
VLSI	Vert Large Scale Integration
GDSII	Graphic design system
ASIC	Application specific integrated circuit
GPU	Graphical processing unit
SOC	System on chip
FPGA	Field programmable gate array
HDL	Hardware description language
DFT	Design for testability
STA	Static timing analysis
CTS	Clock tree synthesis
UF	Utilization factor
PnR	Placement and Routing
DRC	Design rule checks
IP	Intellectual property

IC	Integrated circuit
EDA	Electronic design automation
CLK	Clock
DUT	Design Under Test
DUV	Design Under Verification
CDC	Clock Domain Crossing
UPF	Unified Power Format
ALU	Arithmetic and Logical Unit
MTBF	Mean Time Before Failure
F/Fs	Flip Flop
I/O	Input or Output
SDs	Standard Devices
PD	Physical Design

Chapter 1

INTRODUCTION

1.1 VLSI DESIGN FLOW

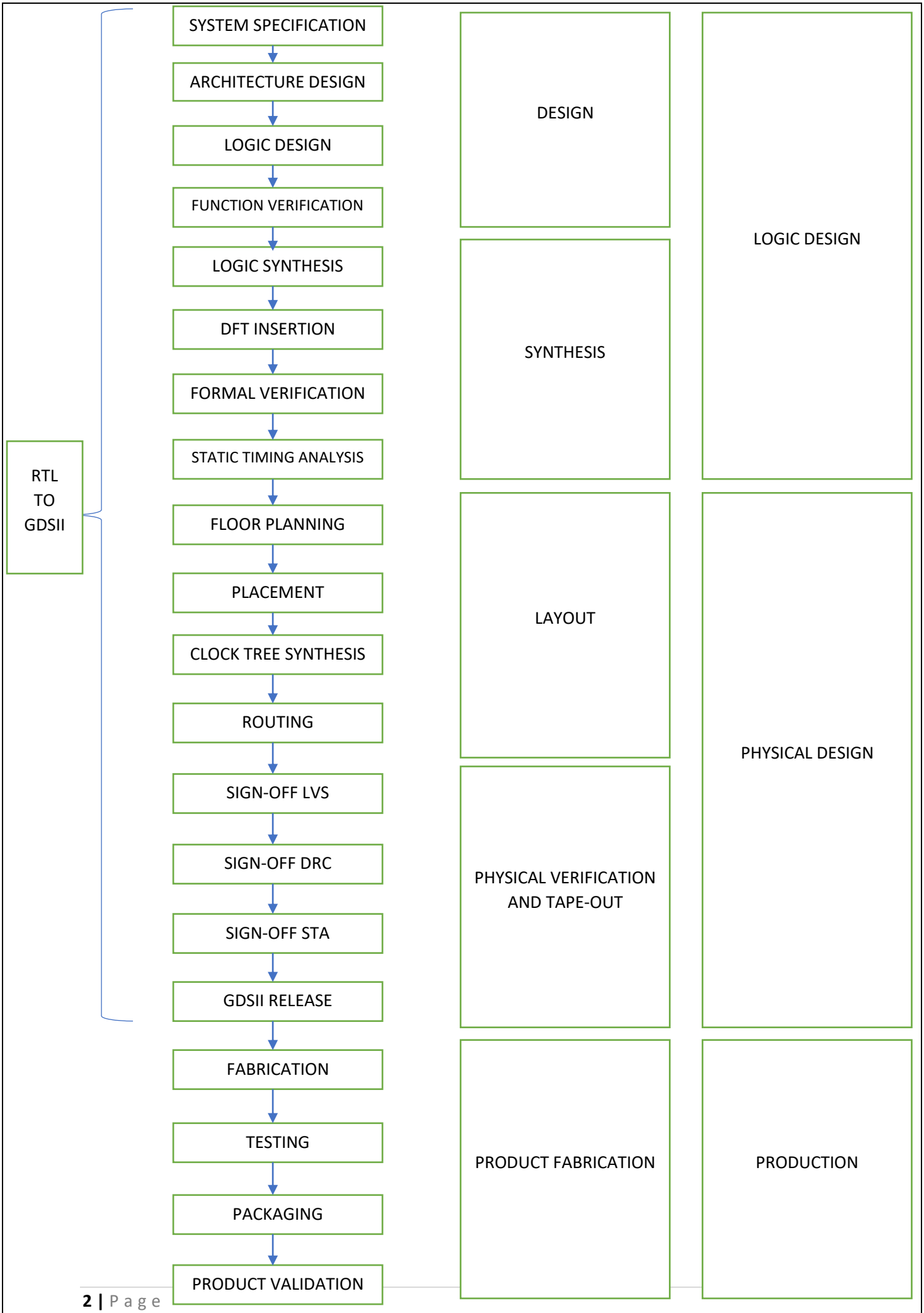
With the increase of the semiconductor industry and with the emergence of the VLSI, there is a lot of contribution of the big giants like Intel, Qualcomm, Western digital to produce millions of chips within a small amount of time. And the experience we are getting to the evolution of these Products and the IPs impact the world of artificial intelligence up to great extent.

The ongoing process of making the chips by the semiconductor industries requires the particular flow for the correct functioning of the chip and the good yield which further leads to the marketing strategy of the big giants like Western Digital, Qualcomm, Intel and many more. The VLSI Design Flow is depicted below which is followed by almost all the industries in the world to make their products a better one.

Here in this session, the VLSI design flow is explained thoroughly so that a brief idea could get, how the industry follows the basic steps from the design specification, up to the final silicon, which we are using in our PCs, in our mobile phones and in our day to day life.

There are many steps, which mainly include the front end of the VLSI design flow and the backend design flow and also the packaging part. Each industry has its different names given to be intermediate steps but generally, the major ones are as follows. There is a difference between the ASIC design flow and FPGA design flow.

The front end of the basic design for an FPGA design flow is common for the ASIC as well. It has input as the design specification and goes to the gate level netlist, which is nothing but the synthesizable RTL code. The difference between the ASIC and FPGA design flow comes in the backend part, as in FPGA we already have the hardware, so we don't need the floor planning and placement steps of the physical design. These steps have been eliminated in the FPGA part since FPGA design flow basically eliminates, and we don't get the entire flow of the backend part. So here, the VLSI design flow is basically depicted in terms of the ASIC design flow.


SYSTEM SPECIFICATION
ARCHITECTURE DESIGN
LOGIC DESIGN
FUNCTION VERIFICATION
LOGIC SYNTHESIS
DFT INSERTION
FORMAL VERIFICATION
STATIC TIMING ANALYSIS
FLOOR PLANNING
PLACEMENT
CLOCK TREE SYNTHESIS
ROUTING
SIGN-OFF LVS
SIGN-OFF DRC
SIGN-OFF STA
GDSII RELEASE
FABRICATION
TESTING
PACKAGING
PRODUCT VALIDATION
RTL TO GDSII
DESIGN
SYNTHESIS
LAYOUT
PHYSICAL VERIFICATION AND TAPE-OUT
PRODUCT FABRICATION
LOGIC DESIGN
PHYSICAL DESIGN
PRODUCTION

1.1.1 LOGIC DESIGN

Starting with the first part of the ASIC design flow is the logical design. This is also known as the front end of VLSI, where we have the specifications of the product which we needed for the designing part. Then we had the architectural design, logical design, functional verification, logical synthesis, DFT, formal verification and static timing analysis and this is basically what the logical design do. The input of the logical design part of the ASIC Design Flow is the specifications provided by the marketing team, or by the directors and the managers of the industry, which are having their look to the ongoing production which they have launched in the market, till the gate level netlist which can be synthesizable and which should be given as an input to the physical design engineers of the ASIC design flow.

1.1.2 SYSTEM SPECIFICATION

Starting with the VLSI Design Flow, the first step is a system specification. System Specification basically includes all the things which industry have as the ongoing customer demand, which the marketing team have analyzed in the market phenomena. It contains all the specifications of the blocks which the organization wants to design in terms of functionality, speed, power and area, or whether in terms of the research done by the marketing team. The one such kind of specification is shown in Figure 1.1.

Host type / Card type	SD - up to UHS50	SD UHS104	SD-UHS-II	SD-UHS-III	SD Express
SD - up to UHS50	Up to 50MB/s	Up to 50MB/s	Up to 50MB/s (basic SD interface)	Up to 50MB/s (basic SD interface)	Up to 50MB/s (basic SD interface)
SD - UHS104	Up to 50MB/s	Up to 104MB/s	Up to 104MB/s (basic SD interface and if host supports it)	Up to 104MB/s (basic SD interface and if host supports it)	Up to 104MB/s (basic SD interface and if host supports it)
SD-UHS-II	Up to 50MB/s	Up to 104MB/s (if supported by card)	Up to 156MB/s (Full Duplex) Up to 312MB/s (Half Duplex)	Up to 156MB/s (Full Duplex) Up to 312MB/s (Half Duplex if supported by host)	Up to 104MB/s (basic SD interface and if host supports it)
SD-UHS-III	Up to 50MB/s	Up to 104MB/s (if supported by card)	Up to 156MB/s (Full Duplex) Up to 312MB/s (Half Duplex if supported by card)	Up to 624MB/s (Full Duplex)	Up to 104MB/s (basic SD interface and if host supports it)
SD Express	Up to 50MB/s	Up to 104MB/s (if supported by card)	Up to 104MB/s (basic SD interface and if host and card support it)	Up to 104MB/s (basic SD interface and if host and card support it)	Up to 985MB/s (PCIe Interface)

Fig 1.1: Memory Specification Chart

1.1.3 ARCHITECTURE DESIGN

Next step after the design and Market Specification is Architecture Design. Now in architecture design the organization generally have the broad idea of the architecture or the higher level of abstraction where engineers know all the components and all the SOCs needed to be integrated for the designing part. One SOC contains millions of IPs and the interconnections between them so it's the part of the high level of engineers to implement such an architecture like for example, some SoC then it's a part of the architectural design, which designers have to implement this, either by using the ripple carry adder or by the carry Look ahead adder or skip carry adder or many other topologies. This is the type of architecture design the industry generally have. Figure 1.2 shows one such type of Architecture of the NAND Memory.

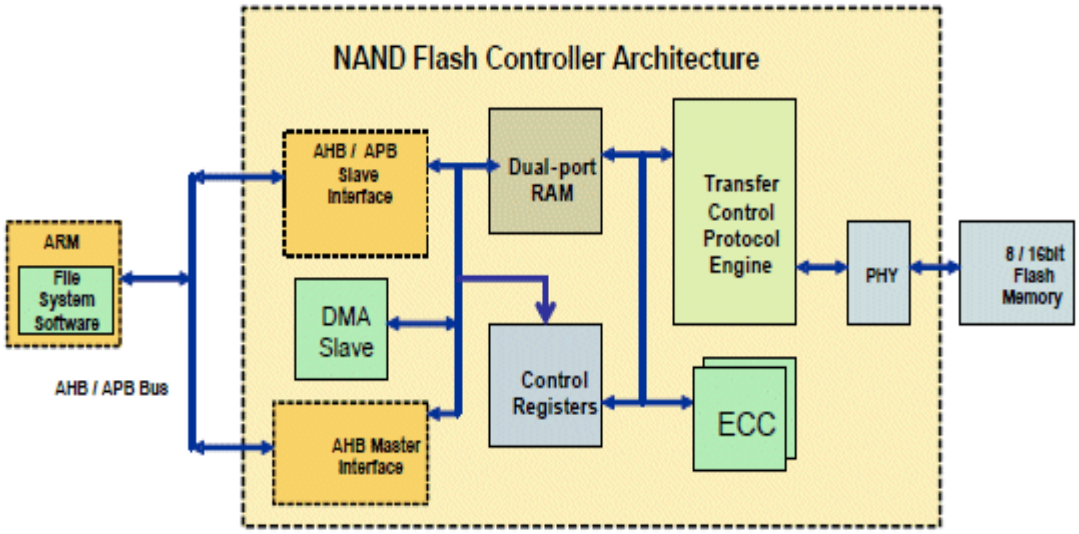


Fig 1.2: NAND Controller Architecture.

1.1.4 LOGIC DESIGN

The next step after the architectural design is the logical design, where designers are actually designing the logic of the Product based on the specification. For all the specification and for all the product, designer want to design for the requirement of the industry logic design, like for example if the requirement is to design the and gate then, let's suppose there is a four input NAND gate using two input AND gate. Then there are two ways to design it, the logic one is this and the logic, two is shown in the figure. The design can be implemented with the help of HDLs (Hardware Descriptive Language),as shown in Figure 1.3.

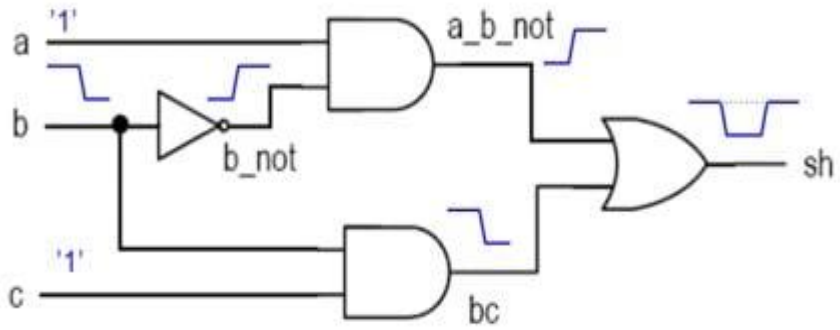


Fig 1.3: RTL of function and showing data flow.

1.1.5 FUNCTIONAL VERIFICATION

After that we'll go for the functional verification. Functional verification is the major step for defining the yield of the VSI design flow as to check the functionality of the logic design which we have implemented so that in the later part of this flow, the industry can get the correct yield, and the correct output so that we should not go to the entire flow process which is time consuming and expensive as well. This can be done with the help of high-level verification environment made with the help of system Verilog and UVM (Universal Verification Methodologies). The components of Environment are shown in the Figure 1.4.

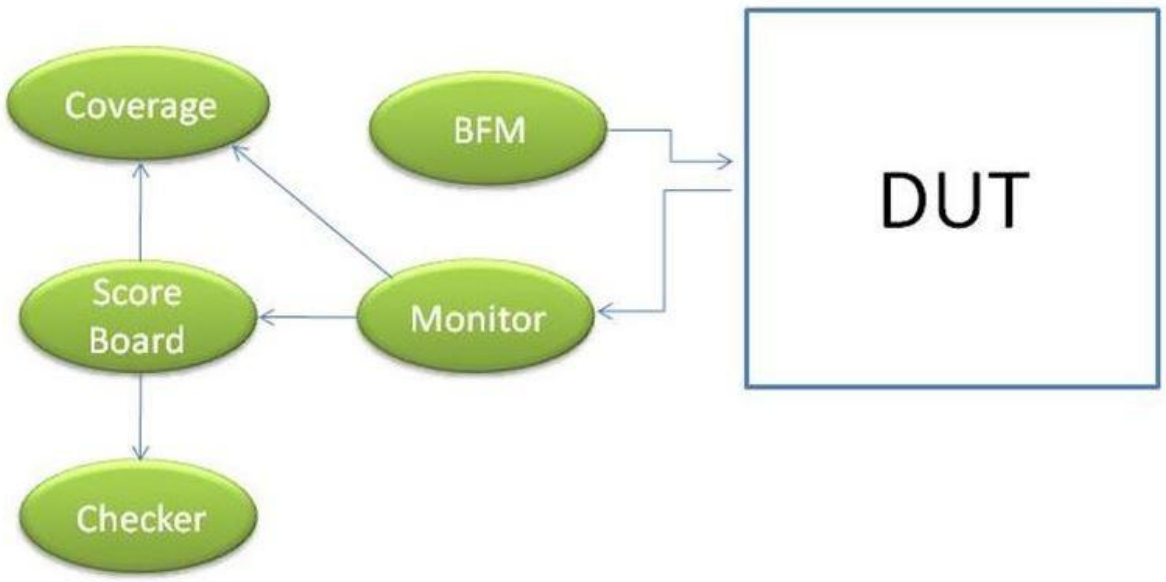


Fig 1.4: Showing the Basic Architecture of the Testbench Environment.

1.1.6 LOGIC SYNTHESIS

After the verification. After the functional verification of the above design which we have implemented using these specifications given by the marketing team or by the customers, we want to design it for the hardware purpose so the next step comes is the logical synthesis. Synthesis is basically the step where we need to convert the RTL into the gate level netlist, so that it can be synthesizable. And it can be implemented in the actual hardware. This step, eliminates all the unwanted and unnecessary coding styles which we have made by mistake in RTL Design part while writing the Verilog and System Verilog code, it contains the elimination of unwanted Latches, width mismatch, unintentional loops, and many other errors. So, this part is basically the actual synthesis part of this design process where we are getting the Actual Hardware, as shown in Figure 1.5.

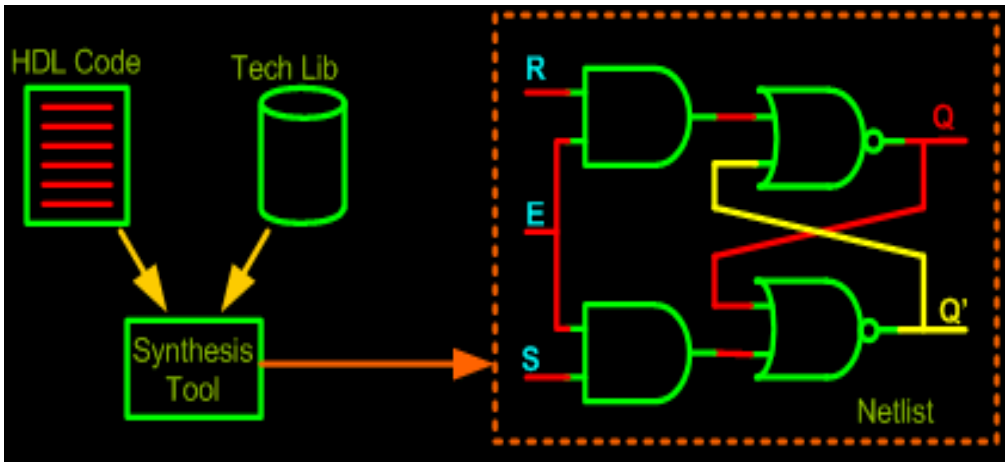


Fig 1.5: Conversion of RTL to Gate Level Netlist

1.1.7 DFT INSERTION

After the synthesis, where we get the gate level netlist of the design, we move to the DFT insertion In this step we are just inserting the DFT that is design for testability block into the design, so that the block can test itself in the simulation flow, and it will create test vectors for functional analysis, we have random generator block, which will create error patters and test patterns to test the circuitry of design which we have implemented so that it can be verified properly. The Scan and MBIST topologies are shown in Figure 1.6.

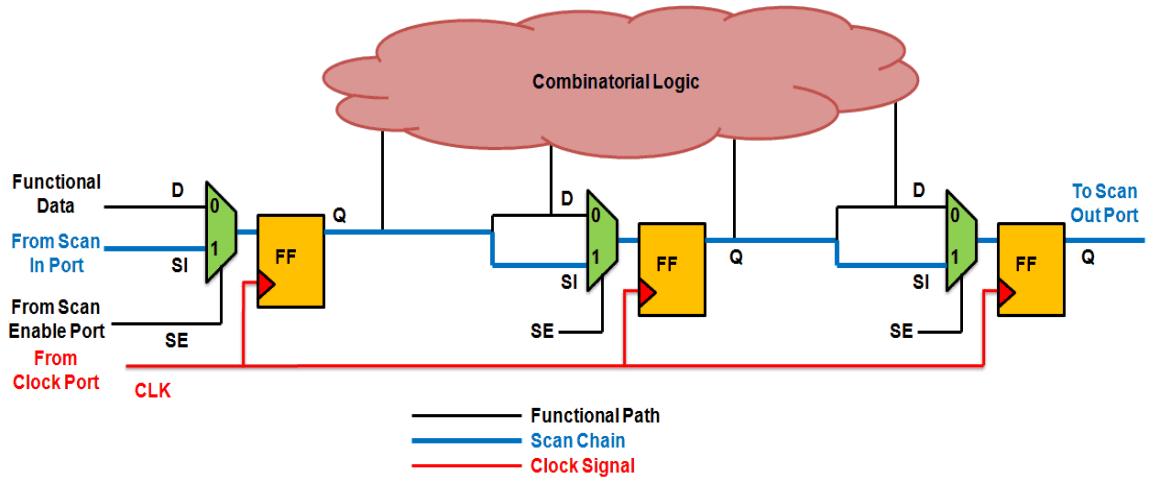


Fig 1.6: Showing the DFT Insertion.

1.1.8 FORMAL VERIFICATION

After the insertion DFT insertion, next is the formal verification, where the tool verifies the logic and the DFT part which it has implemented, and verify all the things in a broad aspect. The concept of emulation is used by the verification engineers where, the bits patterns are given to the DUT in terms of C Codes and the log file is maintained for all the mis-matches based on the golden response. The following image is shown below for the formal verification.

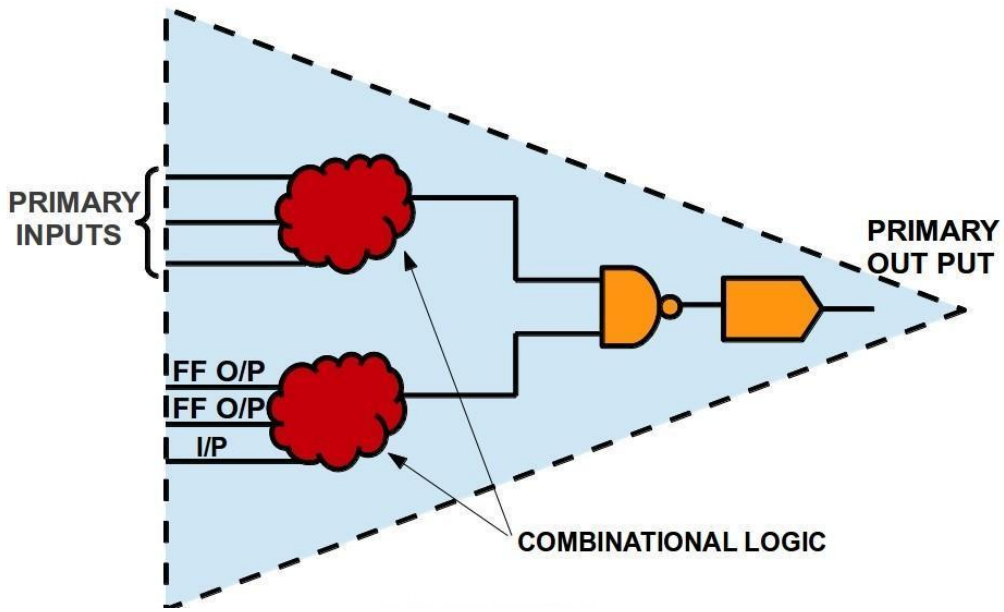


Fig 1.7: Showing the Functional Verification of the logic.

1.1.9 STATIC TIMING ANALYSIS

After the formal verification done by the verification engineers, next is to move to the static timing analysis. The analysis done till now is for the functionality of the design. Now it comes to the timing part of the design, where industries are using static timing analysis, static timing analysis is also the timing analysis, which contains many constraints and many parameters like setup constraints, hold constraints, clock frequency, skew rate, effect of jitter on the timing, effect of delta delay on the timing. There are many questions in a static timing analysis where STA engineer checks all the parameters, and then verify, and then rolls the ball to the layout team. So here ends the logical design of the ASIC design flow, which is generally refers to the FRONT END of ASIC design Flow.

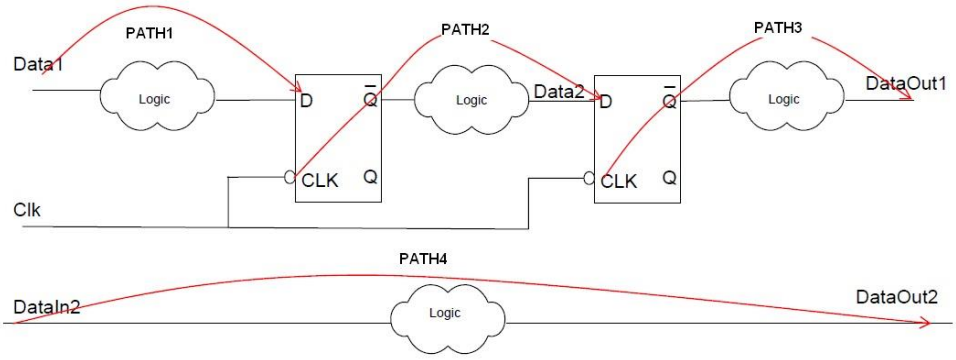


Fig 1.8: Showing Data path for Timing Analysis

1.1.10 PHYSICAL DESIGN FLOW

After completing the logical design of the ASIC design flow, we move to the physical design. Physical design is the other angle of the flow where we have the gate level netlist generated by the step of synthesis, and then we move to the layout part. There are various steps which include from floor planning to the Layout part, which comes under the physical design.

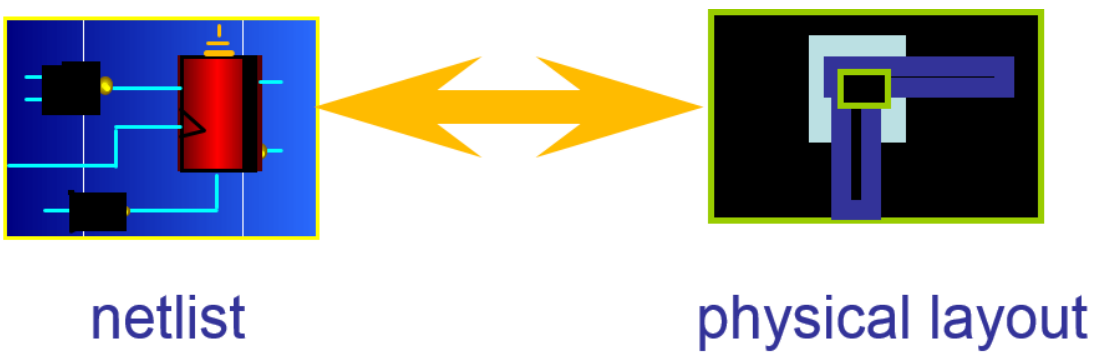


Fig 1.9: Conversion of Netlist to Layout is PD

1.1.11 PARTITIONING AND FLOOR PLANNING

This is the first step of the Physical Design where the big SoC is divided into small sub groups and generally this step is referred to as Partitioning which is based on some constraints like interfaces must be minimum, each partition much get almost equal amount to gates and complexity. After that, the next step is Floor-planning. Like for example, when we make the house we have the blueprint of the house we have the bathrooms, we have the location of the kitchen, all the blueprints which we have for the house similar Blueprint we have for the chip as well, where we have the location of the pre-placement cells, where we have the pin placement, power planning and other information.

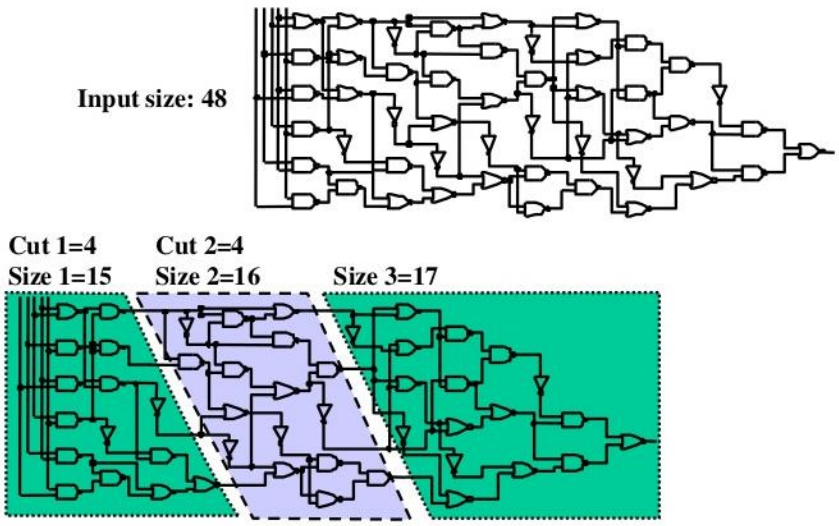


Fig 1.10 (a)

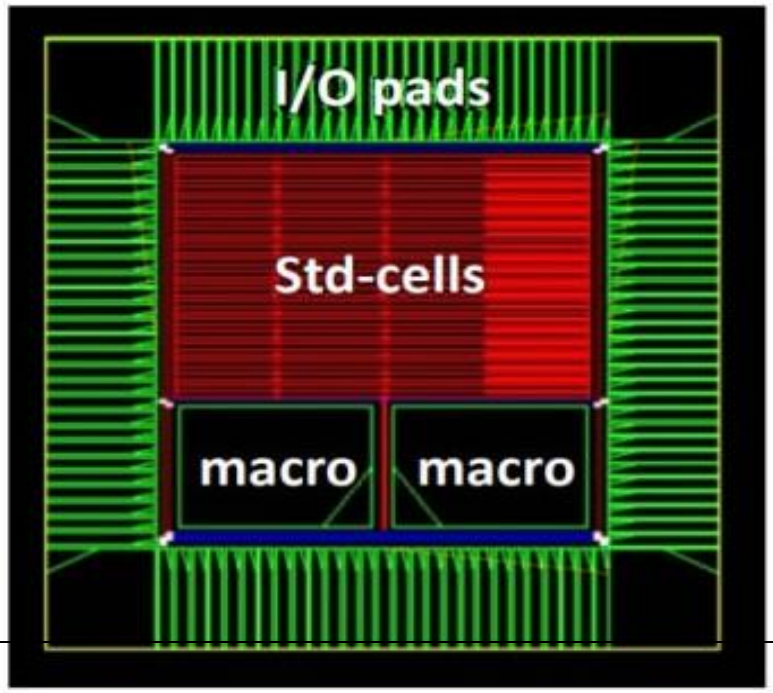


Fig 1.10 (b)

Fig 1.10 (a) Showing the Partitioning (b) Showing the Floor-Planning

1.1.12 PLACEMENT

After the Floor-planning, we have the placement of the logical blocks, how they are placed on the basis of the flow of the data. So, there are many constraints of the placements like whether we have to include buffers in the parts, whether we have to replace IOs, technology cells, S/Ds, IPs. So, in this step we roughly placed all these cells, and we just measure the locations inside the core.

As mentioned in the floor-planning step, we generally measured the width and the height of the core, and in the placement, we take the blocks of the cells from the libraries, and we just fit it inside the core so that it can be placed according.

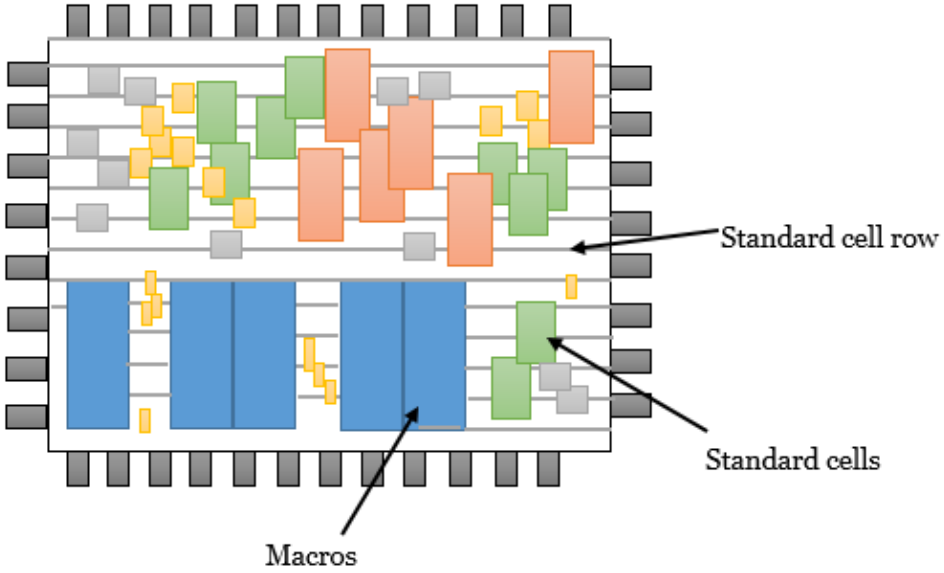


Fig 1.11 Placement of the Blocks

1.1.13 CLOCK TREE SYNTHESIS

After the placement is done, the most important step is to fix the clock nets, as we know that clock is a major source of concern in the field of VLSI, it is the most switching signal, and the product efficiency depends majorly on the clock frequency. It decides the overall frequency of this system. So, this step includes the synthesis of the clock tree, what are the measures which are taken for the implementing clock in this circuit, it contains many constraints like we have to check the pulse width of the clock, pulse duration, we have to check skew, we have to check slew rate, we have to check the latency. So, these things are covered in the clock the synthesis part.

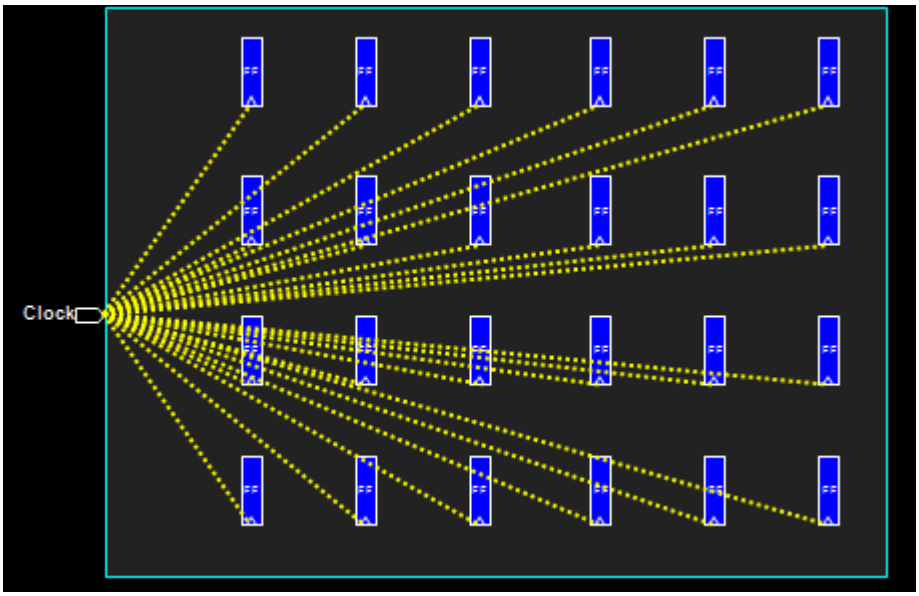


Fig 1.12 Showing the Clock Nets connected to F/Fs

1.1.14 ROUTING

The next part after synthesizing the clock, we generally synthesize the data paths, since we have just placed the cells we haven't connected, or we haven't routed it yet. So now in the routing part we can only connect the data path in an efficient manner. So here we generally use the algorithms of the shortest path, of which we generally covered in the data structure subject.

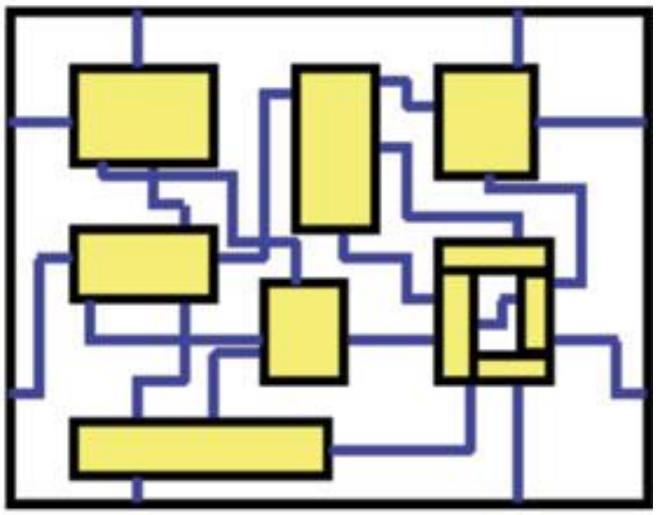


Fig 1.13 Showing the Net connection between the Blocks.

1.1.15 SIGN-OFF DRC, LVS, STA

After the routing part we have the sign-off things, Layout v/s Schematic, DRC and STA. These are the three tasks, we complete with the help of tool so that it measures all the timing requirements of the product, of the placement and clock tree synthesis and the routing which we have developed. If there is any error or if there any mismatch in the timing, which we found in these checks, then we'll go above the flow and we'll do the changes in the placement, in the clock tree and in the routing part, we generally avoid changes in the clock path as the clock net is the most critical net and we have to take care of it, so we generally change in the data path that is in Routing.

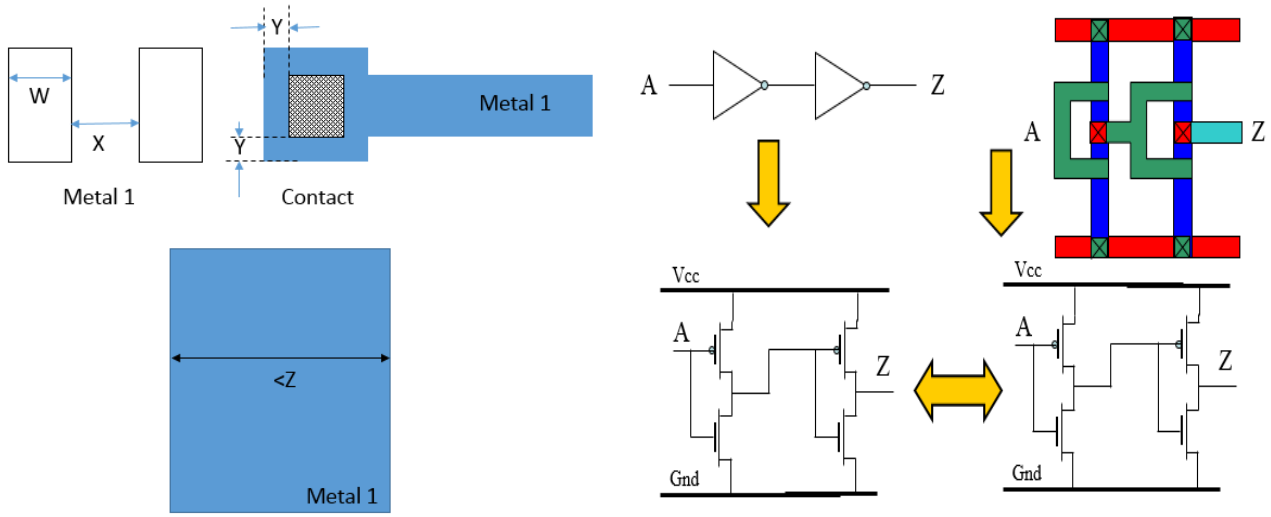


Fig 1.14: Showing the Design Rule Check and Layout Versus Schematic

1.1.16 PRODUCTION

After the functional checks which we have done, like Layout versus schematic, DRC checks and STA checks, we are ready for our GDSII log file which we have the information of all the layout kind of thing that is layout and systematic, which is the input to the foundry. So, many VLSI industries have their foundries, which take the GDSII file from the engineers from the software patch, and they do the necessary steps of lithography and etching so that they can fabricate the Silicon chips.

The next step is to fabricate the chip based on the GDSII and the machines present in foundry. Now, after the fabrication of the chips, there are many chips inside. Inside the silicon wafers which are not functionality correct, which should be checked so that so to maintain the functionality of the products which we have implemented, so the next step is Testing. This is the post Silicon testing. In the above verification part, we have used pre-silicon testing where we have tested we have verified through simulation. Now this is we are verifying with the help of hardware we have in the hands. We have many test vectors, we generally use C codes and the Python codes for testing. We verified these results with the golden results and verify the functional coverage. After the testing, the product which we are desired, is allowed to done with validation

part, where we do the last function checks of the product which we have created.

In this way the entire basic flow goes from the specification to the product validation, which includes several steps and there are many intermediate steps between these major steps, so the necessary parts are covered in the following sections.

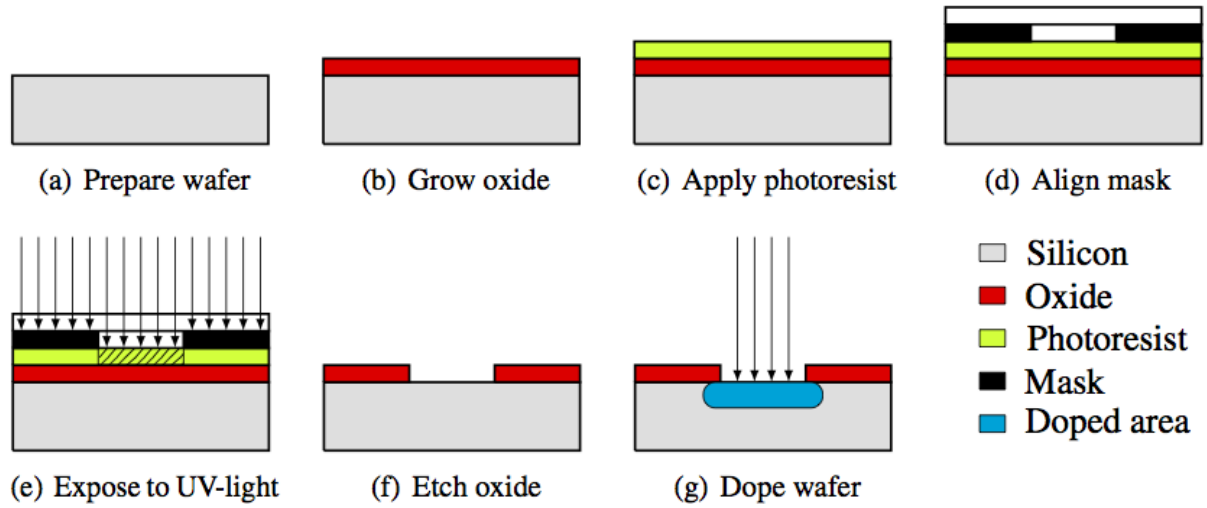


Fig 1.15: Showing the CMOS Fabrication Process

1.2 OBJECTIVES

The main objectives of this thesis are as follows:

- To understand the ASIC design Flow in great extent by analysing its various stages from Design Specification till the Production of the Product.
- To understand the various stages in the Physical Design Flow.
- The functional checks in the Verilog like: Lint, Clock Domain Crossing and UPF.
- To Design and Verify the ALU with the help of Quartus Prime and calculate its functional coverage.

1.3 THESIS ORGANIZATION

The Thesis is organized in 5 chapters which are as follows:

- **Chapter 1** introduces the overall aspect of the ASIC Design Flow, which starts with the specification till the Production of the product. And the objective of the thesis which is showing the motivation while taking this title.

- **Chapter 2** helps in understanding the Physical Design Flow in detail, which includes the Floor-Planning step and then the placement and Routing till the Sign-off STA.

- **Chapter 3** includes 3 major steps of Linting, Clock Domain Crossing and UPF. These are the functional checks which helps in the synthesis of the gate level netlist from the Verilog code.

- **Chapter 4** shows the results which includes the ALU Design and Verification of ALU depending on the code coverage. And the various cases in the Lint using Spyglass which helps in understanding the Functional Checks easily.

- **Chapter 5** results in the conclusion and the future scope of the thesis and also what can be done next and the improvement in the design flow which helps in the industry to improvise.

Chapter 2

LITERATURE REVIEW

2.1 INTRODUCTION

As discussed earlier in the first chapter that the VLSI design flow comprises of the physical design flow. Here is a quick overview of the physical design flow, where the input is the gate level netlist, and we have to design till the GDSII which contains the layout files of the gates for all the SOCs or the IPs which are needed to be implemented. There are many steps in the physical design flow, which contains many further steps depending on the topologies used by the particular industry.

The physical design steps comprise of:

1) Floor Planning
2) Placement and Optimize Placement
3) Clock Tree Synthesis
4) Clock Shielding
5) Routing
6) DRC check.
7) Parasitic Extraction

2.2 OVERVIEW OF PHYSICAL DESIGN FLOW

2.2.1 Floor Planning

Starting with the floor-planning, where we have the constraints on width and height of the core, and to integrated many pre-placed cells. Power planning and Pin placement is also done in this step of Physical Design Flow. Figure 2.1 is showing the snippet of Floor Planning where there are many components like, core and die, decoupling capacitance, power grids and pins connections.

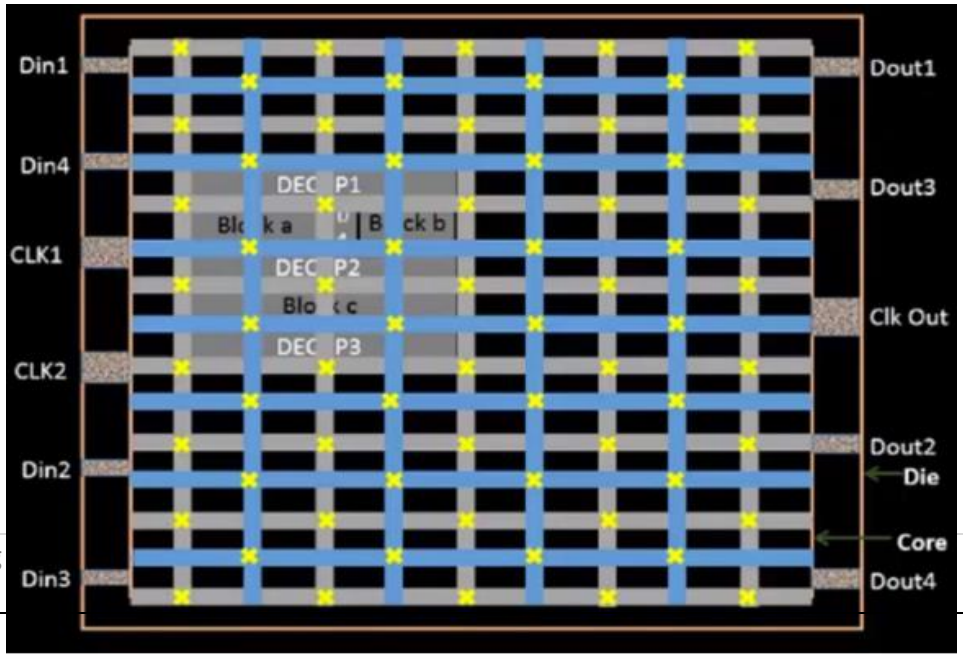


Figure 2.1: Snippet of the chip after Floor-planning

2.2.2 Placement and Optimised Placement

Then we have the netlist binding and placement optimization, where we are just finding the netlist and we have the placements, which get optimized, including the buffers and all the necessary things. Figure 2.2 is showing the chip where the blocks are placed according to the defined regions based on the placement optimisation tool.

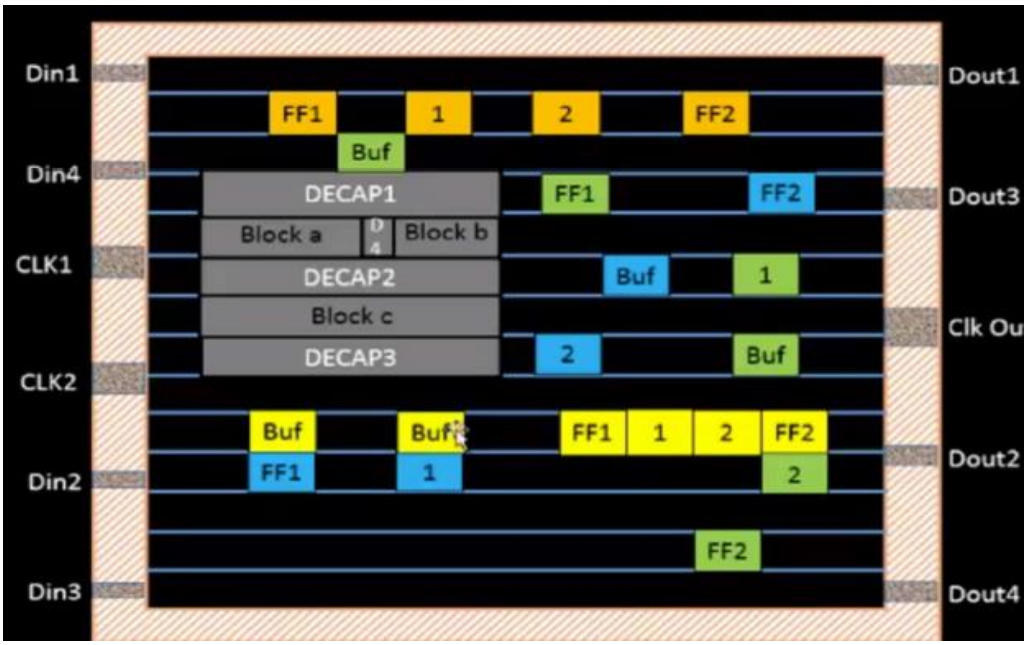


Figure 2.2: Snippet of the chip after Placement

2.2.3 Clock Tree Synthesis

Then we have the timing issues and the clock and data path in the physical design flow, we have to generally study, the static timing analysis, and the clock synthesis. Figure 2.3 depicts the chip have the clock path connected with the flip-flops using the H-Tree algorithm.

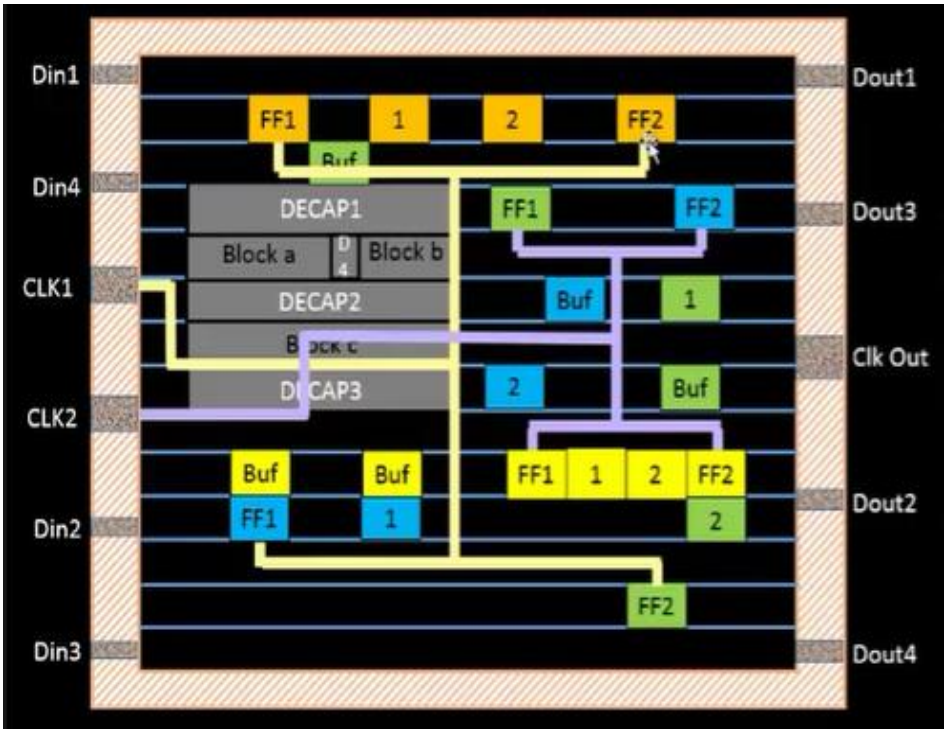


Figure 2.3: Snippet of chip after Clock Tree Synthesis

2.2.4 Clock Net Shielding

Then we have the shielding of the clock nets as clock is the most critical net so we shield them so that it can be glitch free. So here we study the signal integrity issue, which is generally a crosstalk, which is a combination of glitch and the data delays, which will affect the timing. Figure 2.4 shows that the clock nets have been shielded by the lines which are not fluctuating, to reduce the effect of glitch in clock signal.

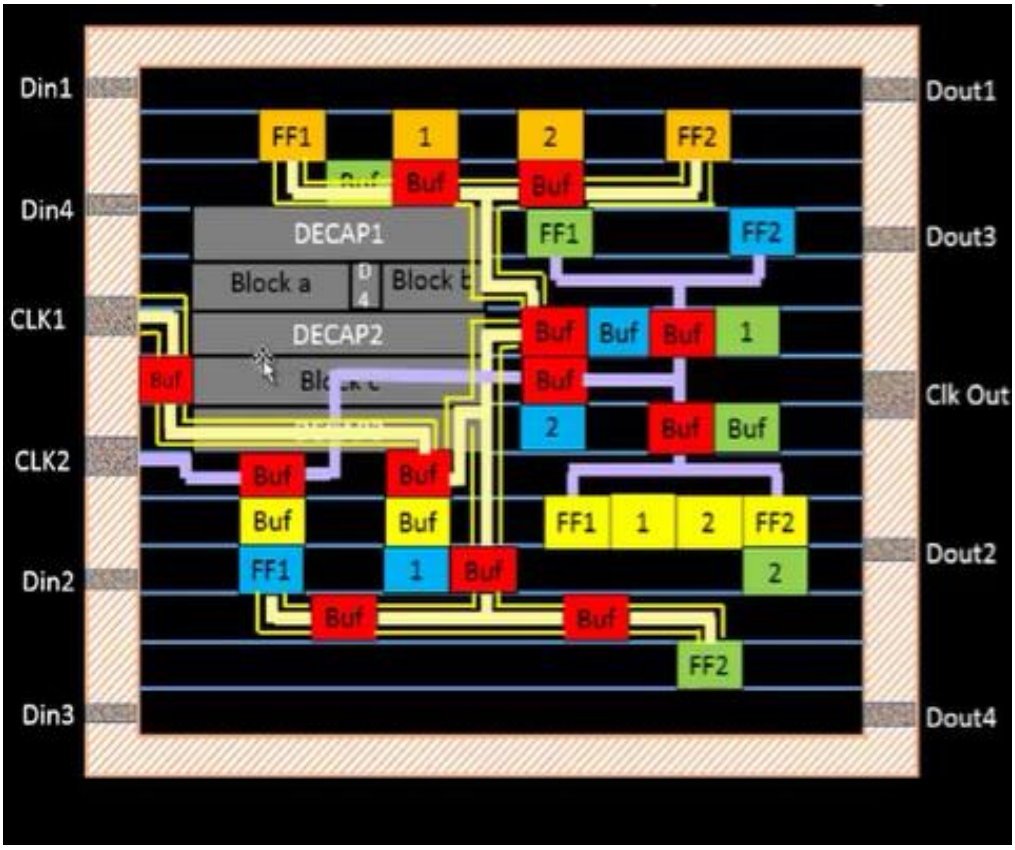


Figure 2.4: Snippet of chip after Clock Net Shielding

2.2.5 Routing

After the shielding of the clock net, we have to route the data for the proper functioning of the SOC. The routing of the data path can be done by various ways using the shortest path algorithms where we have the source point and the destination point and the connection between them is generally, the L shape. Figure 2.5 depicts the routing of the data path, and if some data paths are critical, which means that if data path is crucial for the timing and performance point of view, then it can also be shielded as like the clock nets.

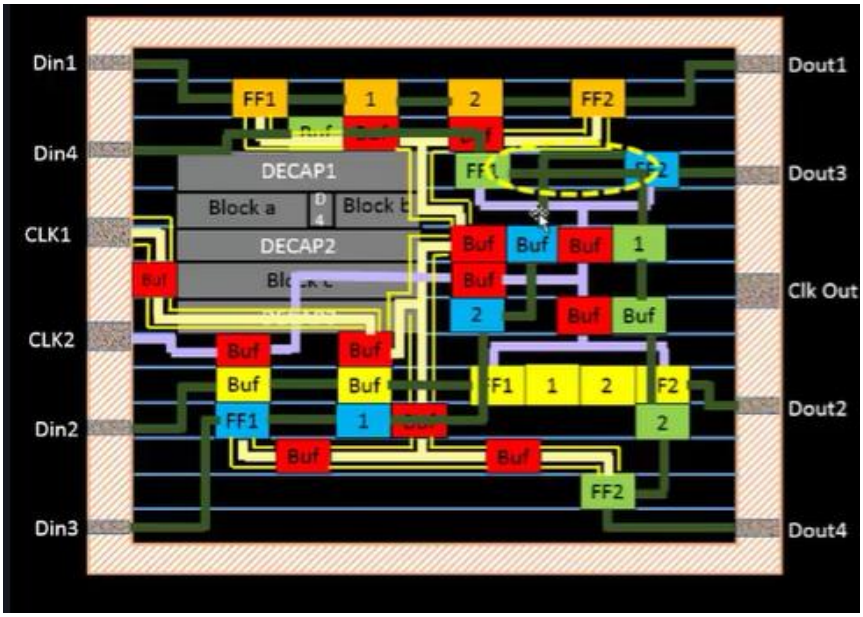


Figure 2.5: Snippet of chip after Routing

2.2.6 DRC and Parasitic Extraction

Then the last step of the physical design flow is the violation checks of the data wires, which contains the DRC checks and parasitic extraction, and then after the final step we have the static timing analysis for all the real clock path and the data path. Figure 2.6 shows the parasitic resistances and capacitances which are present in the SOC because of the interconnection of the wires and the nets running very close to each other.

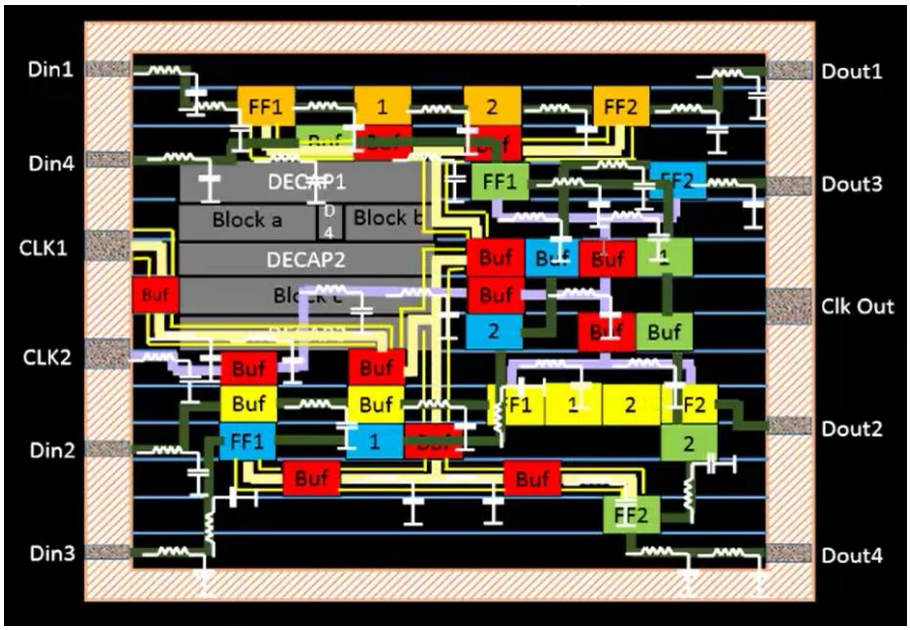


Figure 2.6: Snippet of chip after Parasitic Extraction.

2.3 FLOOR PLANNING

There are series of steps that are followed in floor planning. The sequence of steps that are followed are:

1) Define width and height of core and die.
2) Define the location of preplaced cells.
3) Surround the preplaced cells with a decoupling capacitor.
4) Power Planning
5) Pin placement
6) Logical Cell Placement Blockage

2.3.1 Width and Height of core and die

The first step of the physical design flow is floor-planning which includes many small steps, the initial one is the utilization factor and the aspect ratio of the core and the die. The first step is to measure the width and the height of the core and die so that the designer can get the overall area where he can do the placement of the blocks.

Here, the utilization factor is calculated and the aspect ratio is calculated in terms of the gates and in terms of the area of the core and the type of the technology node used, which is effectively used by the core and by the interfaces. Utilization factor can never be 100% since we want space between the cells, so that there can be an interconnection amongst the blocks on the same layer. Figure 2.7 is showing the core and die of the chip, where the actual logic of the chip will be placed in further steps. Figure 2.8 is showing the calculation of the effective area of the logic which is used inside the chip for the desired functionality. Figure 2.9 depicts the utilization factor, which means the amount of area of the core being utilized by the logic to the total amount of the area of core.

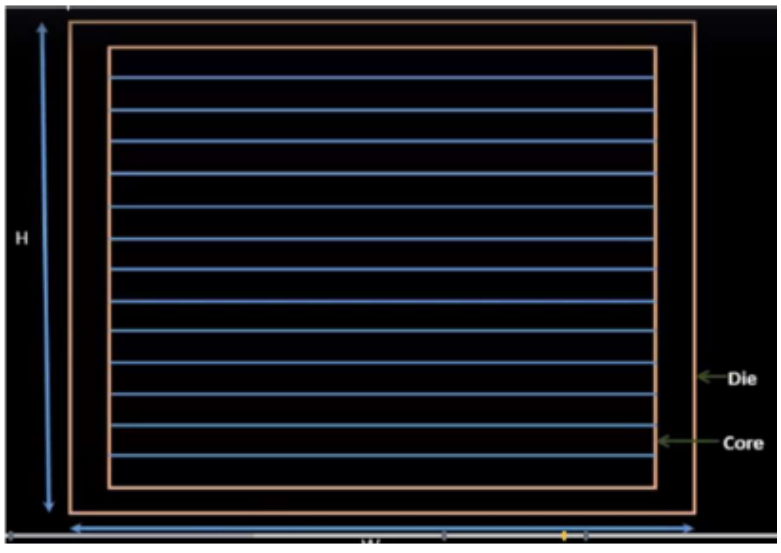


Figure 2.7: Snippet of chip showing the width and Height of the core.

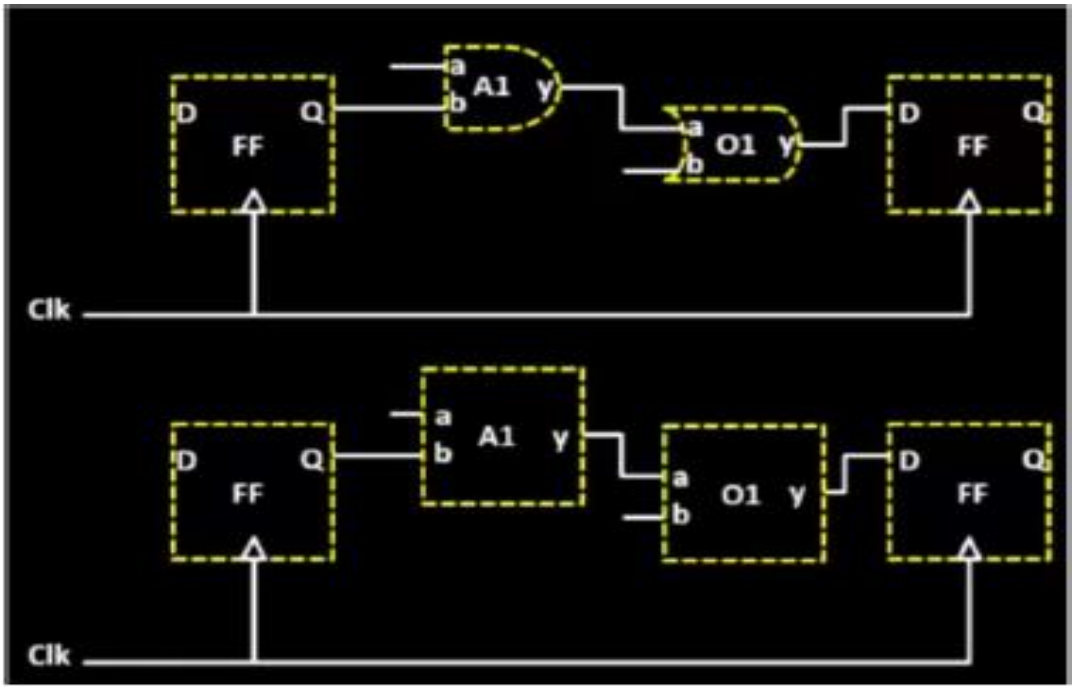


Figure 2.8: Snippet of chip showing the effective area calculation.

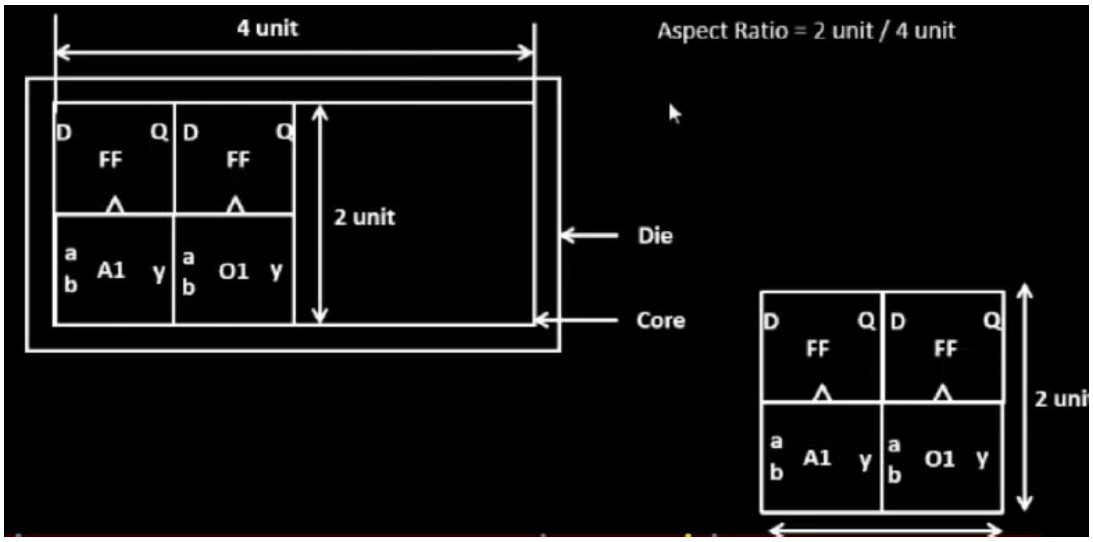


Figure 2.9: Snippet of chip for finding the utilization factor.

2.3.2 Pre-placed cells

After the calculation of the width and the height of the core and utilization factor, pre-placement cells are included in the core. These pre-placement cells are basically the IPs/ IOs used frequently into the design and in the big SOCs, so that they can be available in each and every session in which they are required. For example, some of the common IPs, which we generally used are

some Digital IPs, like adders, multiplexers, some Analog IPs like regulators, crystal oscillators, so these are some of the examples of IPs which frequently use, so these are considered under the pre-placement cells. Figure 2.10 shows the logic which is commonly used in the SOC. Figure 2.11 shows, how the common logic can be converted into black box and can be used as the IP. Figure 2.12 shows the commonly used macros in the SOC Design which contains both Analog and Digital IPs, IOs and Macros.

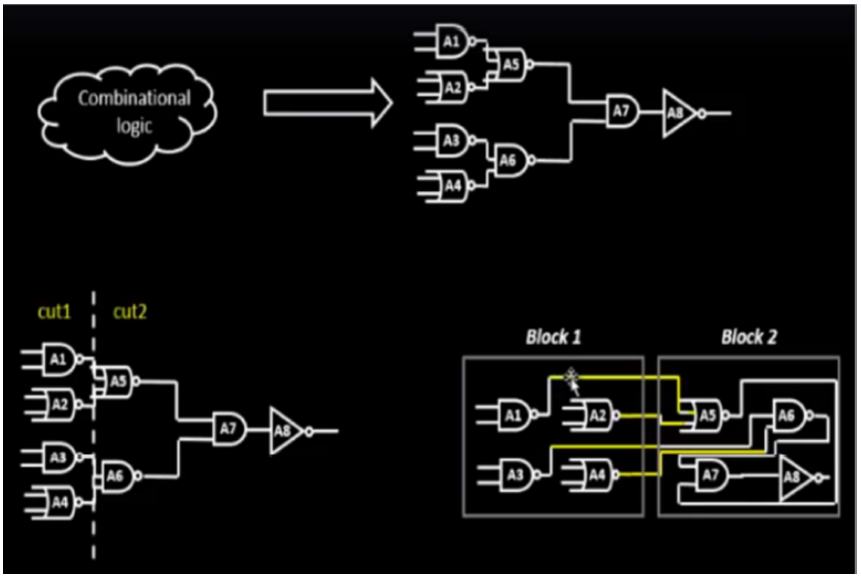


Figure 2.10: Snippet of chip making interconnection of IPs.

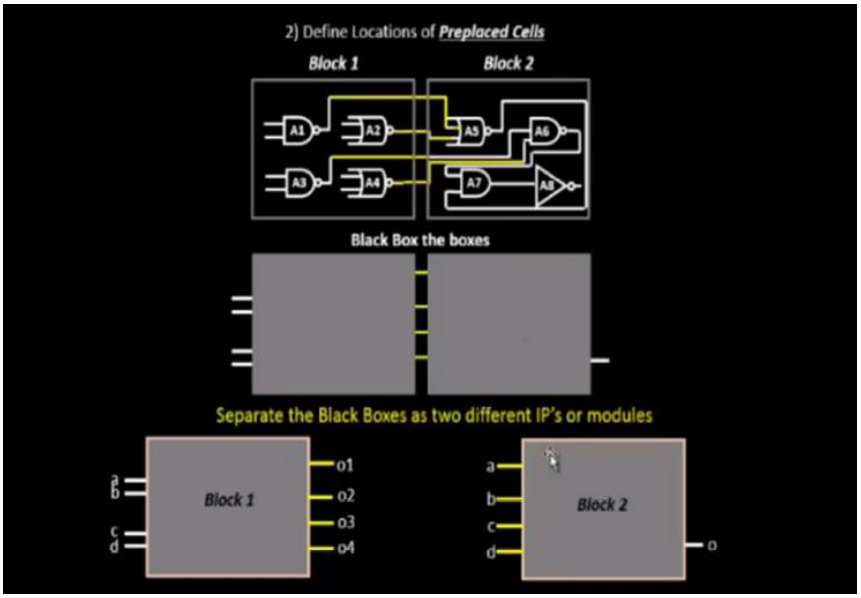


Figure 2.11: Snippet of chip for Black-Boxing the logic.

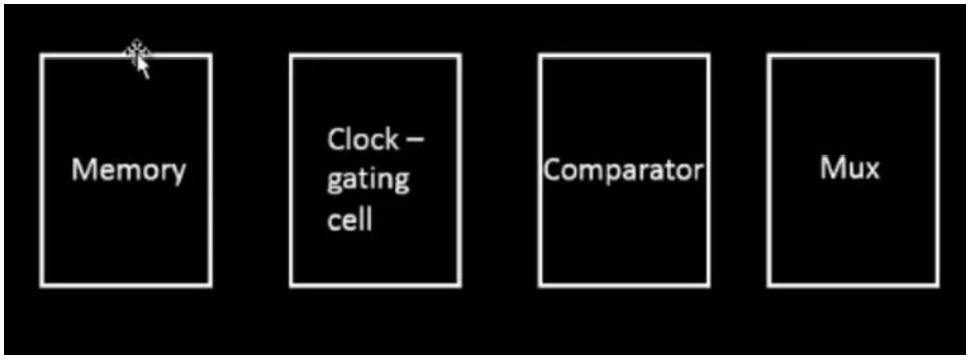


Figure 2.12: Snippet of chip for commonly used Macros.

2.3.3 Use of Decoupling Capacitance

Then the next step is to include the decoupling capacitors. The decoupling capacitors are used in this circuit so that there can be a measure for the unwanted saturations in the power supply. As some of the common problem of Voltage drawn and ground bounce occurs in the SOC, so these decoupling capacitors provide a backup supply to the cells, when there is a need of power. Figure 2.13 shows the logic which is not having the Decoupling capacitance. Figure 2.14 shows the logic which is having the capacitance for resolving the issue of voltage drop and ground bounce. Figure 2.15 shows the placements of Macros and Decoupling capacitances.

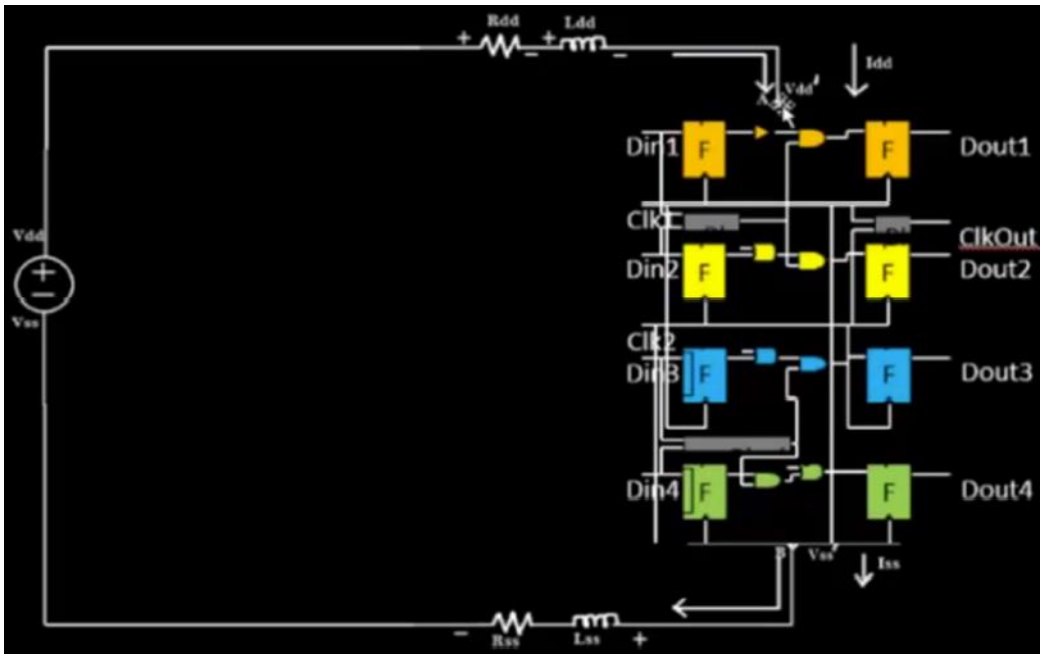


Figure 2.13: Snippet of chip showing cells without Decoupling Capacitor.

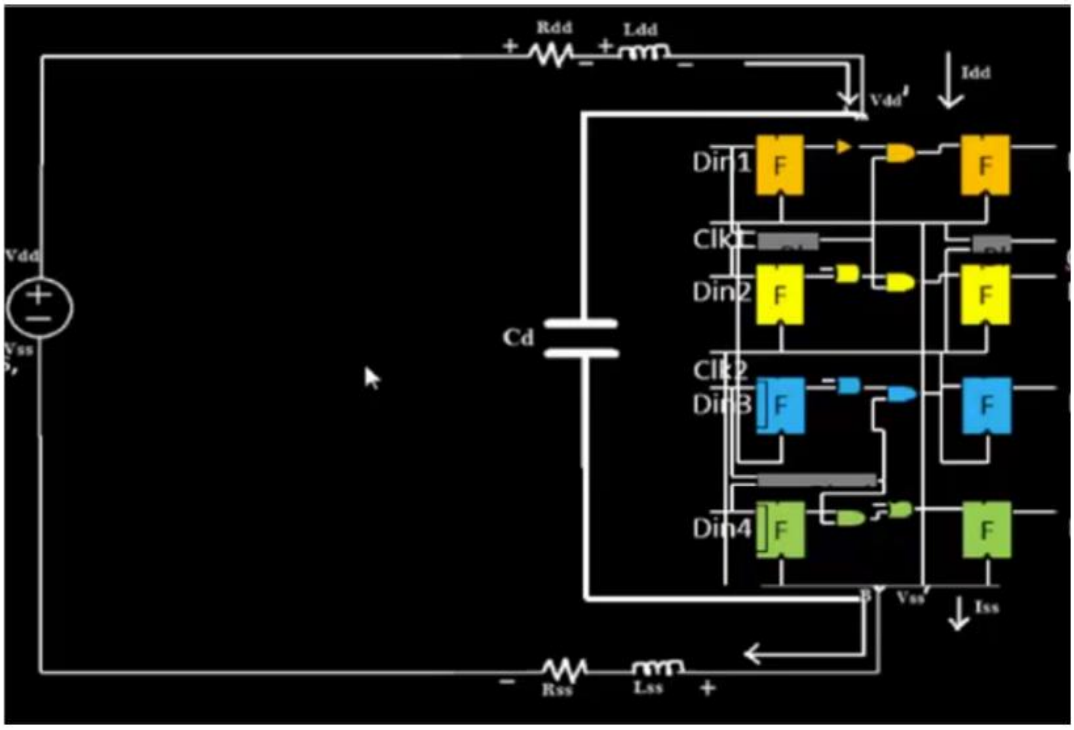


Figure 2.14: Snippet of chip showing cells with Decoupling Capacitor.

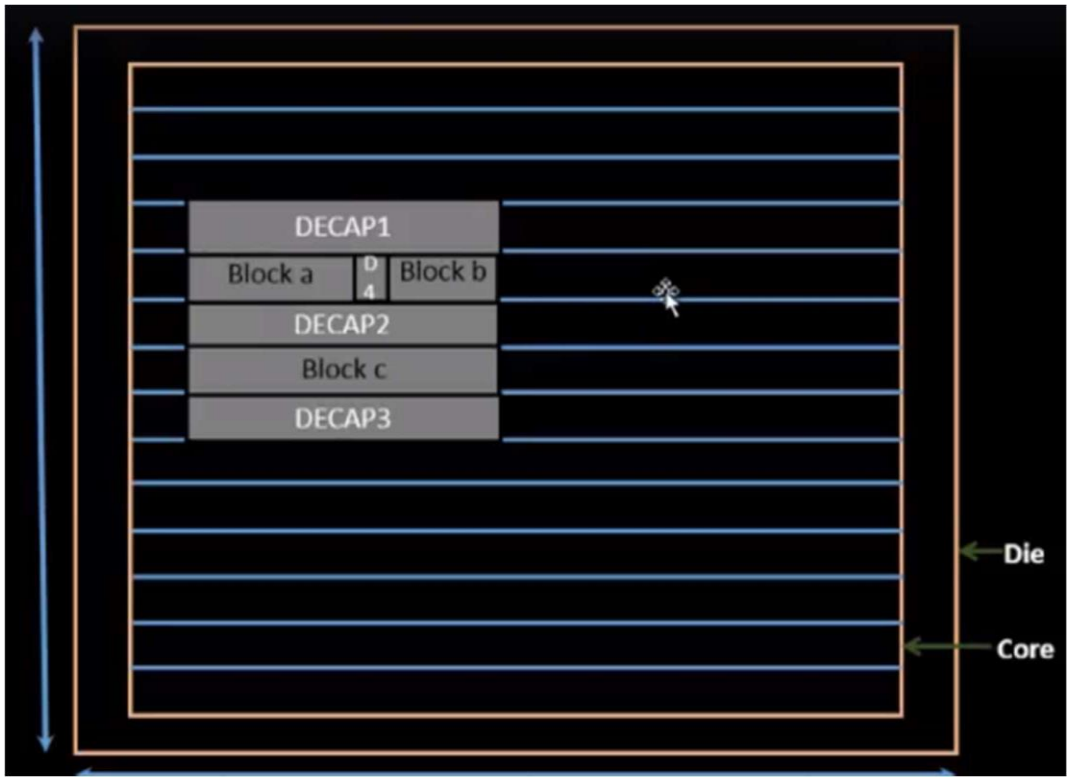


Figure 2.15: Snippet of chip showing placement of Decoupling capacitances.

2.3.4 Power Planning

The next step after the Decoupling capacitance is the Power-Planning, where generally, the names of the powers are made, the layer above the blocks, so that each and every block can get the continuous supply of the VDD and the VSS for the correct operation. And there would be the minimum delay amongst the power nets and the wires so that they are within the noise margins, and the effect due to parasitic delays can be avoided. Figure 2.16 depicts the logic high line and logic low line supply, which is connected to each logic, so that there will be no problem of supply rails while the particular block is not active mode, or in power save mode. This is basically the local power connection technique. Figure 2.17 depicts the global power grid for the entire SOC, so that the appropriate power rails is available within the short distance to each and every block of the IP or SOC.

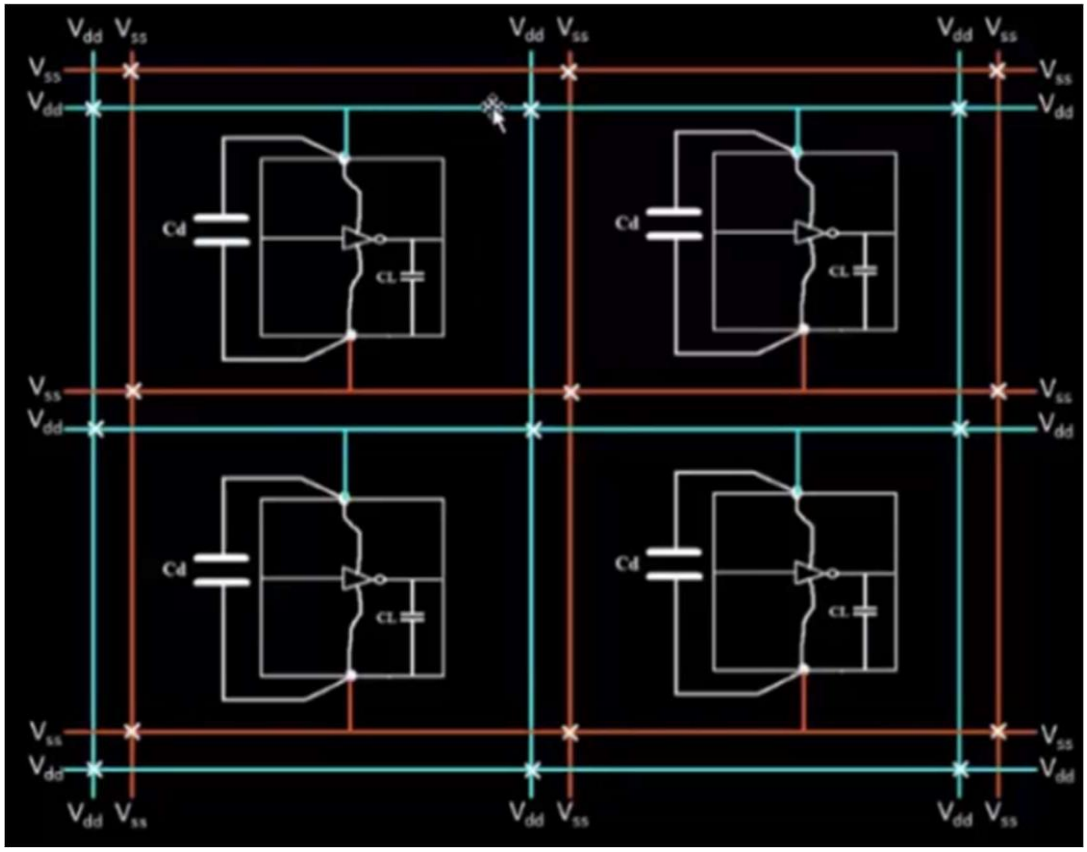


Figure 2.16: Snippet of chip showing Power Planning

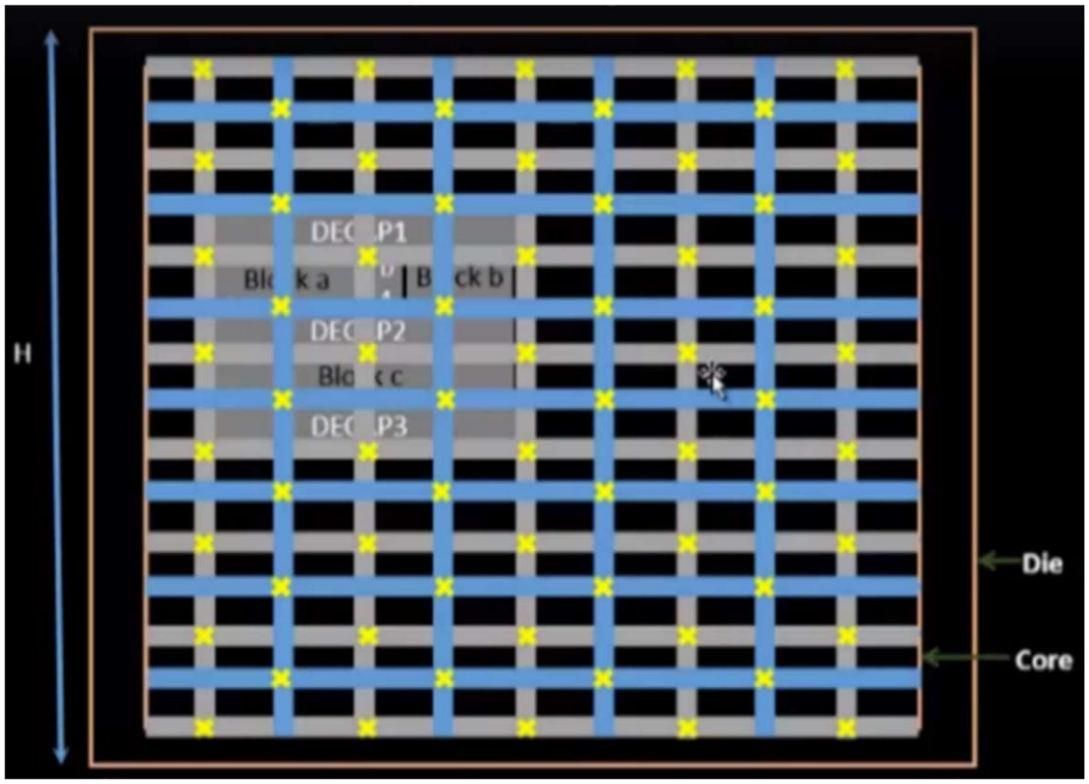


Figure 2.17: Snippet of chip showing Mesh arrangement of Power Rails.

2.3.5 Pin Placement and Logical Cell Placement Blockage

And the final step is logical cell placement blockage, where the pins are placed according to the need of the SOC design, the pins are placed according to their interfaces inside the code and logical cell placement blockage is done to block the area of the core where we don't have to place any cells, these areas can be blocked due to the other block which is inherently present there, or avoid to run the placement optimization tool. Figure 2.18 shows the arrangement of the pins in the die location which depends on the requirement given by the design team for the timing constraints. Figure 2.19 depicts the Blockage of the placement tool, so that no logical cell could be allowed to get placed onto that specific region where it should not be.

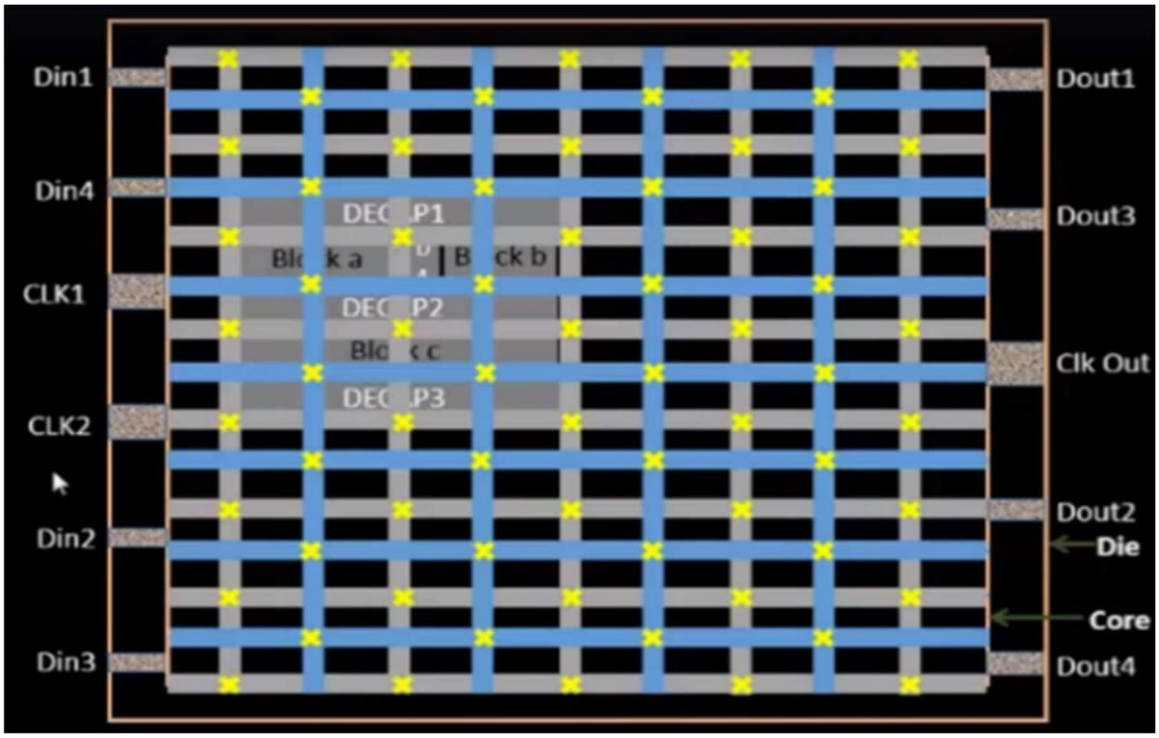


Figure 2.18: Snippet of chip showing Pin Placement

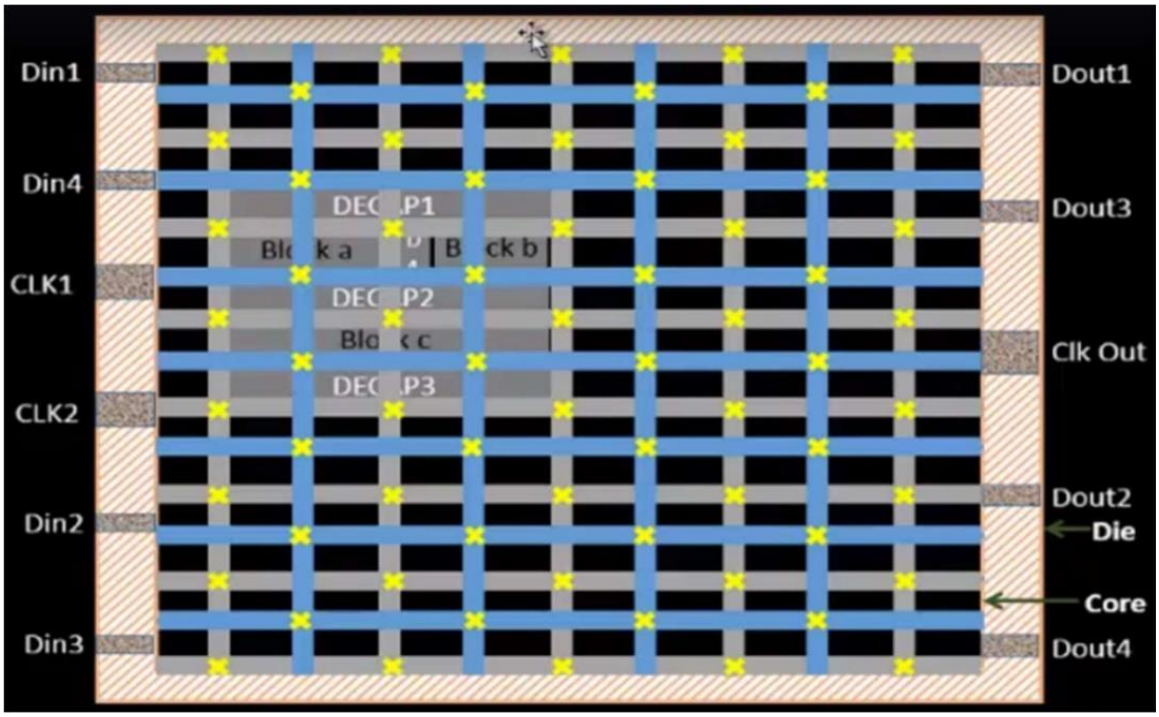


Figure 2.19: Snippet of chip showing Logical Cell Placement Blockage.

2.4 PLACEMENT

The next step after the Floor Planning is the placement process, it carries various steps starting with binding the netlist into the physical cells with the help of technology node present in the library of the particular dataset of the organization which is generated after the synthesis part in the design flow.

Following steps are performed in Placement

1. Bind the netlist with Physical Cells.
2. Placement
3. Optimize Placement

2.4.1 Physical cells in the netlist

Netlist contains many IPs, IOs, S/Ds that are technology dependent so we have to link those blocks or cells to some technology node. There are the libraries present which contains different technology in the database, so the optimized gate with respect to time period, power and area is taken from the library and are integrated and placed into that core region. Figure 2.20 shows many logical cells which are present in the library and are needed to be integrated in the netlist.

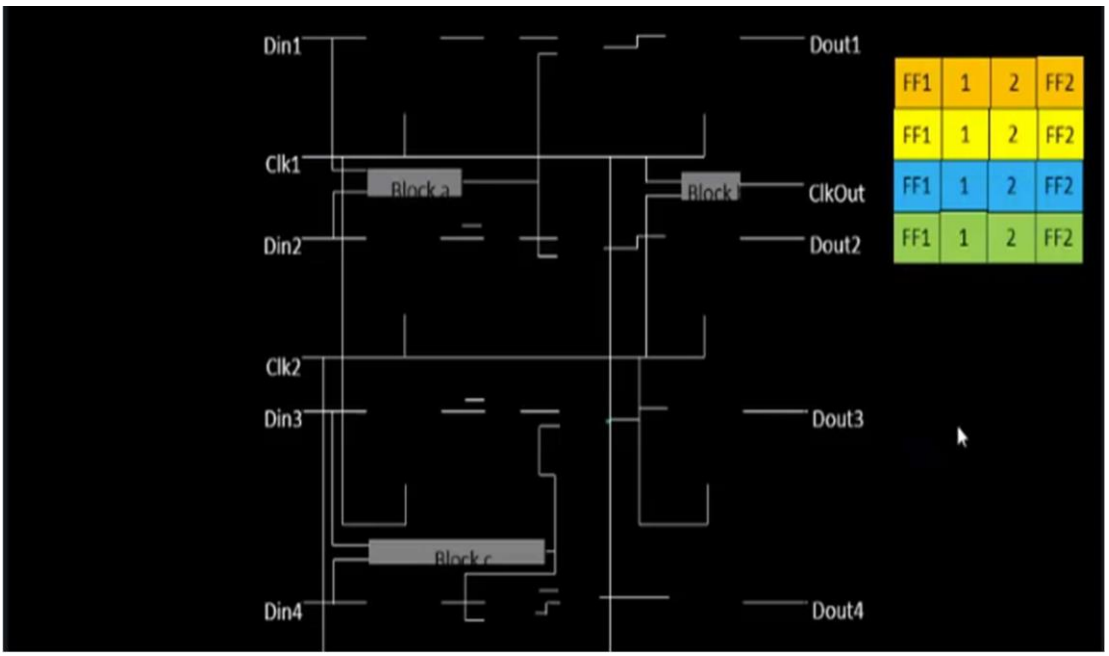


Figure 2.20: Snippet of chip showing Mapping of Library cells.

2.4.2 Placement of the logical Blocks

The placement of the blocks is done based on the requirement of the design of the IP or SOC. All the blocks which have common path of data are placed to each other. Placement is done with the help of automatic tools which exclude the region of pins and location of pre-placement cells. Figure 2.21 shows the placement of the blocks based on the EDA tool.

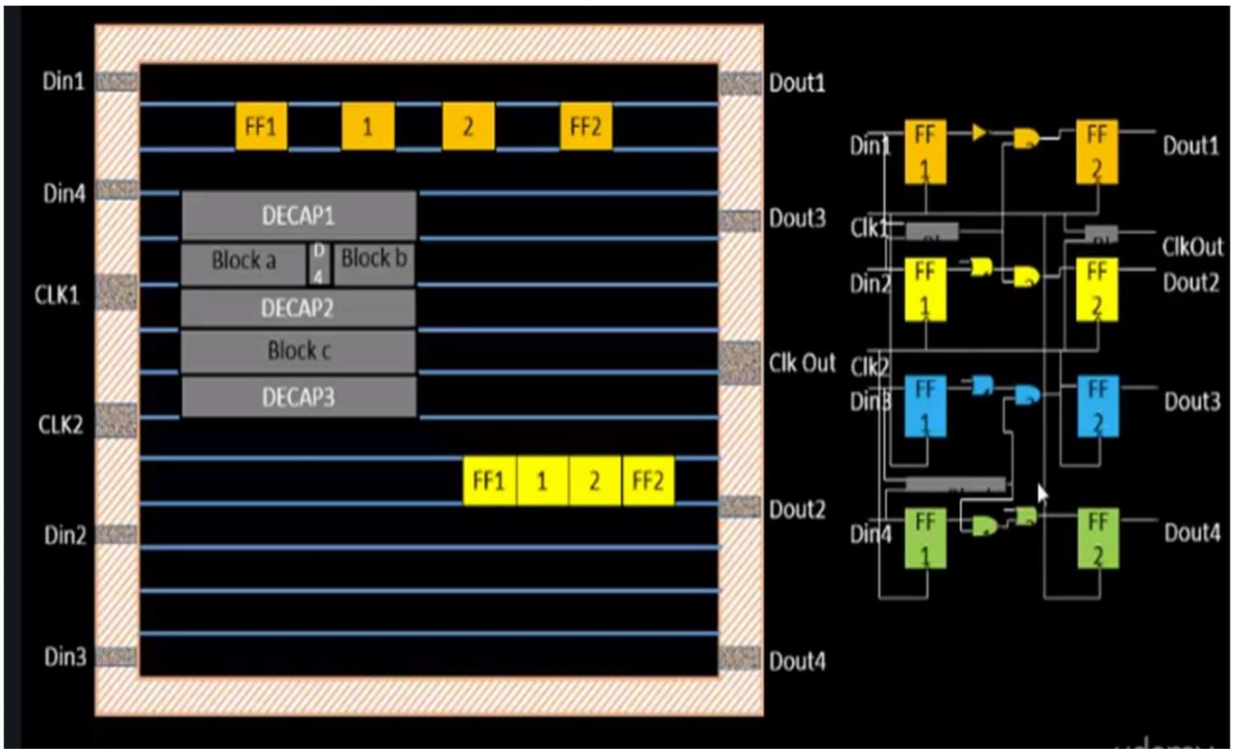


Figure 2.21: Snippet of chip showing Placement of Logical Cells.

2.4.3 Optimised Placement

Optimised placement is done to ensure the correct functionality of the data path, while transferring the data from one logic gate to another. There are many parasitic resistances and capacitances present in the SOC, due to which there can be the effect of signal integrity issues and the most common is of crosstalk which can disturb the original signal. Thus, to eliminate or to minimise the effect of glitches and crosstalk, buffers are added in the path to maintain the signal strength of the data. Optimize placement is done using the estimated wire delay and lumped capacitances to measure the delays, which can change the

strength of the signal. So, accordingly optimized placement is done. Figure 2.22 shows the introduction of the buffers inside the Core, so as to improve the quality of the signal being transferred from one logic gate to another, depending on the relative connectivity of the logical blocks.

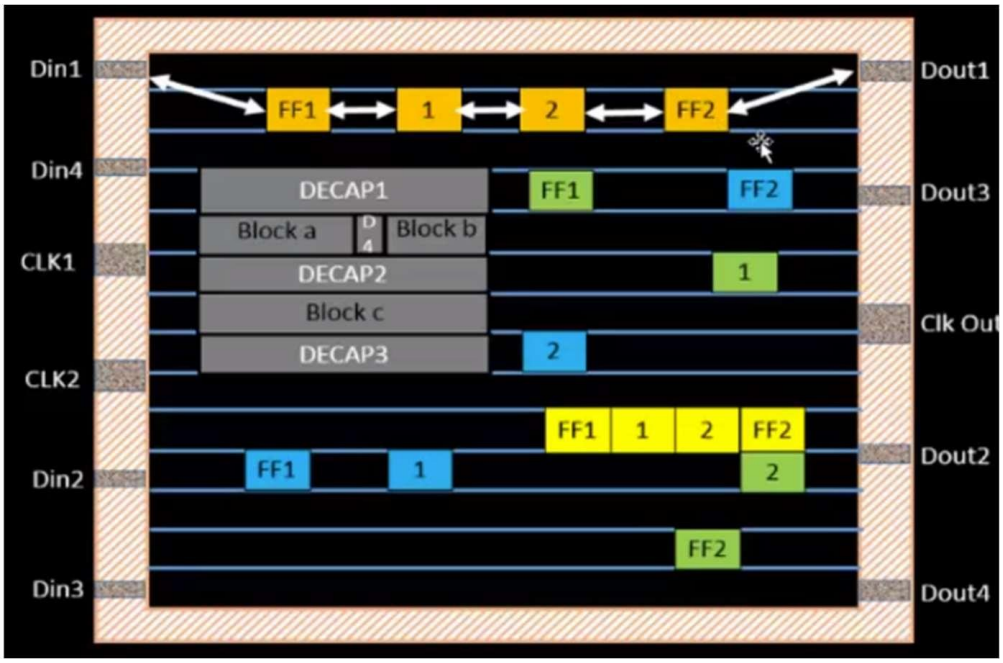


Figure 2.22(a)

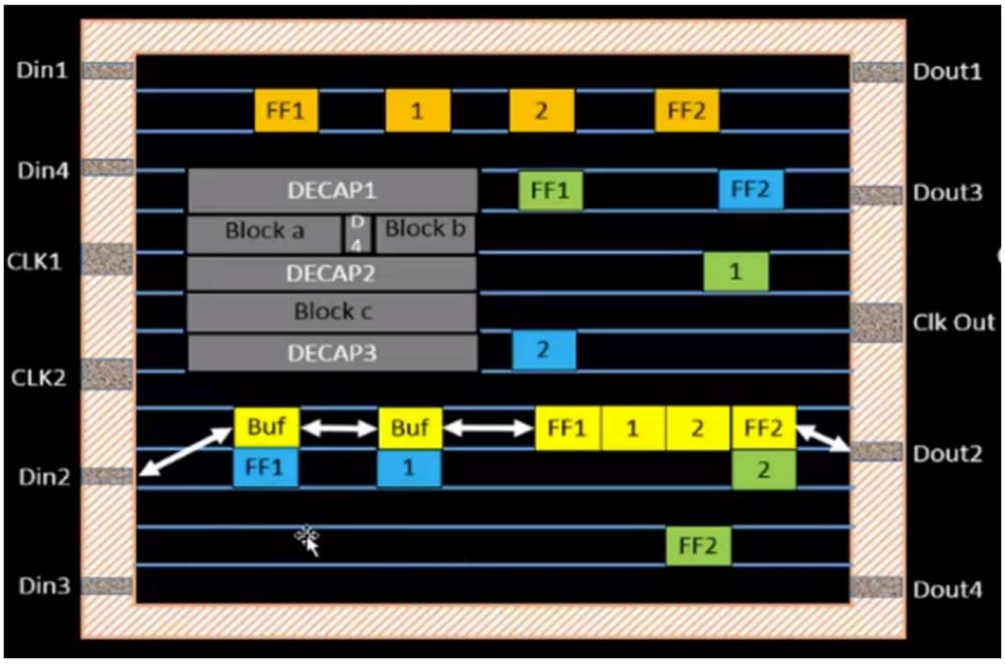


Figure 2.22(b)

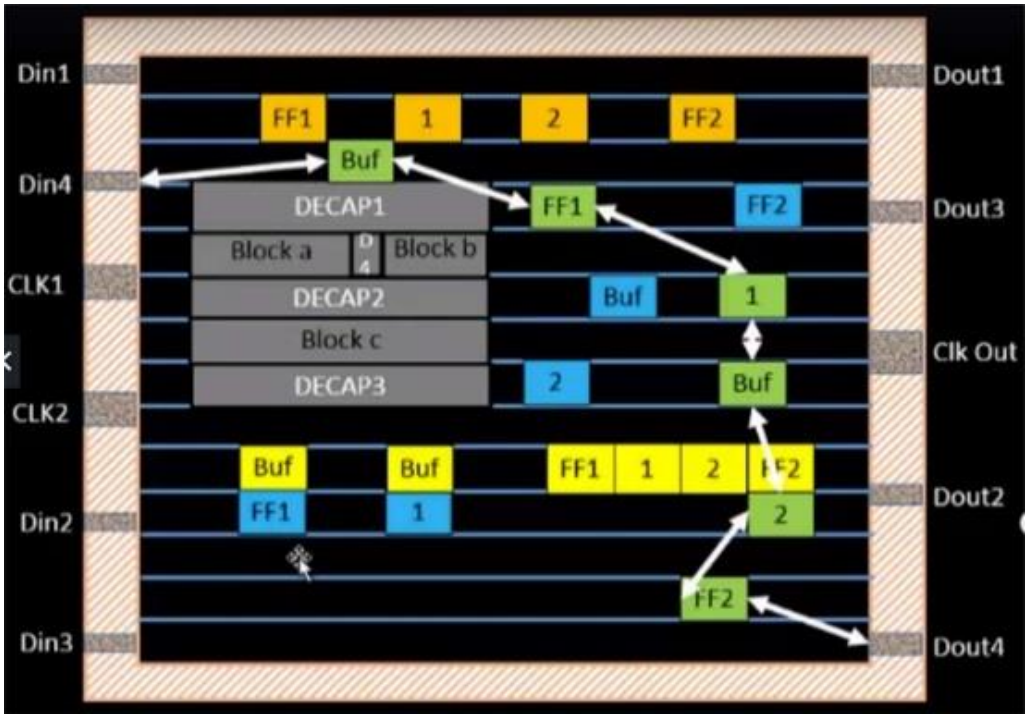


Figure 2.22(c)

Figure 2.22: Snippet of chip showing Optimised placement with inserting Buffers.

2.5 Static Timing Analysis with Ideal Clocks

Now since for the connection of the clock, for the proper functioning of the SOC, there are some measures like the data delays, and the slew rate, latency, should be studied so that the signal integrity should be maintained.

The next step after the placement is to perform the timing analysis with the estimated clock nets and the estimated data wire delays. Here the timing analysis performed on the basis of the setup and hold constraints, and the introduction of the clock jitter and uncertainty is also engaged so to perform the clock domains in a more efficient manner. After the static timing analysis, with a single clock. the STA is performed with the help of multiple clocks. Multiple clocks are introduced in this method and the multifrequency clocks, multi-level clocks are introduced. Figure 2.23 shows one of the stages of pipelined architecture where there are two F/Fs, one is Launch (Left) and the other is Capture (Right) and Setup timing constraints have been verified with real clock. Figure 2.24 shows the calculation of Hold time constraints with real clock.

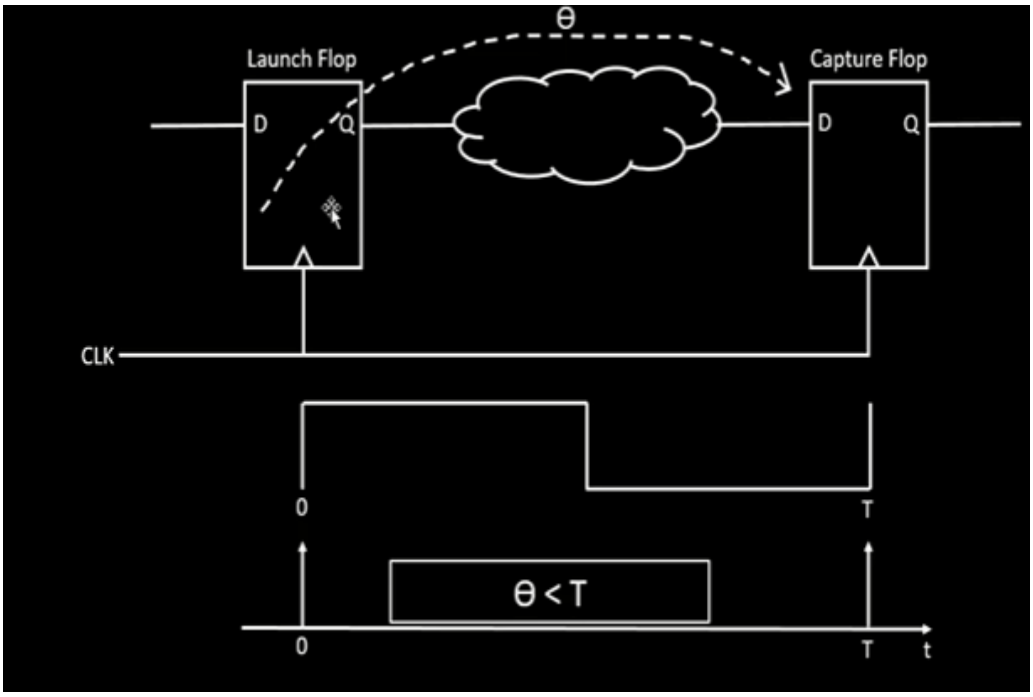


Figure 2.23: Snippet of chip showing Setup time analysis for ideal clocks.

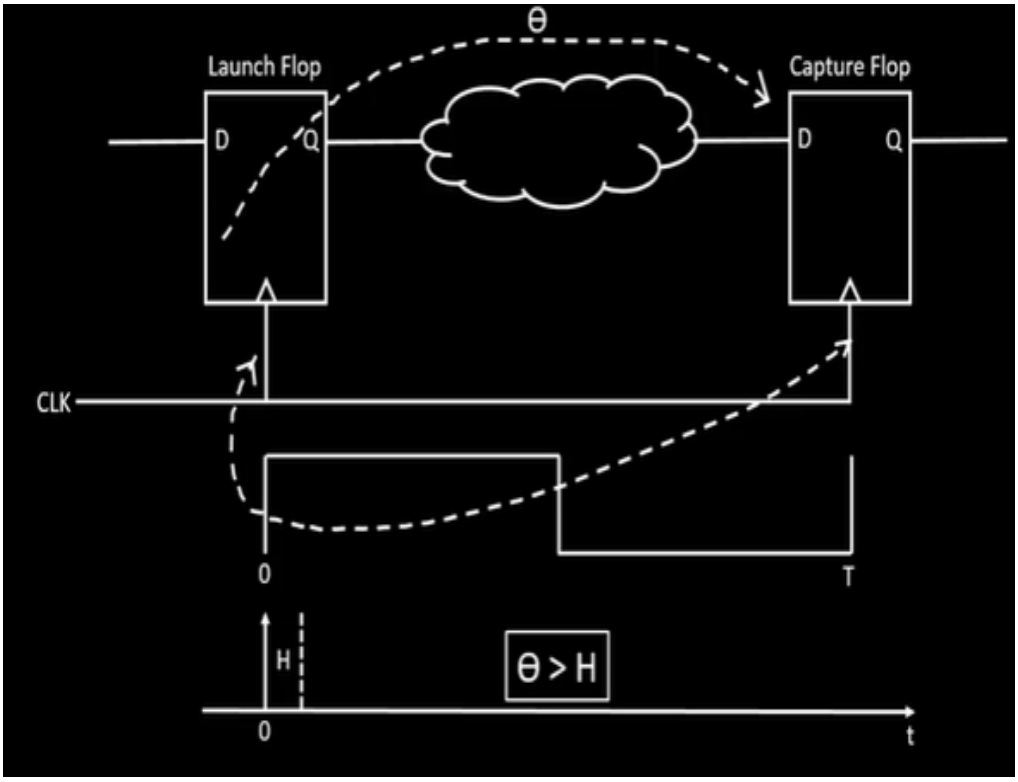


Figure 2.24: Snippet of chip showing Hold time analysis for ideal clocks.

2.6 CLOCK TREE SYNTHESIS

Now, after the static timing analysis status check is done. To maintain the clock net and the data net of the SOC functional correct, the next step is clock synthesis, where the clock path is analysed and synthesis is done in between the clock signals so that the signal integrity can be matched accordingly.

The general algorithm which is used in the clock tree is the tree algorithm, where the main objective is to minimize the latency, and to reduce the skew for launch and capture F/Fs. Figure 2.25 shows the clock pin CLK1, which is needed to make the connection to the desirable F/Fs.

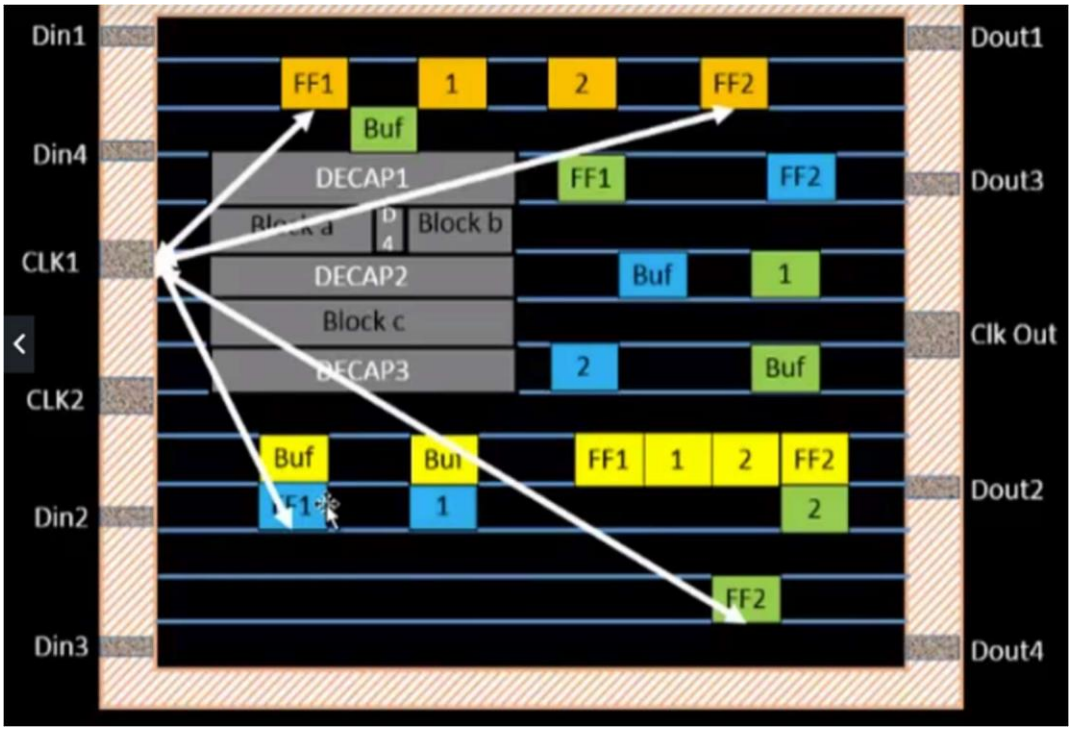


Figure 2.25: Snippet of chip showing Clock available at CLK1.

For Clock there are various quality parameters check that are needed to be verified. The six quality parameters checks are:

1. SKEW
2. PULSE WIDTH
3. DUTY CYCLE
4. LATENCY
5. CLOCK TREE POWER
6. SIGNAL INTEGRITY AND CROSS TALK

Figure 2.26 shows the typical implementation of the H tree, which is used to make the skew as low as possible by eliminating the latency difference between the Launch F/F and Capture F/F which will be working and transferring the data from the same clock pin. Figure 2.27 depicts the H-tree implementation into the SOC, where the shape of 'H' can easily be visible through the clock path.

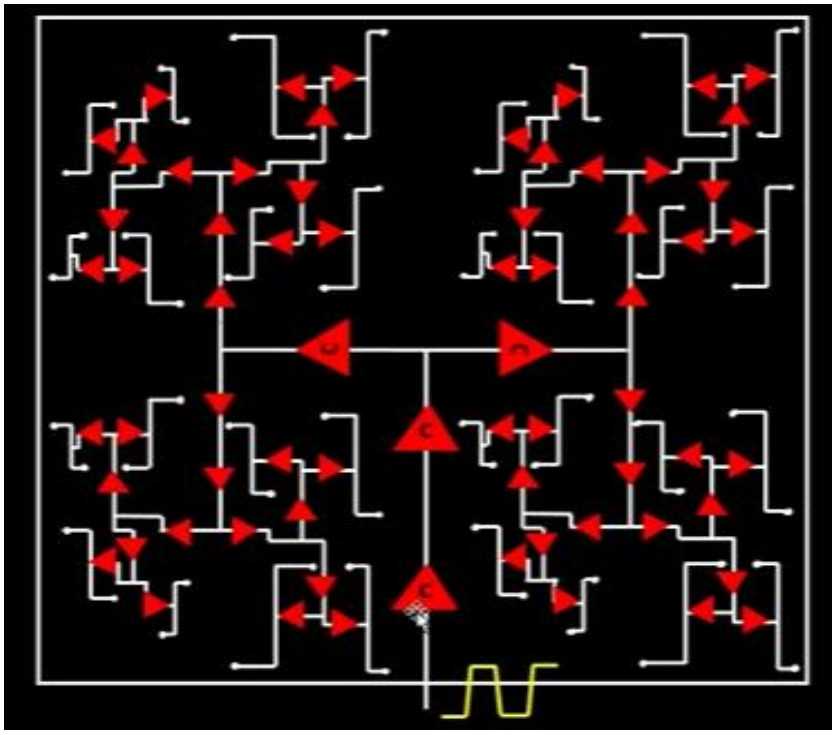


Figure 2.26: Snippet of chip showing Clock Tree Synthesis

Figure 2.27(a)

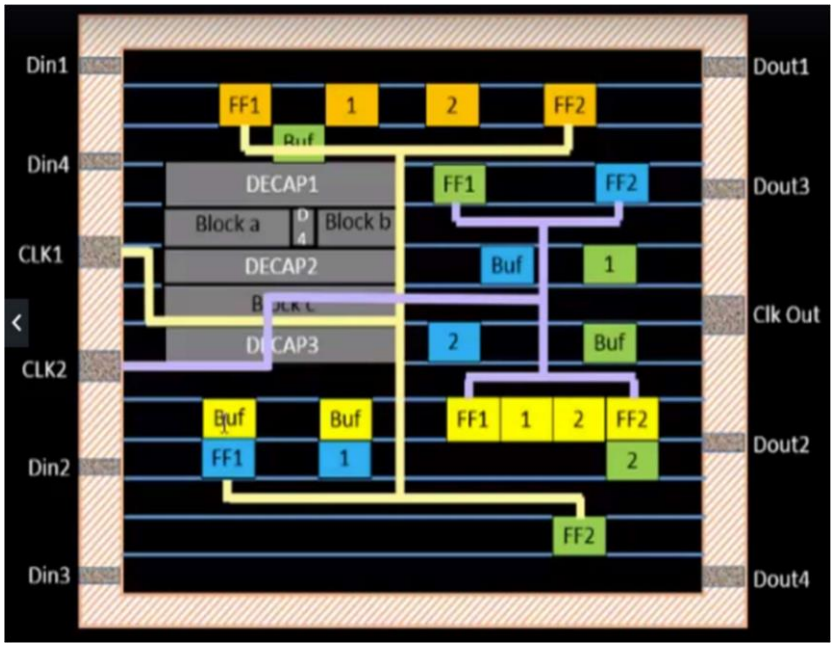


Figure 2.27(b)

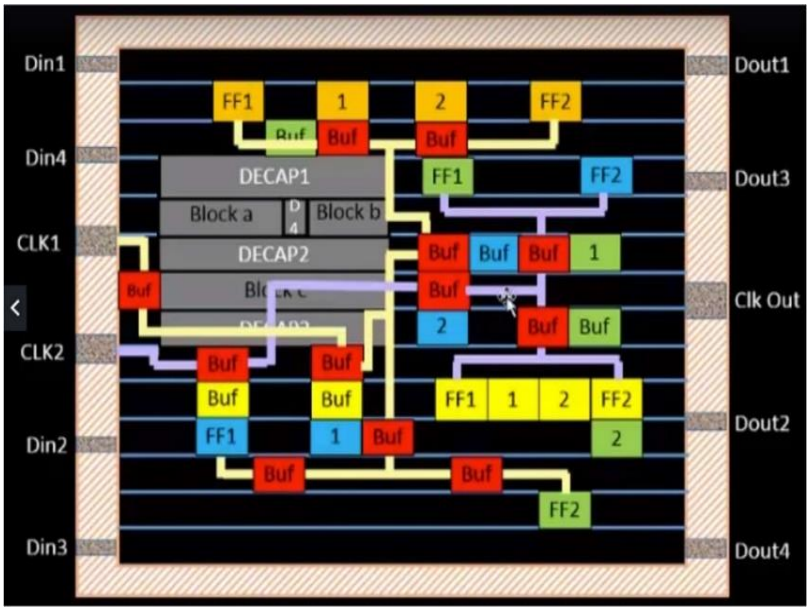


Figure 2.27: Snippet of chip showing Clock path using H-tree.

There are various checks which are needed to be performed while doing the proper synthesis of clock tree. Latency is basically the time, from the pin of the clock to the input of the F/F, the clock skew is basically the latency difference between the two F/Fs which are engaging in the data flow. These are generally termed as the launch clock path and the capture clock path. The third is the pulse width of the clock, it should be constant throughout as desirable. And it can be changed by the signal integrity issues and with the change in the strength of the signal is because of the parasitic resistances and the capacitances in the SOC. The another is the duty section of the clock. This is the parameter to be checked, and the clock power and signal integrity of the clock tree. These are the various parameters which are needed to take care by designing the clock tree synthesis.

2.7 CLOCK SHIELDING

After the CTS, another important point, which comes after the clock synthesis in the flow is that the clock is a critical net so it should be glitch free. The concept of crosstalk, and the interface amongst the clock were needed to be minimized and for that the net shielding is done, and after the clock shielding, static timing analysis is done with the real clock. Figure 2.28 shows the view of clock path which is shielded by the rails of either Vdd or Vss, so that there will not be the fluctuations on these lines and through which, the problem of glitch,

delta delay, change in the rise time and fall time, and change in the pulse width and duty cycle of the clock can easily be eliminated.

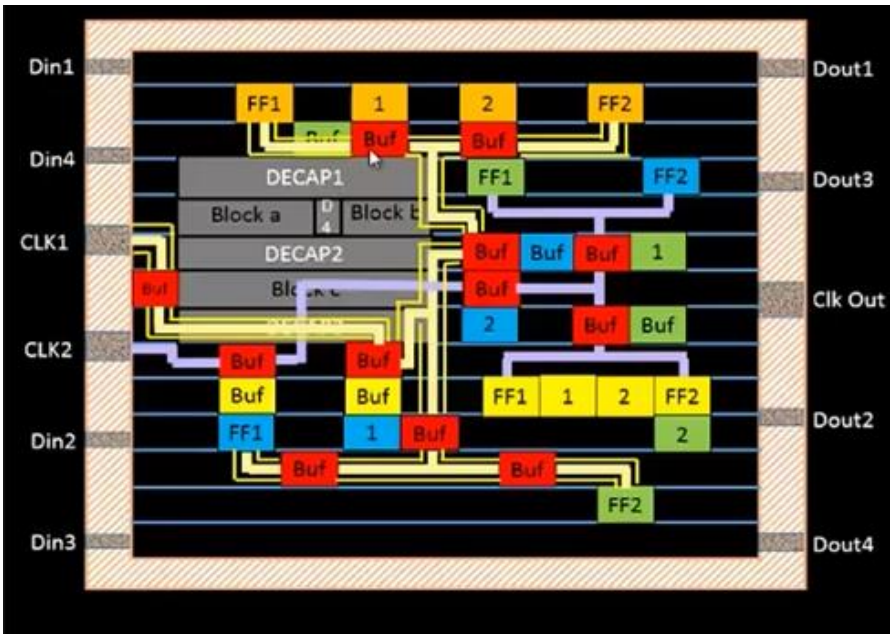


Figure 2.28: Snippet of chip showing Clock net shielding.

2.8 Static Timing Analysis with real clocks

After the clock net shielding, STA is done with real clocks, which includes the presence of buffers in the clock path. These buffers were not in STA when done with ideal clocks. The effect of Jitter and Skew is also been taken into consideration in the STA. Figure 2.29 shows the Setup time constraints analysis for real clock and figure 2.30 shows the hold time constraints for real clock.

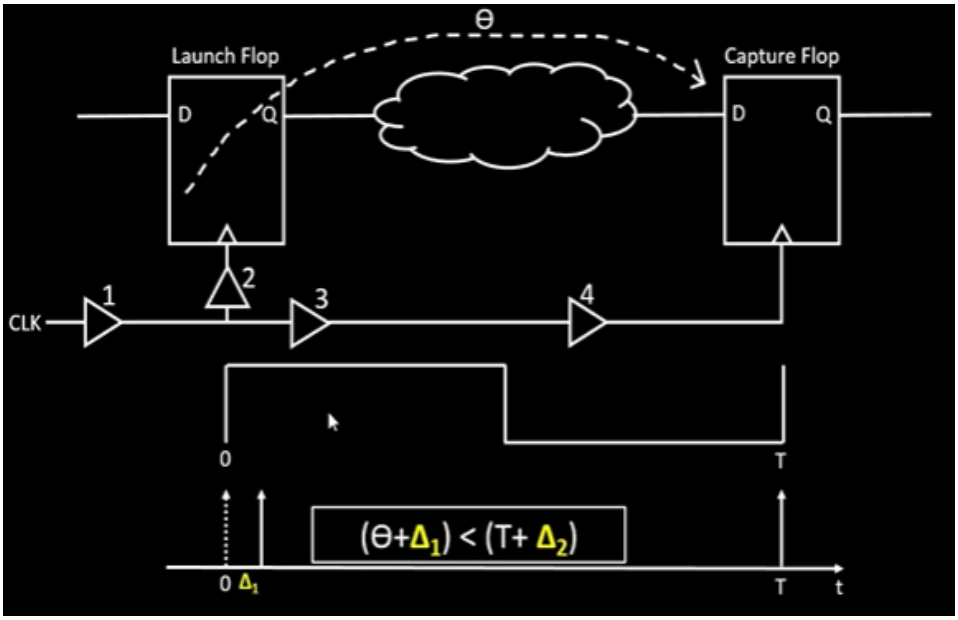


Figure 2.29: Snippet of chip showing Setup time analysis for Real clocks.

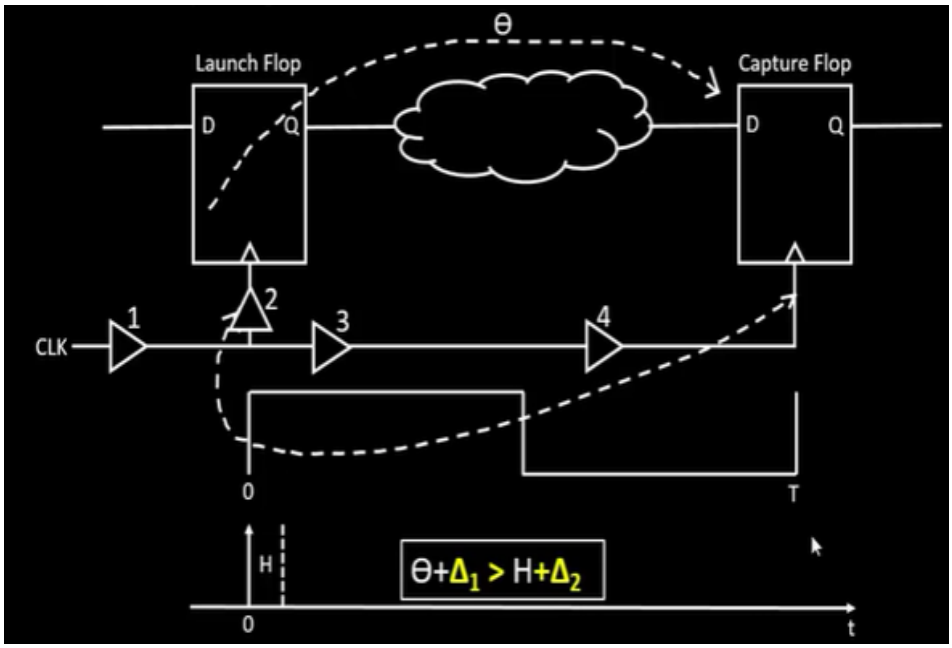


Figure 2.30: Snippet of chip showing Hold time analysis for real clocks.

2.9 ROUTING

The next step after performing the STA in the real clock path and in the estimated data path, then the routing is turned with the help of the maze algorithm that is maze routing. It includes various steps.

1. Finding the Source point and the sink point (destination point).
2. Mark grids on the core and die region.
3. Mark the points on the neighbouring cells in the increasing order while moving from source to destination cell location.

4. Trace the shortest path.
5. The path can be straight or L-shape depending on the availability of the connection between the two blocks.

Figure 2.31 shows the Maze Algorithm, where the numbers of the adjacent cells are increasing when the tool is moving from source point location to the destination point location. Figure 2.32 depicts the path taken by the tool based on the shortest path via the values of the adjacent cells. Figure 2.33 shows the SOC, where the data path is also routed after applying the Maze Algorithm.

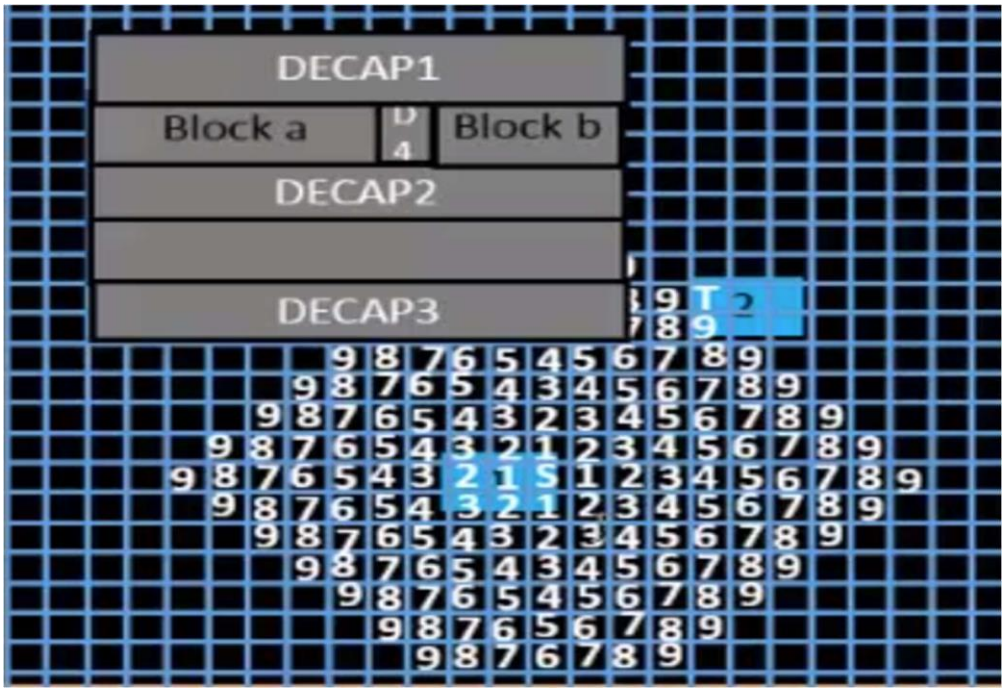


Figure 2.31: Snippet of chip showing Maze Algorithm for Routing.

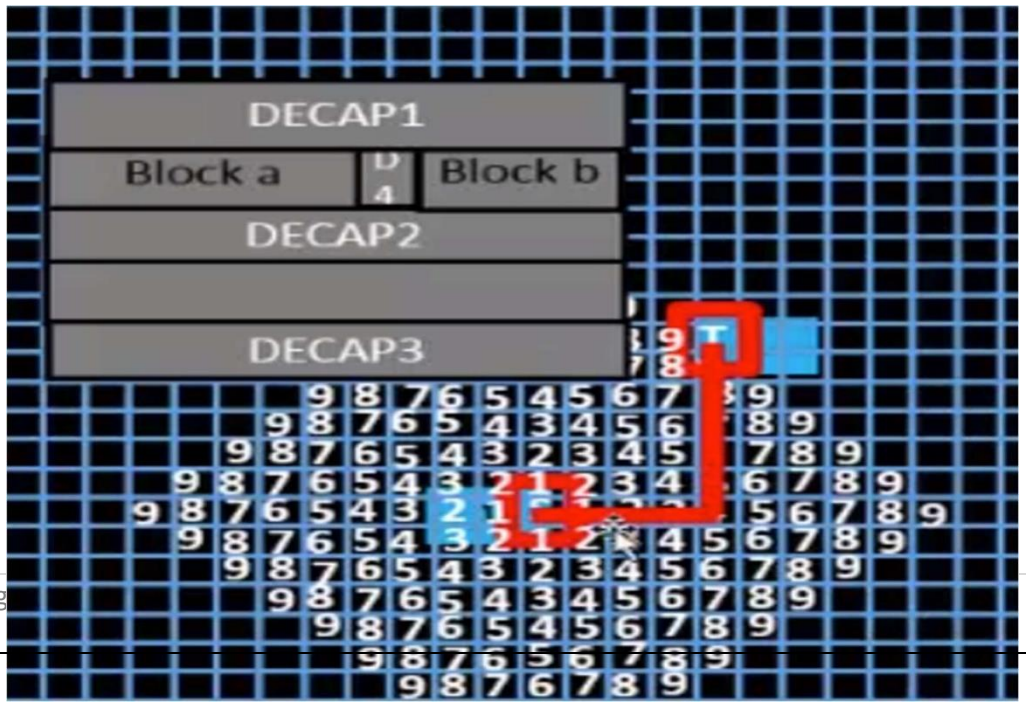


Figure 2.32: Snippet of chip showing Routing.

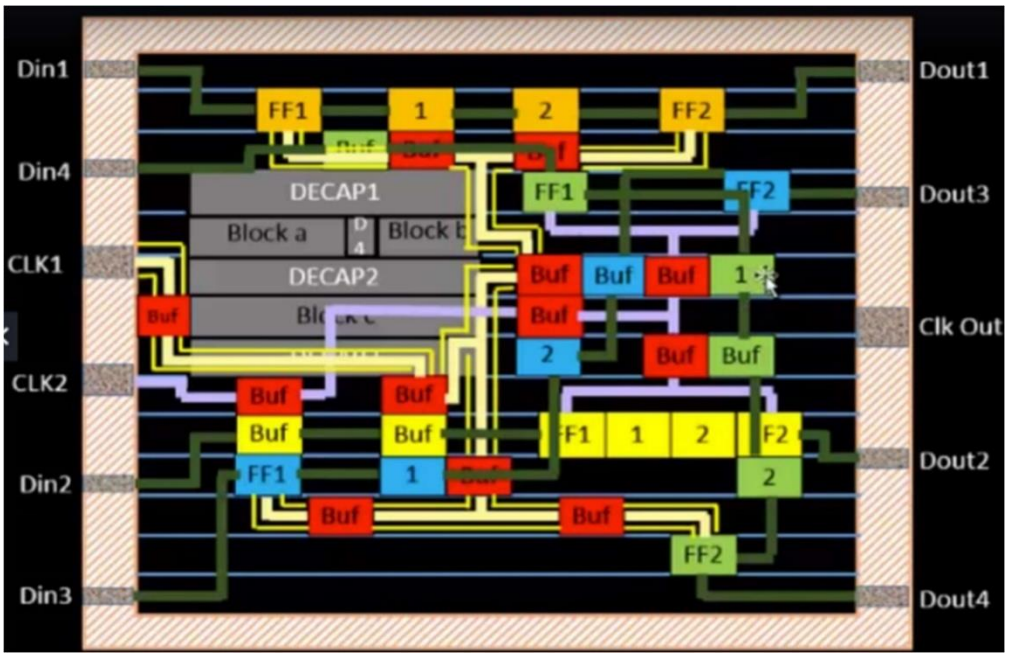


Figure 2.33: Snippet of chip after Routing the data path.

2.10 DESIGN RULE CHECKS

After performing the STA with real clock delay and real data path delay, the parasitic extraction has been done and design rule check have been implemented onto the SoC, so that it can be fabricated in the industry with the proper available resources of lithography and mask proximity. Lambda rules have been set by the foundry to make the layout a proper one and which can be used to generate the desirable silicon. Figure 2.34 shows some of the Lambda rules which are given by the foundry, based on their measure of approximations. The actual values needed to be filed by the Layout Engineer which can be greater than the Design Rule, but cannot be less, since it will violate the DRC

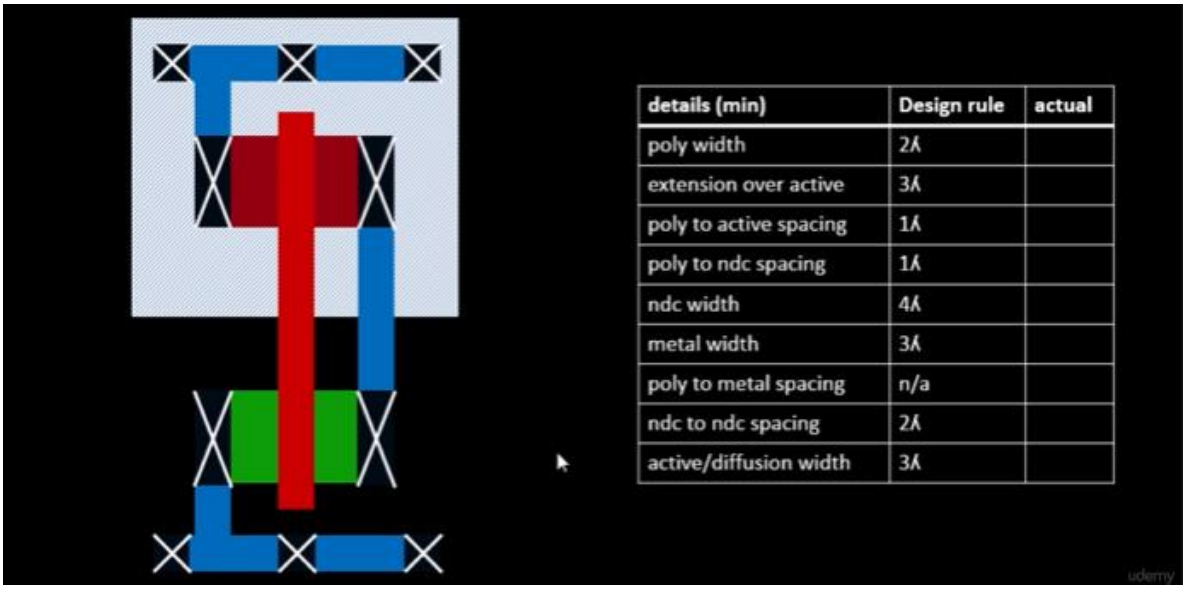


Figure 2.34: Snippet of chip showing Lambda rules.

2.11 SUMMARY

This is the brief overview of the physical design flow in which, we have the flow planning, then we have the placement, then we have to optimize placement of the cells, then we have static timing analysis where we have the estimated clock delays and the estimated data path delays, then we do the clock to synthesis where we fix the clock wires and the clock path. And after that we again do static timing analysis with the real clock delay and the estimated data path delay. Then we have the routing path. And then we do the STA at real clock path and the real data path.

Then we do the clock shielding so that it can be glitch free, and in the data paths which are the critical paths, also. Then we do the final DRC checks, and then layout checks, and then do the STA for parasitic extractions, as well, where we include the effect of the lumped resistances and the capacitances.

Figure 2.35 shows the chip after each and every transition in the Physical Design Flow, starting with the input of logic synthesis, followed by Floor-planning and then by placement. After Placement, Clock tree synthesis is done and lastly the Routing is done followed by the Design Rule check.

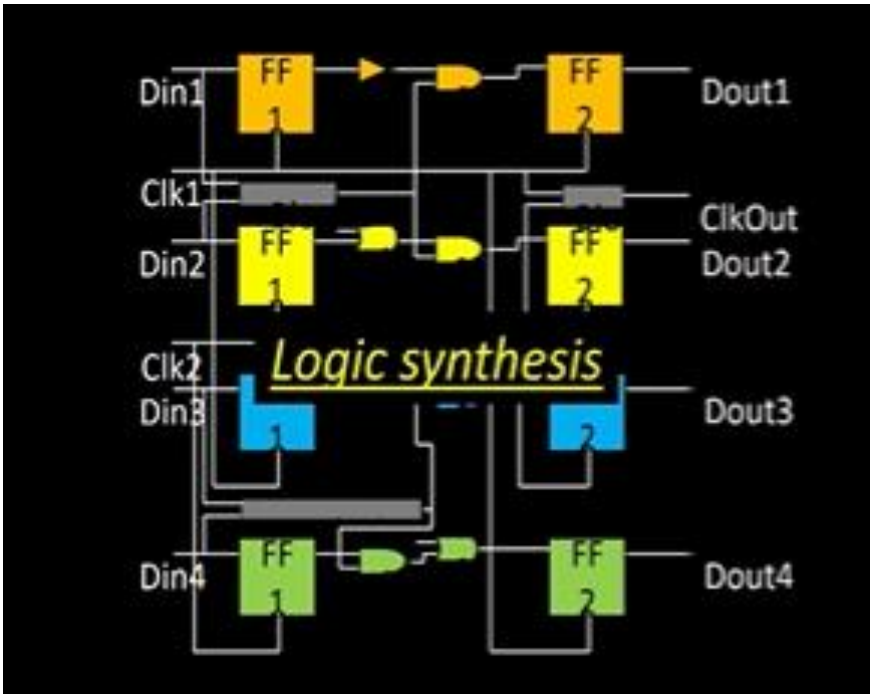


Figure 2.35(a)

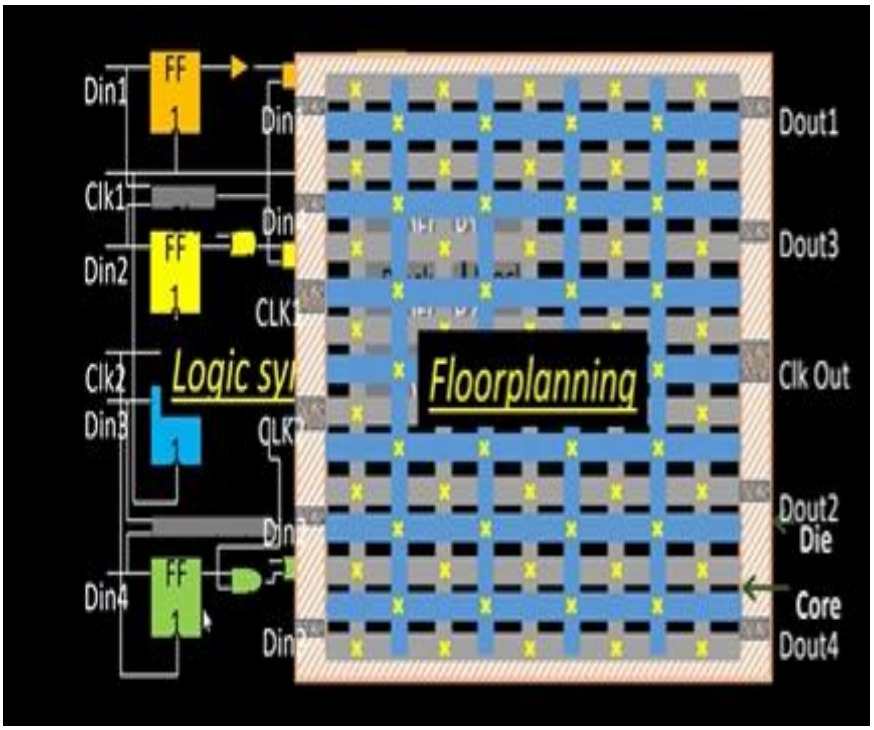


Figure 2.35(b)

Figure 2.35(c)

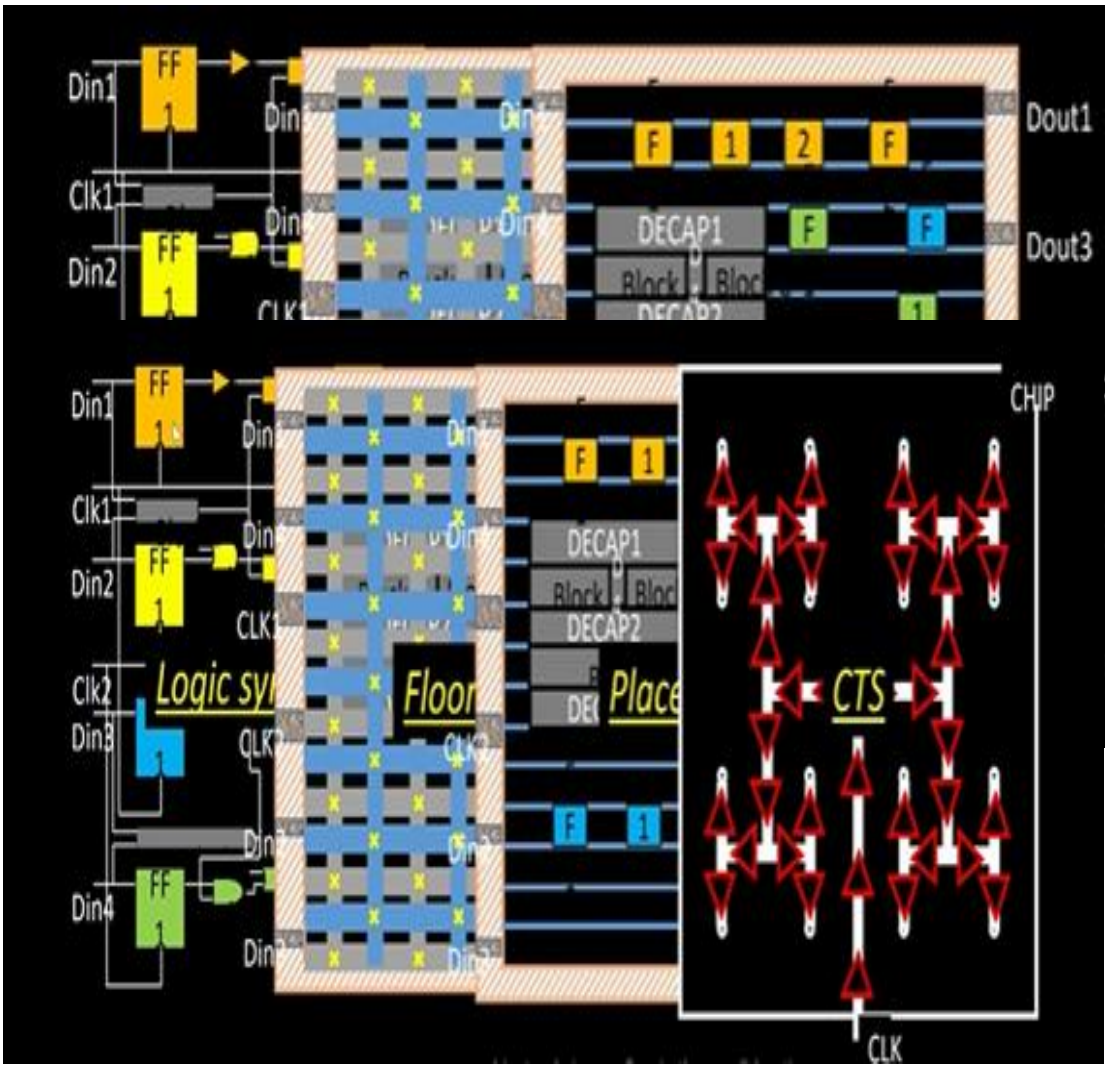


Figure 2.35(d)

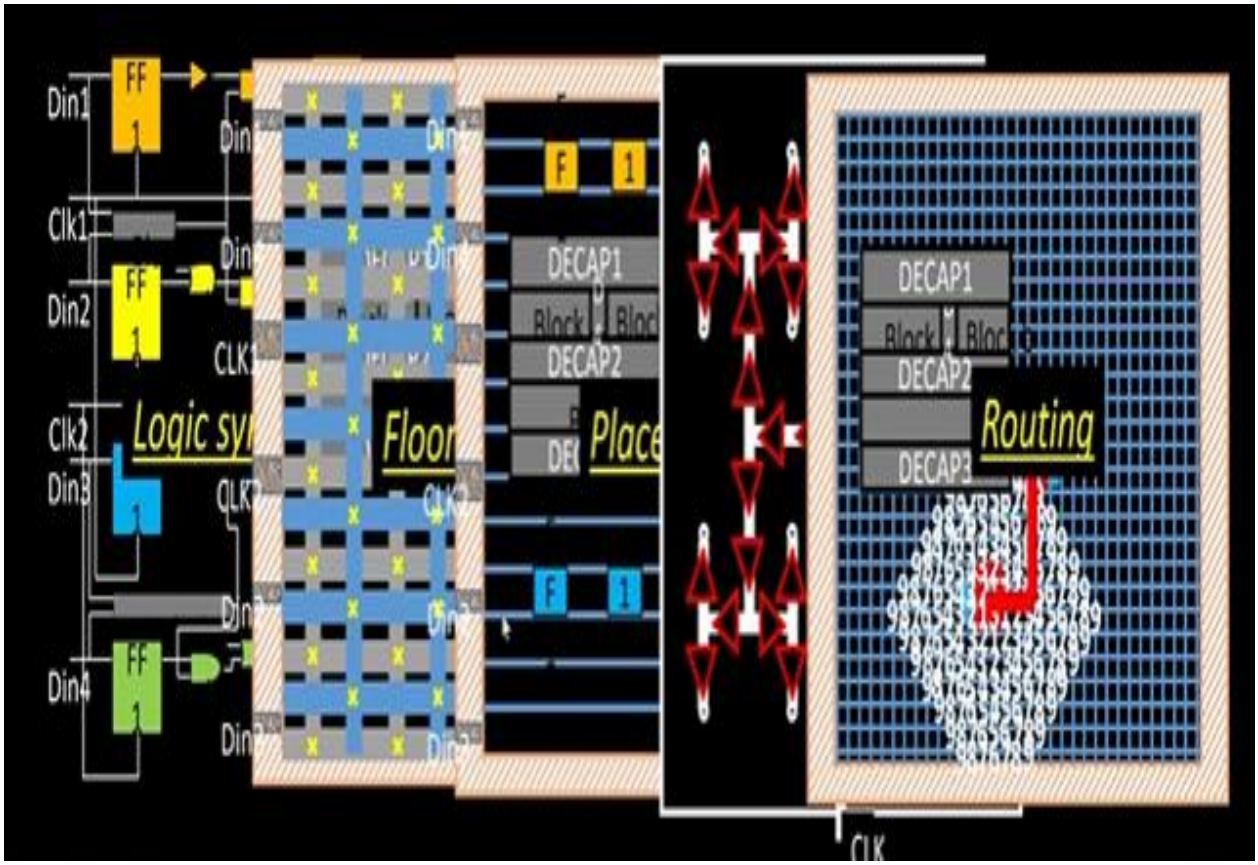


Figure 2.35(e)

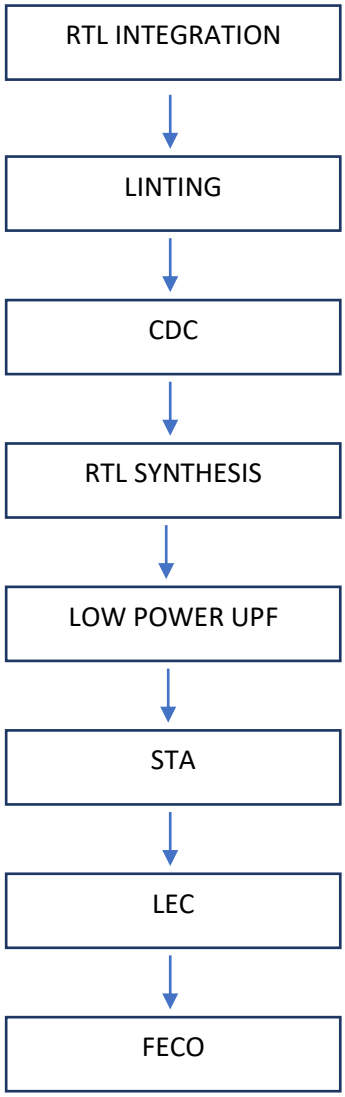
Figure 2.35: Snippet of the chip showing the steps of Physical Design Flow

Chapter 3

LINT, CLOCK DOMAIN CROSSING, UPF

3.1 LINTING

VLSI Design Flow is an integration of various steps, which are needed to be performed to meet the constraints of the design within the limited time frame as required in the Semiconductor industry and one such step is of linting. Linting is a step which comes just after the RTL Design in the flow and after the Lint is done, the flow rolls over to CDC and synthesis.



The concept is Linting is introduced in the Design so as to check the syntax errors in the RTL coding of the Design and to validate the quality of the code on the basis of good coding practices. Linting tool has the facility to raise some indication in terms of flag when there is any misuse of the coding practice and errors in the code.

The importance of the Lint process is to filter out the unwanted errors which are not because of the logic but because of the syntax which may cause a severe problem when the design is allowed to go in back end where the netlist is converted into layouts and thus in the tape-out of the chip.

If the linting process has not been done, then the synthesis tool will take a huge amount of time to verify the correct functionality of the design which again results in the miss in the deadline of the product launch by the big giants of VLSI thus Linting helps in the great reduction of time in design process.

Some of the examples where the process of linting helps in raising the flag are as follow

- Design of any unintentional latch
- Conflicts between set/reset
- Formation of combinational loops
- Presence of Non-synthesizable Blocks
- Errors due to undriven nets
- Unequal length of the operands
- Uncovered case items.
- Unused flops in the design or Extra states in FSM.
- Wrong use of Blocking/nonblocking assignments
- Occurrence of race around condition.

3.2 CLOCK DOMAIN CROSSING

3.2.1 Synchronous and Asynchronous Clocks.

The first concept that while studying the clock domain crossing is the understanding of asynchronous and synchronous clocks, as in the world of VLSI each of the SOC designed is having the clock which is moving from one domain to another. There is a multiple domain clocks, or multiple clock design process. There are very less circuits or IPS, in the SOC, which is having single

block design. The concept of synchronous and asynchronous clocks is also become important while understanding the concept of multi clock designs.

Synchronous clocks as the word implies Synchronous means in relation with each other. The two clocks are set to be synchronous with each other, if they are having the same timing of the event occurred. Synchronous clocks have a phase to phase relation with each other. When there is a defined same phase relation. The respective clocks can be generated with the help of phase lock loop or can be derived with one another. The most important property of the synchronous locks, is that they have the occurrence of either positive edge or negative edge at the same time, or the edges of the clocks in which the circuit is dependent is having the relation with each other.

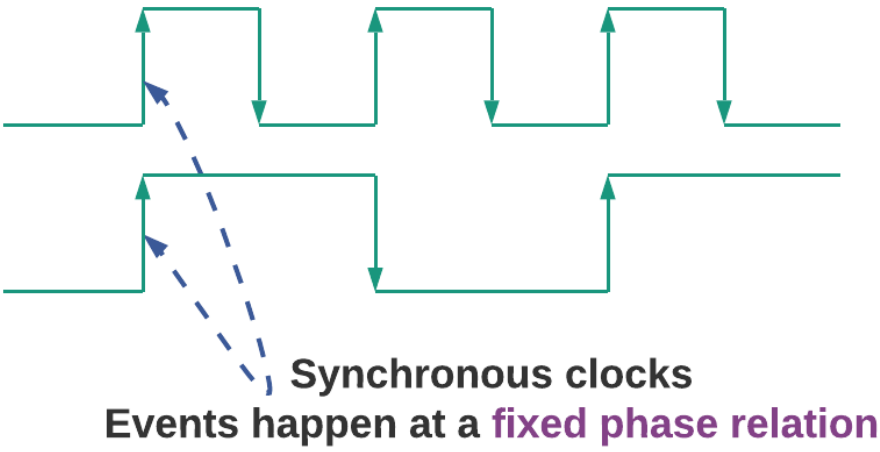


Fig 3.1: Showing Synchronous Clocks

The clocks, are said to be asynchronous when there is no a fixed relation of phases of clocks with each other. When the Launch and Capture flip flop of the same data path are getting the rising edge of the clock, then there is no relation between those two positive edges of the clock. At that time when we don't have the exact relation between the positive edges of the globe, we can say that the clocks are asynchronous.

And if there are multiple clocks present in the circuit then the static timing analysis depends on the frequency of the multiple clocks, and also on the phase relation in the clocks. When the clocks are synchronous, then a static timing analysis will become easier. The calculation of the setup and the hold constraints will be easier.

But when the clocks are asynchronous and doesn't have any relation between them, then the static timing analysis with multiple clock domains requires extra care. Thus, the concept of clock domain crossing came into existence.

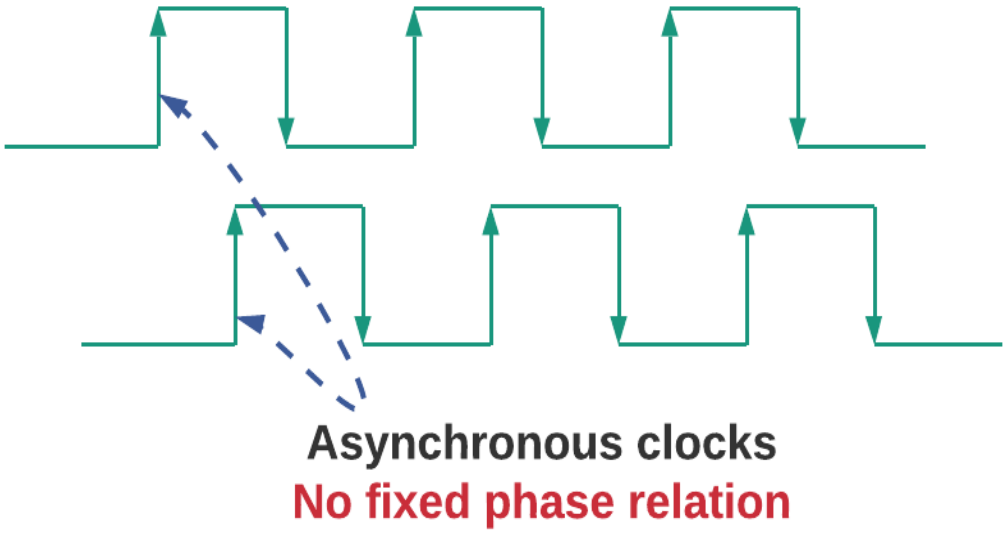


Fig 3.2: Showing Asynchronous clock

3.2.2 Metastability

When the pipeline is introduced in digital circuit it contains the back to back F/Fs in between combinational block. And when there is a data flow between the launch Flop, and the capture Flop, the data will flow, then there is a particular requirement of the capture Flop which deals with the setup requirements. And there are certain requirements of the launch flop, which deals with the hold time requirements.

There is a situation in digital circuits, when the data is changing with the effect of process, variation and temperature. And there is a failure in meeting the timing of the synchronous circuits. So, the output is undetermined, whether it will take the logic high or logic low, or logic value under this condition where the output is not defined to any particular logic, and it will stay in the metastable state. This whole process in VLSI is known as metastability.

Metastability can be avoided in synchronous circuits with the help of proper timing analysis like STA and DTA, but in asynchronous circuits where the clocks are in different domains, this is unavoidable.

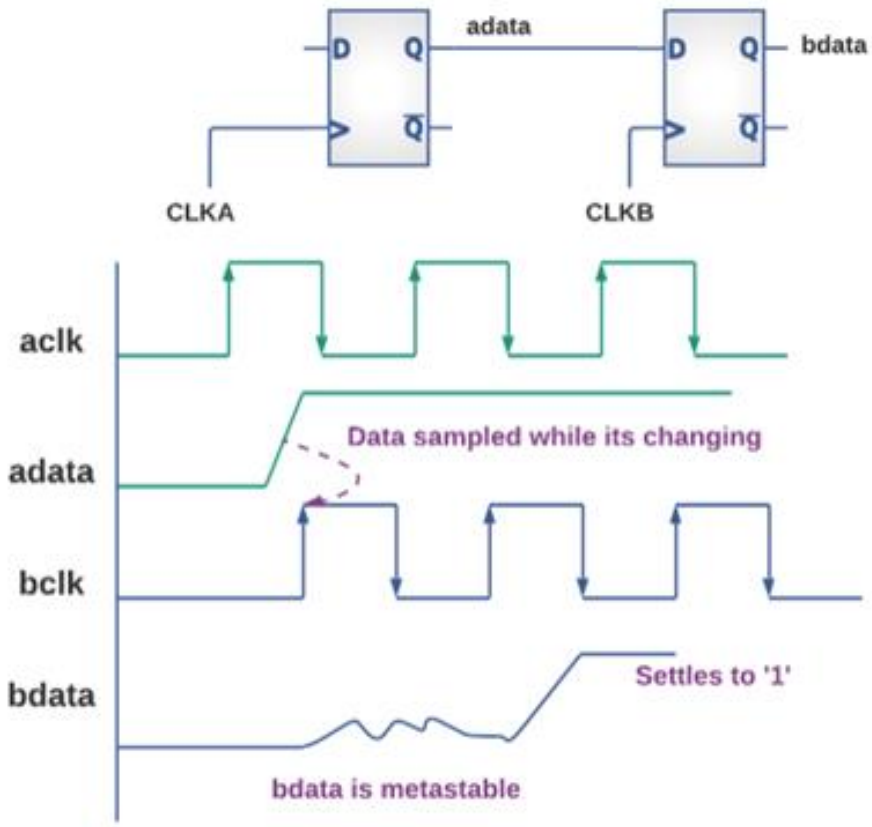


Fig 3.3(a)

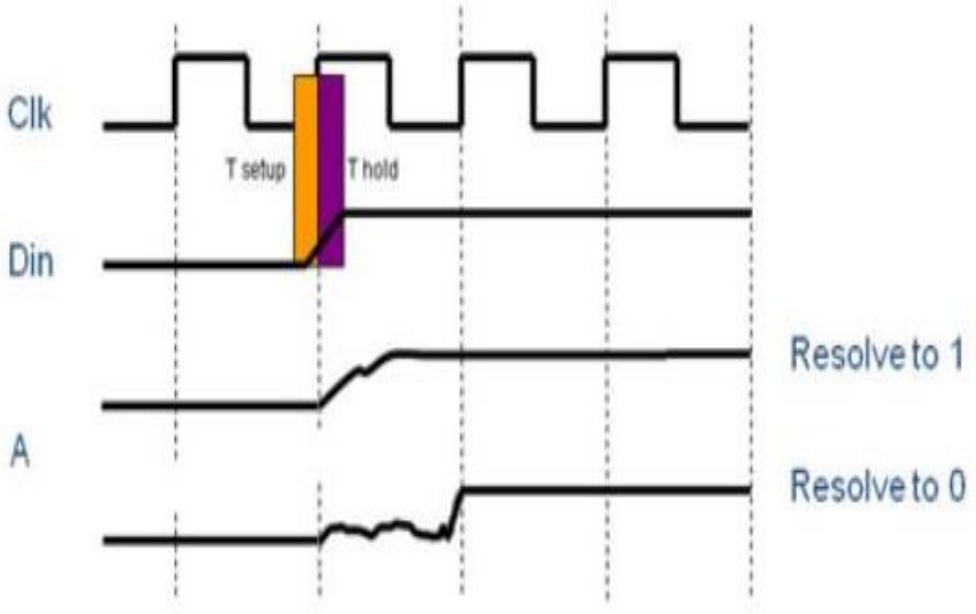


Fig 3.3 (b)

Fig 3.3: Showing the concept of Metastability.

3.2.3 Synchronizers

The basic element in the digital circuit which is used to minimise the effect of Metastability is the Synchronizer. There are various types of synchronisers which are used in the asynchronous digital systems depending on the requirement of the amount of metastability and also about the connection between the ports where there is the cause of the metastability. The most common synchronizer is the cascade connection of 2 F/Fs which is used to minimise the setup and hold constraint violation in the data path of the SOC.

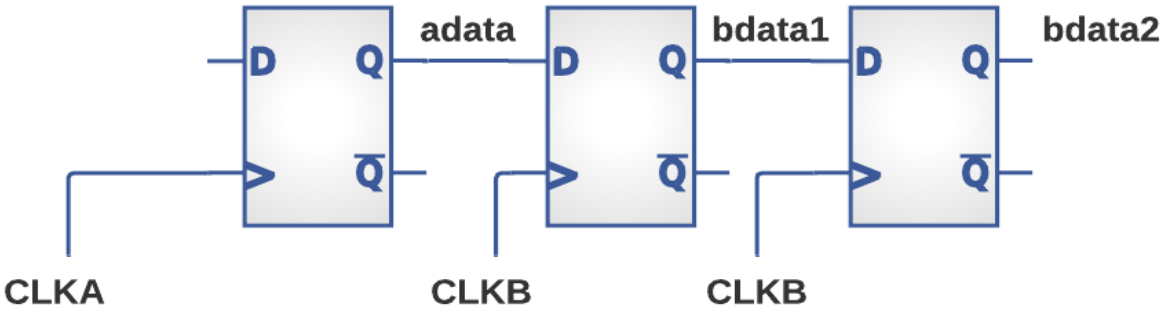


Fig 3.4(a)

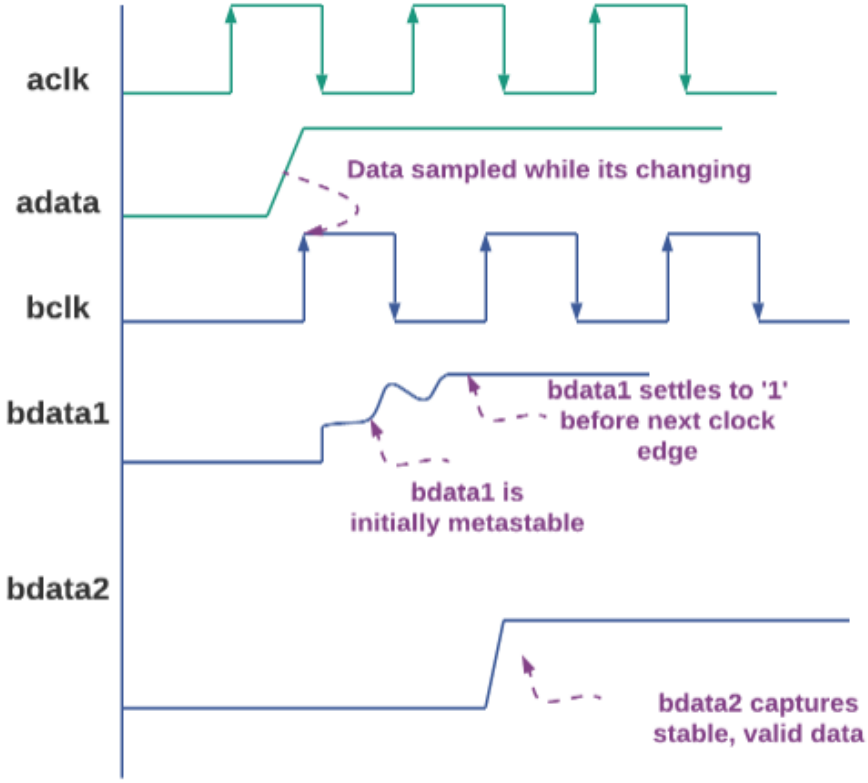


Fig 3.4 (B)

Fig 3.4: Showing Advantages of 2 F/F Synchronizers

According to this type of technique, the stability is achieved when more time is given from the output to get stable. The 2 cascaded F/Fs helps in time borrowing for one more clock cycle, for the output data to get stable.

3.2.3 Mean Time Before Failure (MTBF)

While using the technique of 2 cascaded F/Fs, there is a possibility that the output may not get stable even using 2 F/Fs, so there is a need to calculate the time before which the output data can be stabilize. This parameter is known as MTBF.

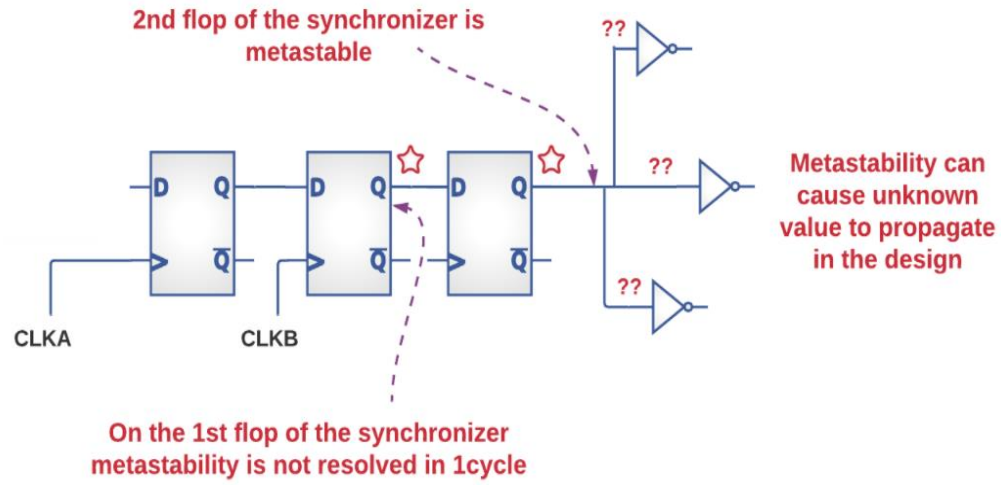
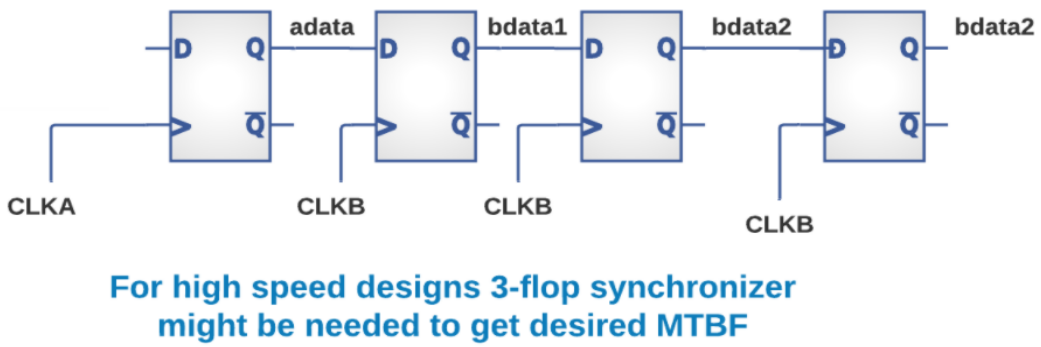


Fig 3.5 Showing Data Copying Problem due to Metastability.

There are some Digital Designs where MTBF is too short for 2 F/Fs, so the 3 F/Fs design is implemented for high speed digital circuits. MTBF should ideally be as high as possible for the design to be free from Metastability, so that the



failure frequency is low and the signal integrity is maintained.

Fig 3.6: Showing 3 F/F Synchronizer

3.2.4 Effects of Metastability

For understanding the effect of Metastability, considering the following figure which contains 2 F/Fs synchronizer:

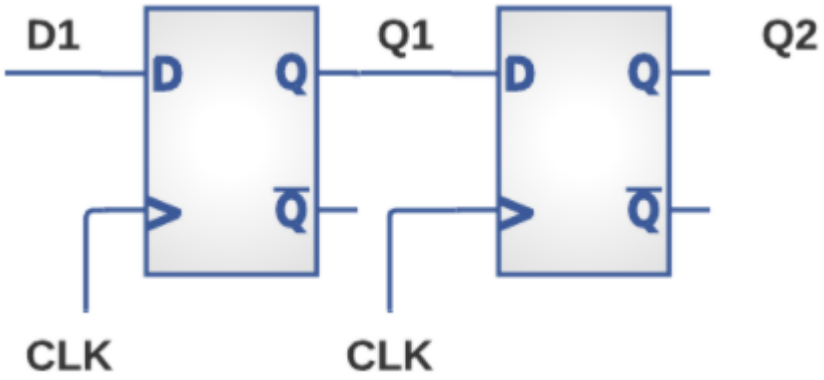


Fig 3.7 Showing 2 F/F Synchronizer for Explaining Effect of Metastability.

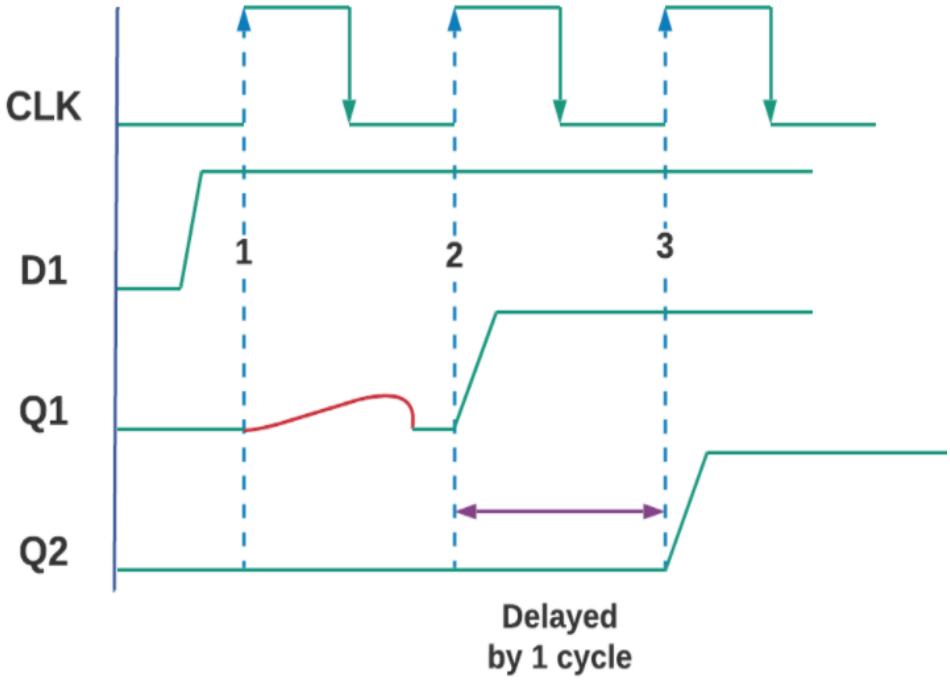


Fig 3.8: Effect of Metastability on Pulse delay.

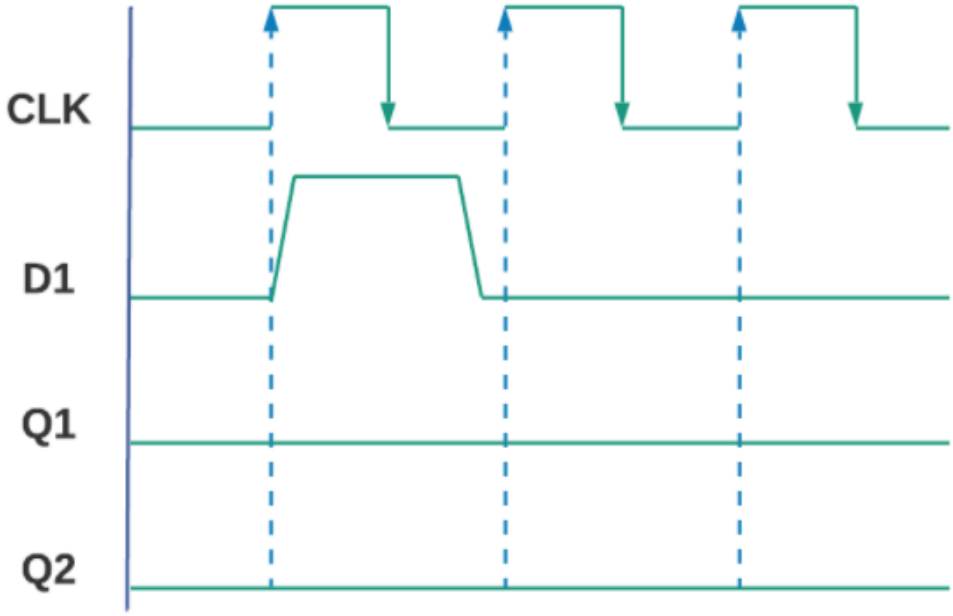


Fig 3.9: Effect of Metastability on Pulse Missed.

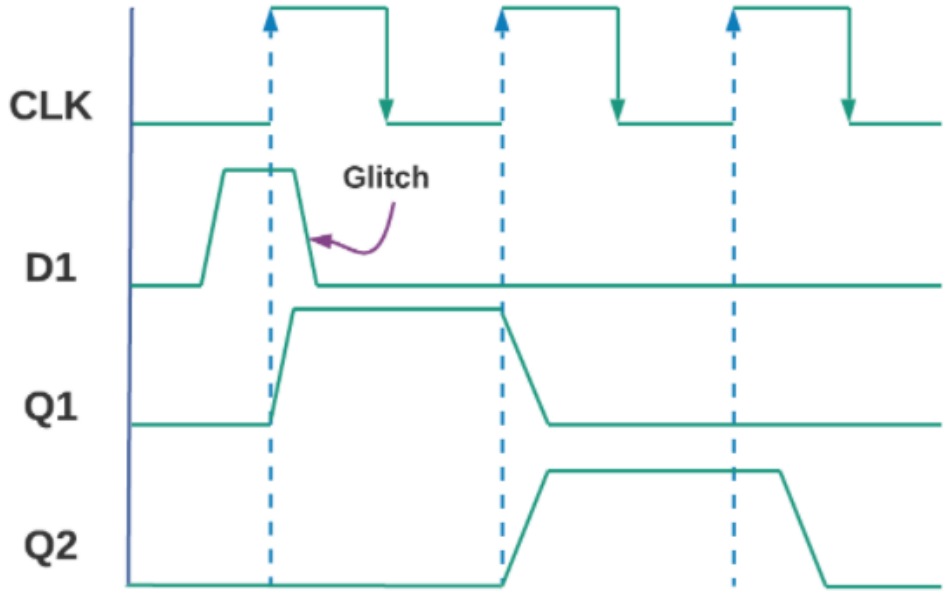
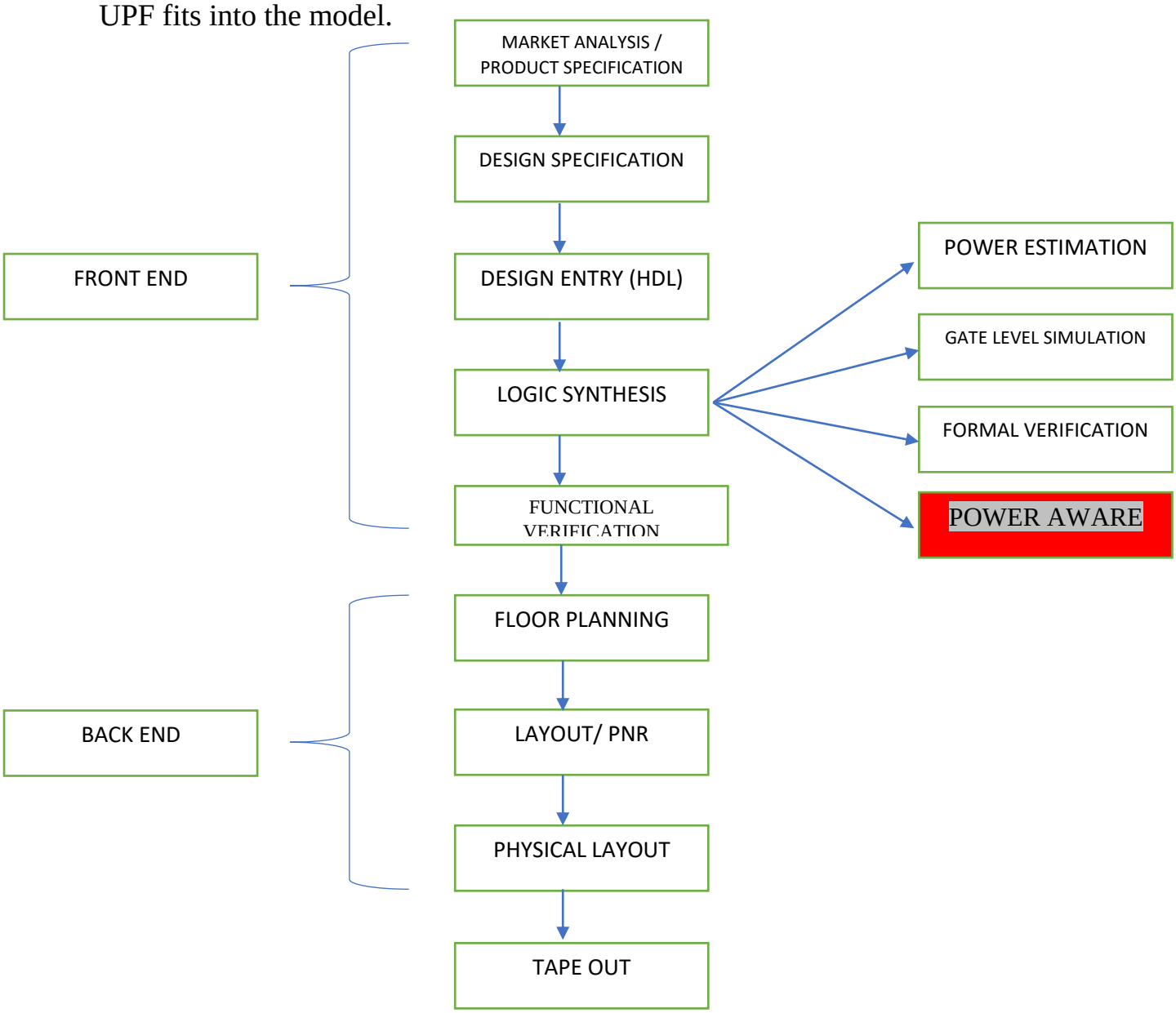


Fig 3.10: Effect of Metastability on Glitch Captured.

3.3 UPF (Unified Power Format)

Unified Power Format is one of the crucial aspects in the world of VLSI, which leads to the enhancement in the design and verification strategy of the semiconductor industry. The following shows the VLSI Design flow and where UPF fits into the model.

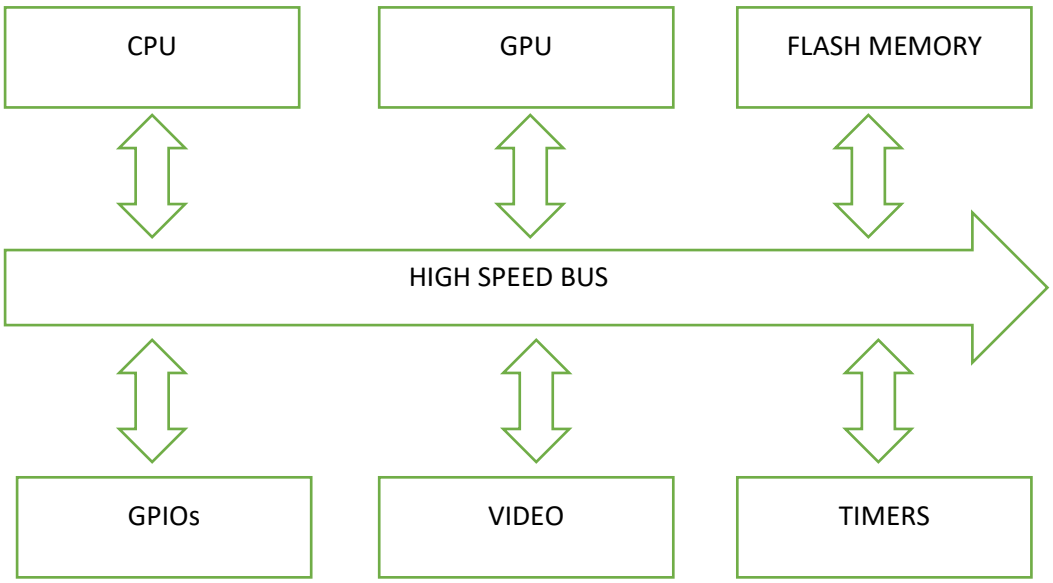


As discussed earlier, the VLSI Design Flow includes many phases and one of the topics to be focused upon is the need of UPF which typically comes both in the design part as well as in the verification part of the cycle.

According to the Moore's Law, the complexity of the design or SoC is increasing every 18 months, so as the Power concern because the Power dissipation in the circuit is increasing exponentially. The size of the transistors is reducing resulting the increase of the parasitic capacitance and Dynamic power dissipation is directly proportional to the Capacitance and frequency and also on the voltage supply.

$$\mathbf{Pd = C*Vdd^2*F_{switching}}$$

The Dynamic power dissipation is increasing so there must be a way to control the unwanted the power wasted in the system. One considerable example is given below.



Here the assumption is that the CPU is working on high frequency like on 3 GHz (1V-2V) and the video player on low frequency like on 2 GHz (0.5V-1.5V). During the time of interaction between CPU and Video player, the operating voltage would be 1V. Since video player can be operated on the given frequency at low voltage so the extra voltage supply results in the extra Power dissipation. Thus, there is a need of power aware concepts in the design architecture.

RTL Simulation v/s Power Aware Verification Using UPF

There is a need of Power aware verification as many things can't be taken into consideration while written the code in Verilog. There are certain predefined assumptions which are made by the simulation tool but at the time of synthesis creates a major cause of error in the design of chip or SOC. The example in this context is given above.

Chapter 4

RESULTS AND DISCUSSIONS

```
module Example_name(d,m,clk,reset,q,qb);
input d,clk,reset;
output q;
always @ (posedge clk)
    begin
    if(reset == 0)
            assign q=d;
        end
endmodule

W18: Missing Else Statements . Infers Latch.
```

Fig 4.1: Spyglass error W18

In the code (Figure 4.1), which is having the module of the D flip flop when the clock edge is positive, then after this synchronous reset becomes 1'b0 it will assign the value, data to the output node. But when the reset is 1'b1. There is no value of the output So, it must retain its value. That means the latch is inferred by the synthesis software. And this is an unknown latch, which is been created and the extra hardware is needed to implement this code, so this is the error, shown by the spyglass lint tool.

```
module Example_name(d,m,clk,reset,q,qb);
input d,clk,reset;
output q;
always @ (posedge clk)
    begin
    if(reset == 1)
            assign q=1'b0101;
    else
            assign q=d;
        end
endmodule

W19: Truncation of Extra Bits.
```

Fig 4.2: Spyglass error W19

The value of Q is assigned as the four bit binary 0101, which means value five is assigned to the output Q, when the reset is one, also the reset is this syncronous reset. when the reset is zero, then the value of input is assigned to Q, D is mentioned as a one bit value. It is not the vector.And in the above case when the set is one, output will getting the value as four bits, but it is declared as a one bit. Output is not declared as the vector. So the rest of the bits will be truncated.And this will create a difficulty in the code coverage and in the design for testability part. (Figure 4.2)

```
module mux_exmaple_name(a,b,c,d,sel,y);
input a,b,c,d,sel[1:0];
output y;
always @ (sel,a,b,c,d)
case(sel or a or b or c or d)
2'b00:y=a;
2'b01:y=b;
2'b10:y=c;
endmodule

W69: All Possible Case & Default statements should be Preser
```

Fig 4.3: Spyglass error W69

In the above code (Figure 4.3), the design function is to implement the 4*1 mux, where there is a two bit select line. Y0 and Y1, and the four inputs are A, B, C and D. Here the behavioral level of coding is done With the help of cases statements where output is assigned according to these selected combination

that is output is A when {Y1,Y0} is 00 output is B when this combination is 01 and output is connected to C when this combination is 01. But here there is no information about the Select line as 2'B11. So this synthesis tool will automatically generate a Latch, when the condition of select line is one, one.To avoid the extra Latch which is created by this synthesis tool, either all the possible cases should be written in the case statement, or the default statement should be present, which allocates a particular input to this output, so that each and every value of the input, select lines, The output has a unique value, it will

```
module lin_ex1(a,b,c);
input [2:0]a, b;
output {2:0}c;
assign c= a[1:0] & b;
endmodule

W111: Not all Elements in an array are read.
```

not infered some latch.

Fig 4.4: Spyglass error W111

The above code (Figure 4.4) shows the illegal you use of an assign statement where on the left-hand side, argument, A is not fully used, which will create the extra hardware loss in the synthesizable process. As A is getting the three bits in

```
module lin_ex1(a,b,c);
input [2:0]a, b;
output [2:0]c;
wire [2:0} net;
assign c= a[1:0] && net;
endmodule
```



```
W123: Identifies Signals and variables are read/used but not set
```

the declaration part, but in the actual code only the two bits have been used so the extra 3th bit which will create some hardware limitations in the synthesis process and will create the problem in the lint, and is depicted as the error w111 in the Spyglass.

Fig 4.5: Spyglass error W123

In the above code (Figure 4.5), using the assign statement, which is not declared above, the net the statement is declared as the wire, and it's not the input or the output port which can be initialized during the simulation process. So, while assigning the value of C, tool should get the value of net as the intermediate wire, and if the value is not getting right, then it will create the signal and the variables issues values by performing the lint process in the Spyglass.

```
module exmaple_name(d,clk,reset,q);
input d,clk,reset;
output q;

always @ (posedge clk or negedge reset)
    begin
    if(reset == 1)  /*== Comapre <=NonBlocking =Blocking */
        {
            assign q<=1'b0;
        }
    else
        {
            assign q=d;

        }
    end
endmodule

W336: Blocking Statements should not be used in Sequential Block-
```

Fig 4.6: Spyglass error W336

There are two types of continuous assignments in Verilog. The first is Blocking type procedural assignment and the second is the non-blocking type, each having its own importance and each will result in a different type of hardware after the synthesis process of the VLSI design. Same variable cannot be assigned to both non-blocking and blocking type because of the implementation order of these statements in Verilog. There will be the conflict type of occurrence of these statements. This will result in the error in the process, the general trend is to implement sequential blocks with the help of non-blocking

statements. When the sequential block is implemented using blocking the statements the error W336 occurred in spyglass lint. (Figure 4.6)

```
module Example_name(d,m,clk,reset,q,qb);
input d,clk,reset;
output q;
always @ (posedge clk)
    begin
    if(reset == 1)  /*== Comapre <=NonBlocking =Blocking */
            assign q<=1'b0;
    else
            assign q=d;
        end

always @ (negedge clk)
    begin
    if(reset == 1)
            assign qb=1'b0;
    else
            assign q=m;
        end
endmodule

W391: Usage of Both Positive & Negative Edge Clocks.
```

Fig 4.7: Spyglass error W391

The above case (Figure 4.7) represents the use of both positive edge and negative edge of the clock, which will create problem in the Timing Analyses of the block and also in the Design for Testability part. The setup and the hold requirements will be difficult while measuring both the end points of the clock.

```
module DFF_exmaple_name(d,m,clk,reset,q,qb);
input d,clk,reset;
output q;
always @ (posedge clk or posedge reset)
    begin
    if(reset == 1)  /*== Comapre <=NonBlocking =Blocking */
            assign q<=1'b0;
    else
            assign q=d;
        end

always @ (posedge clk or negedge reset)
    begin
    if(reset == 1)
            assign qb=1'b0;
    else
            assign q=m;
        end
endmodule

W392: Usage of Both Positive & Negative Edge Triggered Reset/
```

Fig 4.8: Spyglass error W392

The above case (Figure 4.8) represents the use of both positive edge and negative edge of the reset, which will create problem in the Timing Analysis of the block and also in the Design for Testability part. The recovery and the removal requirements will be difficult while measuring both the end points of

```
module exmaple_name(d,q);
input d;
output q;
assign q<=d;
endmodule

W414: NonBlocking Statements should not be used in Combinational Blocks
```

the reset.

Fig 4.9: Spyglass error W414

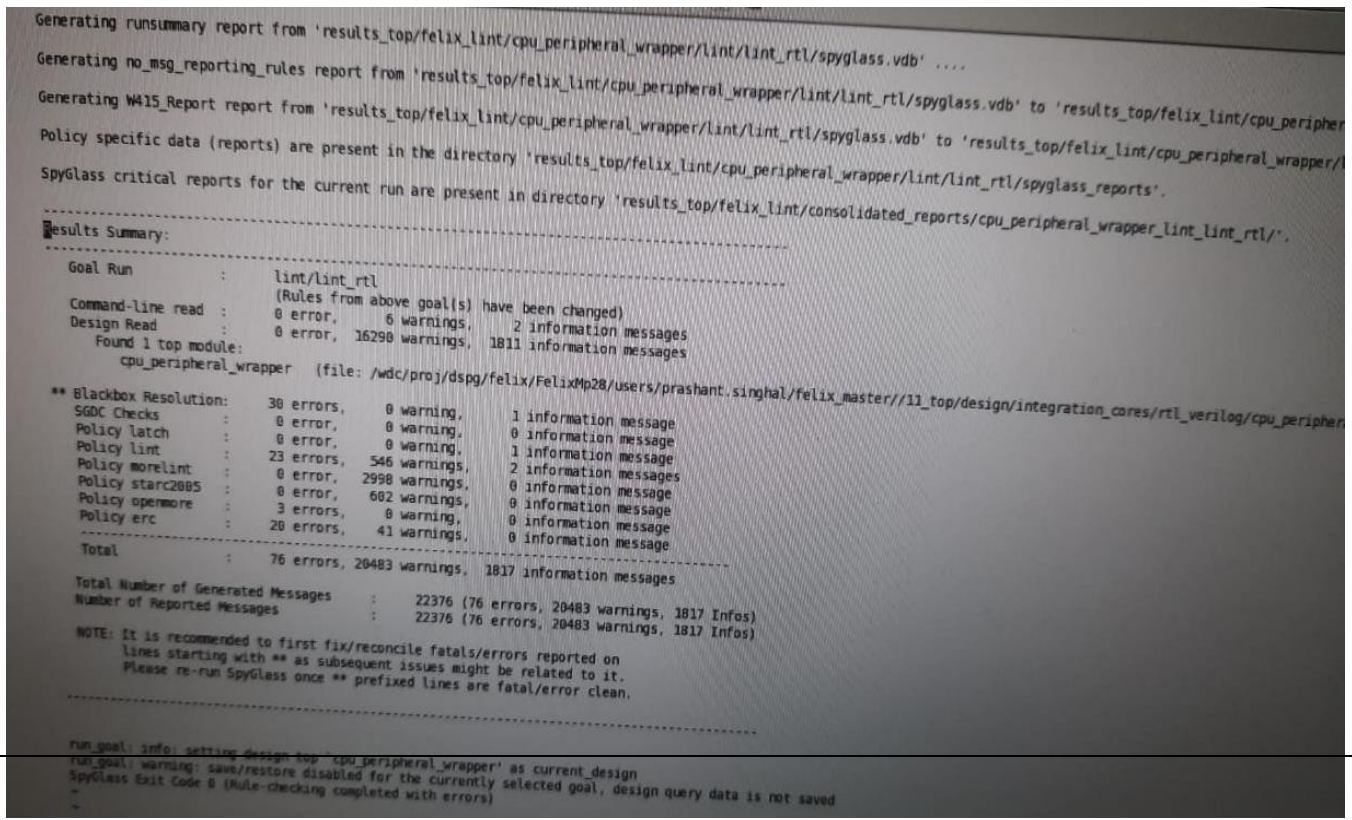
The above case (Figure 4.9) shows the use of Non- blocking statement while designing the Combinational Block, which will create problem in the designing part of the SOC, as there will be a trade-off between the synthesis of the hardware which the different coding style is done for the desired block. (Figure

```
module Example_name(d,m,clk,reset,q,qb);
input d,clk,reset;
output q;
always @ (posedge clk or posedge reset)
    begin
    if(reset == 1)  /*== Comapre <=NonBlocking =Blocking */
            assign q<=1'b0;
    else
            assign q=d;
        end

always @ (posedge clk)
    begin
    if(reset == 1)
            assign qb=1'b0;
    else
            assign q=m;
        end
endmodule

W448: Usage of Both Synchronous & Asynchronous Reset/Set
```

4.10)

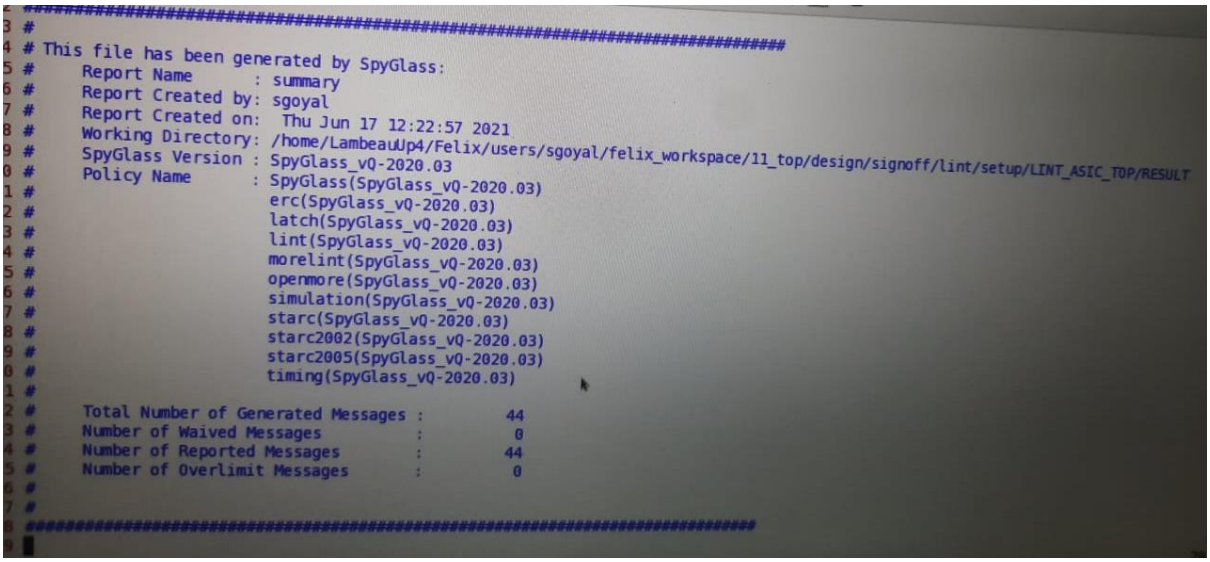
Fig 4.10: Spyglass error W448

The above case (Figure 4.10) represents the use of both Synchronous and asynchronous reset, which will create problem in the Timing Analysis of the block and also in the Design for Testability part. The recovery and the removal



requirements will be difficult while measuring both the end points of the Synchronous and asynchronous reset.

Fig 4.11 Snippet of the report showing the Error information in Spyglass.



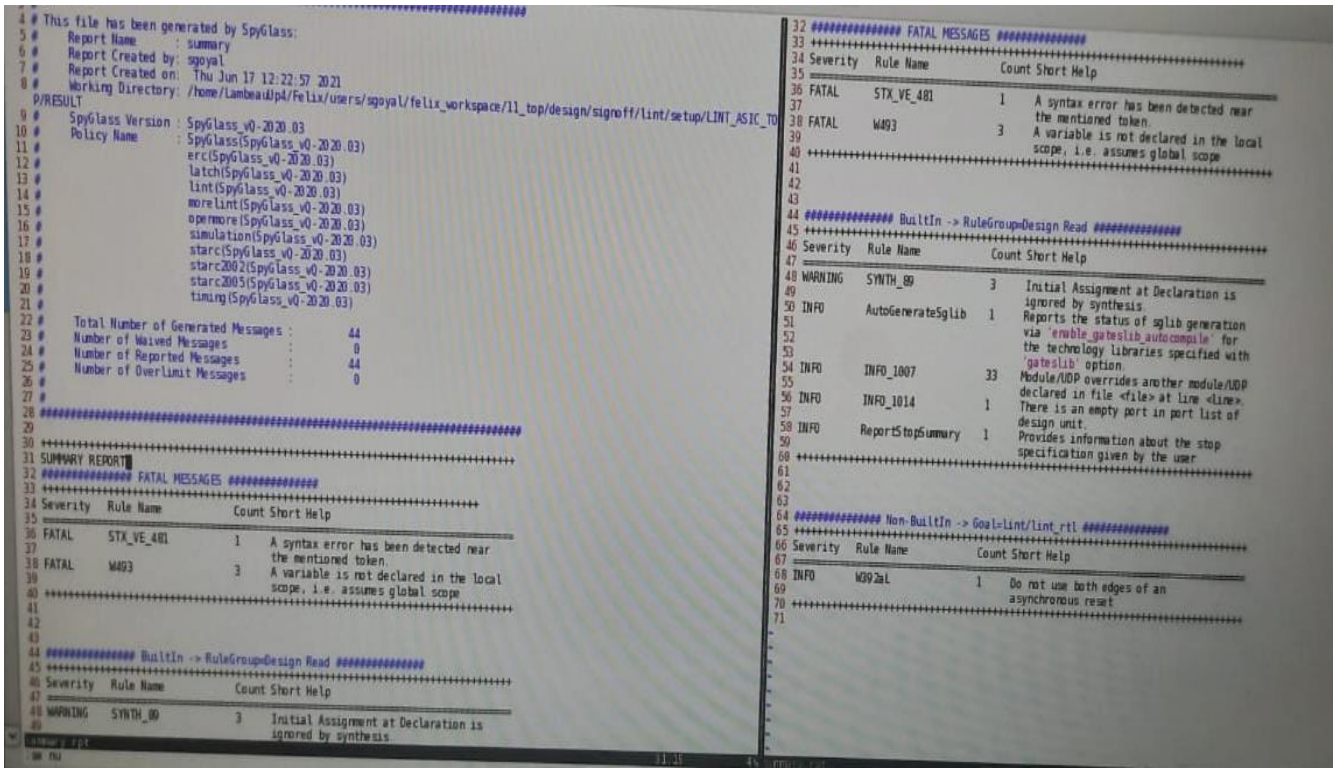


Fig 4.12 Snippet of the report showing the Report generation in Spyglass.

Fig 4.13: Snippet of the report showing the error reduction in Spyglass.

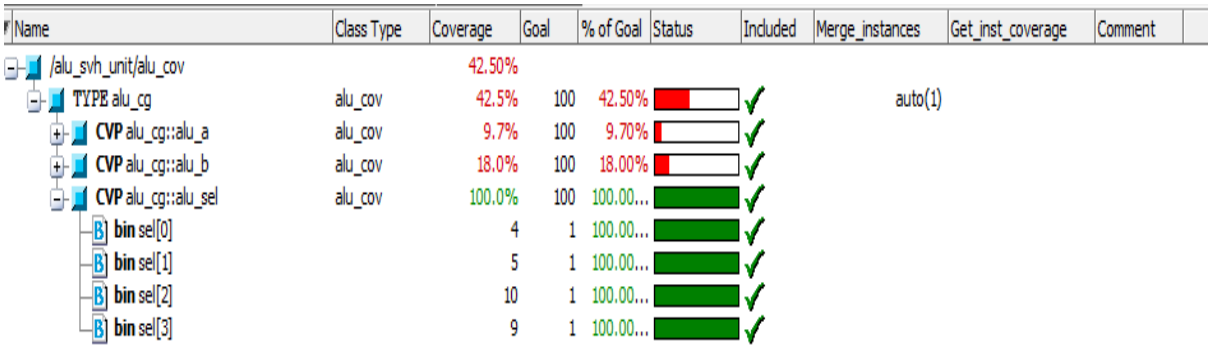
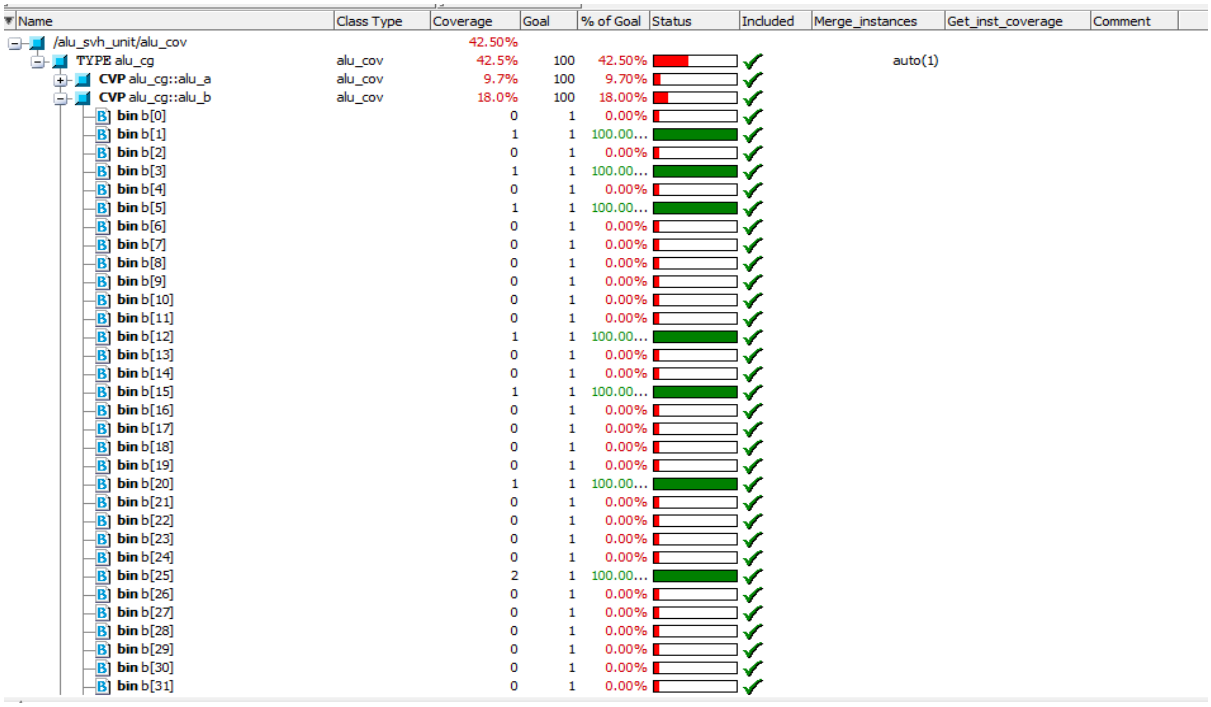
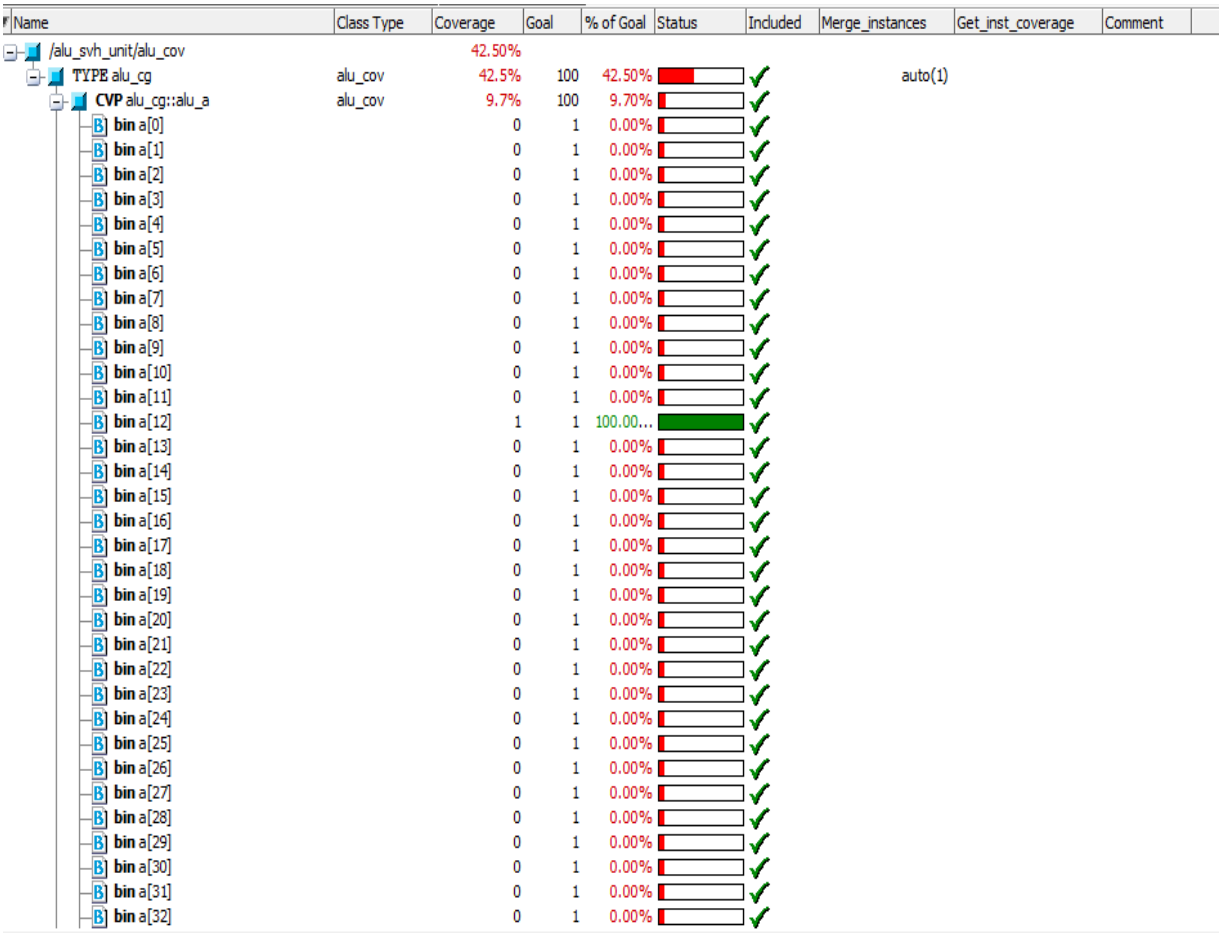


Fig 4.14: Snippet of the report showing of the functional Coverage in QuastaSim with 25 test cases.



Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/alu_svh_unit/alu_cov		42.50%							
TYPE alu_cg	alu_cov	42.5%	100	42.50%			auto(1)		
CVP alu_cg::alu_a	alu_cov	9.7%	100	9.70%					
CVP alu_cg::alu_b	alu_cov	18.0%	100	18.00%					
bin b[0]		0	1	0.00%					
bin b[1]		1	1	100.00...					
bin b[2]		0	1	0.00%					
bin b[3]		1	1	100.00...					
bin b[4]		0	1	0.00%					
bin b[5]		1	1	100.00...					
bin b[6]		0	1	0.00%					
bin b[7]		0	1	0.00%					
bin b[8]		0	1	0.00%					
bin b[9]		0	1	0.00%					
bin b[10]		0	1	0.00%					
bin b[11]		0	1	0.00%					
bin b[12]		1	1	100.00...					
bin b[13]		0	1	0.00%					
bin b[14]		0	1	0.00%					
bin b[15]		1	1	100.00...					
bin b[16]		0	1	0.00%					
bin b[17]		0	1	0.00%					
bin b[18]		0	1	0.00%					
bin b[19]		0	1	0.00%					
bin b[20]		1	1	100.00...					
bin b[21]		0	1	0.00%					
bin b[22]		0	1	0.00%					
bin b[23]		0	1	0.00%					
bin b[24]		0	1	0.00%					
bin b[25]		2	1	100.00...					
bin b[26]		0	1	0.00%					
bin b[27]		0	1	0.00%					
bin b[28]		0	1	0.00%					
bin b[29]		0	1	0.00%					
bin b[30]		0	1	0.00%					
bin b[31]		0	1	0.00%					



Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/alu_svh_unit/alu_cov		42.50%							
TYPE alu_cg	alu_cov	42.5%	100	42.50%			auto(1)		
CVP alu_cg::alu_a	alu_cov	9.7%	100	9.70%					
bin a[0]		0	1	0.00%					
bin a[1]		0	1	0.00%					
bin a[2]		0	1	0.00%					
bin a[3]		0	1	0.00%					
bin a[4]		0	1	0.00%					
bin a[5]		0	1	0.00%					
bin a[6]		0	1	0.00%					
bin a[7]		0	1	0.00%					
bin a[8]		0	1	0.00%					
bin a[9]		0	1	0.00%					
bin a[10]		0	1	0.00%					
bin a[11]		0	1	0.00%					
bin a[12]		1	1	100.00...					
bin a[13]		0	1	0.00%					
bin a[14]		0	1	0.00%					
bin a[15]		0	1	0.00%					
bin a[16]		0	1	0.00%					
bin a[17]		0	1	0.00%					
bin a[18]		0	1	0.00%					
bin a[19]		0	1	0.00%					
bin a[20]		0	1	0.00%					
bin a[21]		0	1	0.00%					
bin a[22]		0	1	0.00%					
bin a[23]		0	1	0.00%					
bin a[24]		0	1	0.00%					
bin a[25]		0	1	0.00%					
bin a[26]		0	1	0.00%					
bin a[27]		0	1	0.00%					
bin a[28]		0	1	0.00%					
bin a[29]		0	1	0.00%					
bin a[30]		0	1	0.00%					
bin a[31]		0	1	0.00%					
bin a[32]		0	1	0.00%					

Fig 4.15: Snippet of the report showing of the functional Coverage in QuastaSim with 25 test cases for particular inputs.

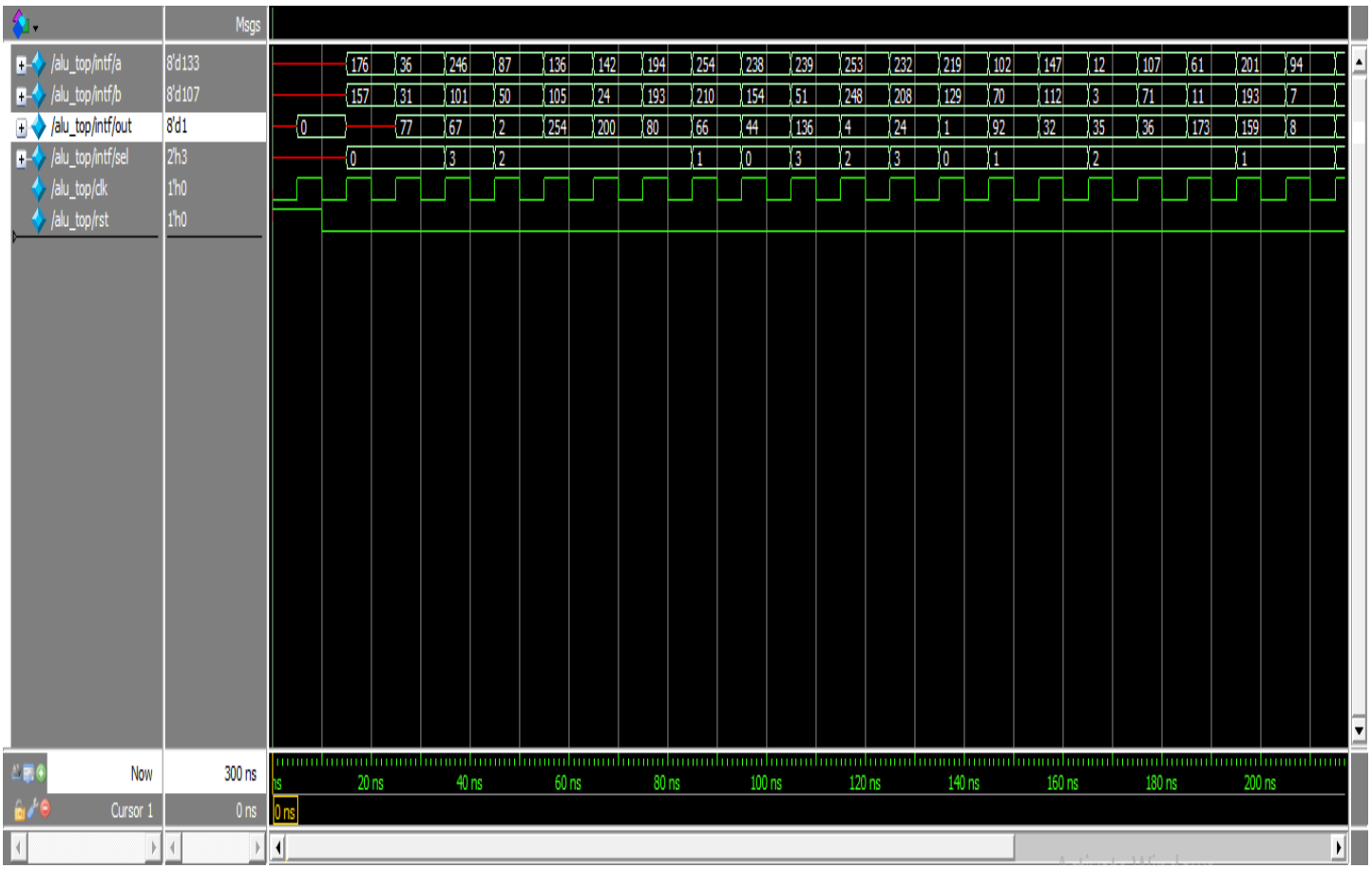
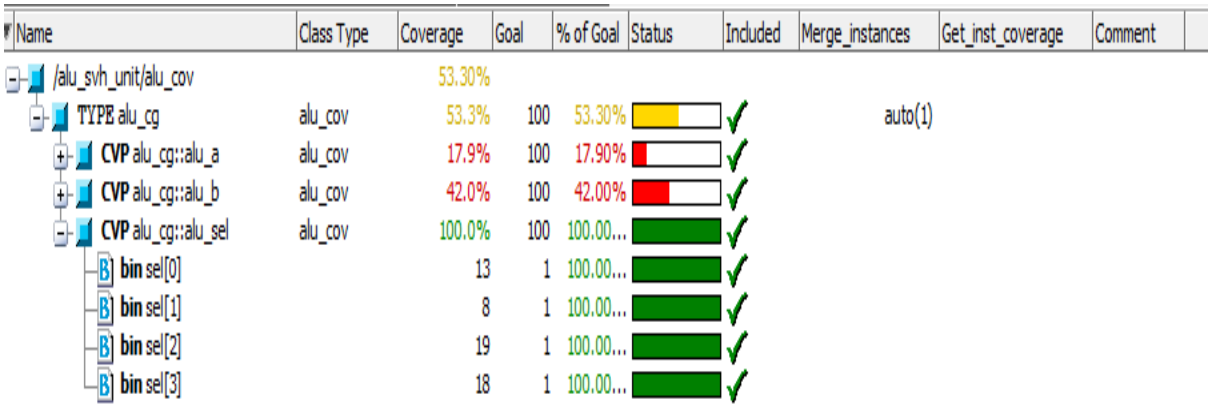
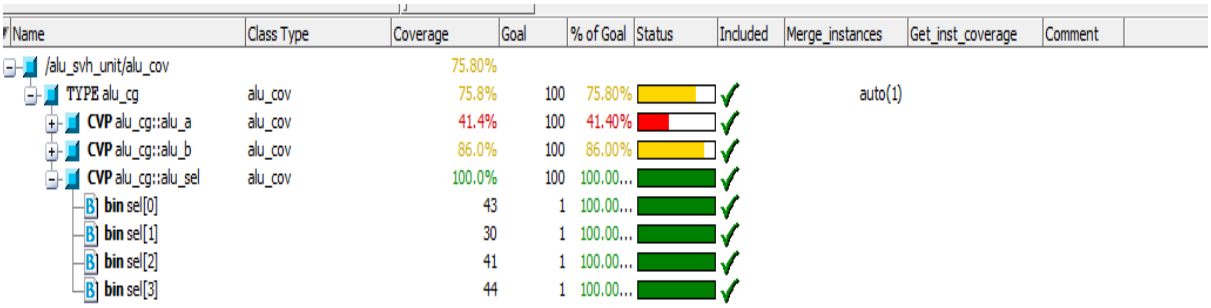


Fig 4.16: Snippet of the waveform showing of the functionality of ALU with 25 testcases.



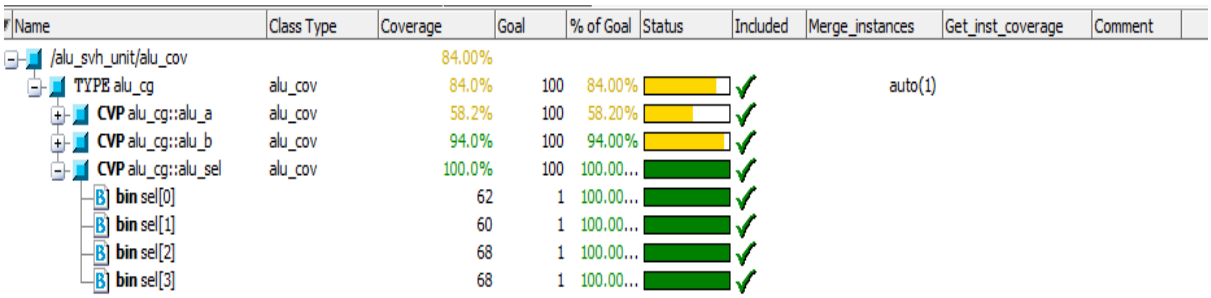
Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/alu_svh_unit/alu_cov		53.30%							
TYPE alu_cg	alu_cov	53.3%	100	53.30%			auto(1)		
CVP alu_cg::alu_a	alu_cov	17.9%	100	17.90%					
CVP alu_cg::alu_b	alu_cov	42.0%	100	42.00%					
CVP alu_cg::alu_sel	alu_cov	100.0%	100	100.00...					
bin sel[0]		13	1	100.00...					
bin sel[1]		8	1	100.00...					
bin sel[2]		19	1	100.00...					
bin sel[3]		18	1	100.00...					

Fig 4.17: Snippet of the report showing of the functional coverage of ALU with 50 testcases.



Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/alu_svh_unit/alu_cov		75.80%							
TYPE alu_cg	alu_cov	75.8%	100	75.80%			auto(1)		
CVP alu_cg::alu_a	alu_cov	41.4%	100	41.40%					
CVP alu_cg::alu_b	alu_cov	86.0%	100	86.00%					
CVP alu_cg::alu_sel	alu_cov	100.0%	100	100.00...					
bin sel[0]		43	1	100.00...					
bin sel[1]		30	1	100.00...					
bin sel[2]		41	1	100.00...					
bin sel[3]		44	1	100.00...					

Fig 4.18: Snippet of the report showing of the functional coverage of ALU with 150 testcases.



Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/alu_svh_unit/alu_cov		84.00%							
TYPE alu_cg	alu_cov	84.0%	100	84.00%			auto(1)		
CVP alu_cg::alu_a	alu_cov	58.2%	100	58.20%					
CVP alu_cg::alu_b	alu_cov	94.0%	100	94.00%					
CVP alu_cg::alu_sel	alu_cov	100.0%	100	100.00...					
bin sel[0]		62	1	100.00...					
bin sel[1]		60	1	100.00...					
bin sel[2]		68	1	100.00...					
bin sel[3]		68	1	100.00...					

Fig 4.19: Snippet of the report showing of the functional coverage of ALU with 250 testcases.

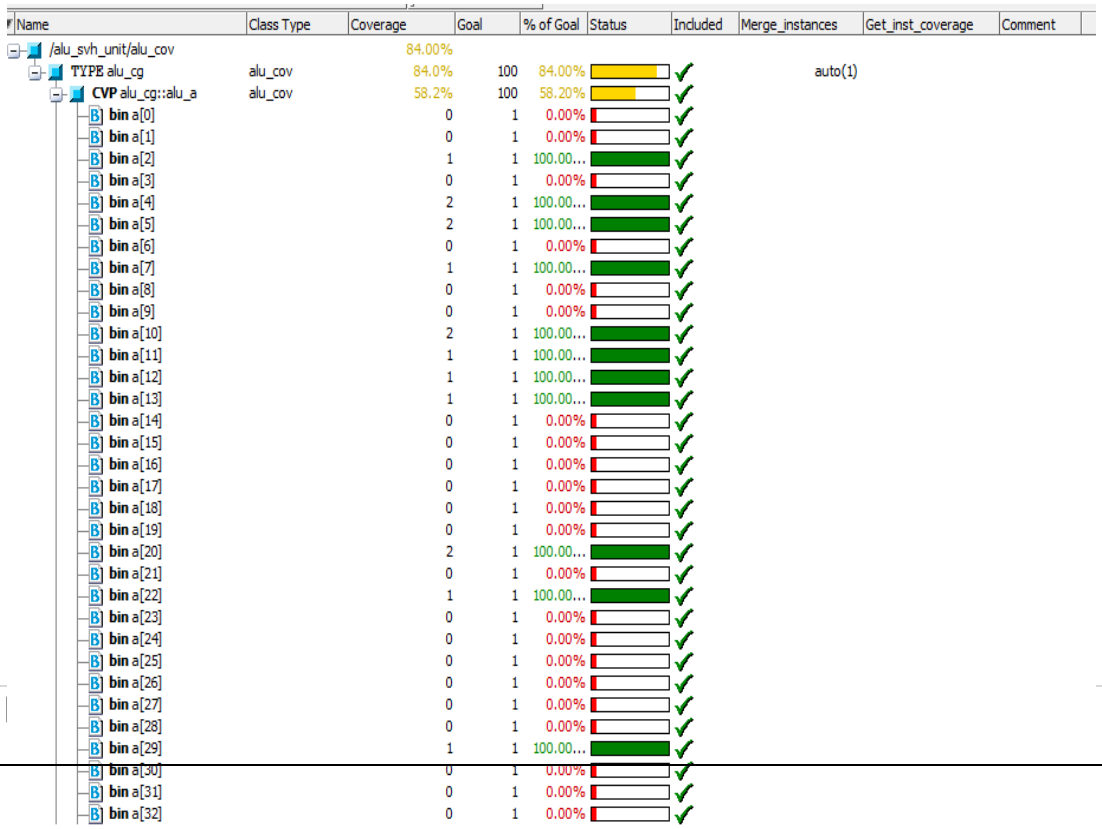


Fig 4.20: Snippet of the report showing of the functional Coverage in QuastaSim with 250 test cases for particular inputs.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

With the increase in the growth of VLSI, the products are manufacturing in great extent. This leads to the increase in demand of electronic products like Memory components, Processors, and Various other functional blocks, which are needed to be designed and verified properly so that it results in good yield. There is a need to proper flow from the design level specification till the product been manufactured and tested. The early part of the work deals with this ASIC Flow.

The Second half denotes to the Functional checks which have to be done, so that the desired functionality which is written in HDL code can be synthesized properly with the help of any EDA tool. The process of Lint, CDC and UPF have been studied thoroughly and the concepts which will create problems in the respective steps have been shown in figures and results.

The Last section of the work depicts the Design and Verification of ALU, so that the compete flow can be understood with one practical example. There the code functional coverage is maximum reached to 84%. For the complete functional coverage, manual test cases must have been inserted to cover all possible cases.

After understanding the topics which have been covered in this thesis, one can easily get the idea about how the chips are going to be made in any semiconductor industry and the correct procedure for designing the required functionality. The Designers working in different industries will get a brief idea about how different profiles are interacting with each other to make a product a valuable one.

The next step after completing the Designing and Verification of the ALU, so to syntheses its hardware using EDA tool and then the by applying BIST concepts the DFT will be done, so that the block can test itself. And then the PnR steps will be performed so that the complete ASIC flow can be understood with the help of simple ALU Block as a reference.

REFERENCES

[I] Sameer Palitkar, “Verilog HDL A guide to Digital Design and Synthesis” SunSoft Press, 1996.

[2] J Bhaskar. “Verilog Primer and Guide for Synthesis”, Springer.

[3] Chentouf Mohamed, Alaoui Ismaili and Zine El Abidine, "Physical Design Automation of Complex ASICs", IJCSI International Journal of Computer Science Issues, vol. 15, no. 1, January 2018, ISSN 1694-0814.

[4] S. N. Adya, M. C. Yildiz, I. L. Markov, P. G. Villarrubia, P. N. Parakh and P. H. Madden, “Bench-marking for Large-Scale Placement and Beyond”,IEEE Transactions on Computer-Aided Design ofIntegrated Circuits and Systems23(4) (2004), pp. 472-487..

[5] Arnold S. Tran; Richard A. Forsberg; Jack C. Lee, “A VLSI Design verification strategy”, IEETrasn. IBM Journal of Research and Development.

[6] A.E. Ruehli; G.S. Ditlow, “Circuit analysis, logic simulation, and design verification for VLSI”,Proceedings of the IEEE (Volume: 71 , Issue: 1).

[7] R. Aitken, G. Yeric, B. Cline, S. Sinha, L. Shifren, I. Iqbal and V. chandra, “Physical Design and FinFETs”, Proc. ACM International Symposium on Physical Design, 2014, pp. 65-68.

[8] Guirong Wu, Song Jia, Yuan Wang and Ganggang Zhang, "An efficient clock tree synthesis method in physical design," 2009.

[9] IEEE Standard for “Design and Verification of Low-Power Integrated Circuits” (IEEE 1801™-2013), New York, NY: IEEE.

[10] Robinson, David, “Aspect-Oriented Programming with the Verification Language”: A Pragmatic Guide for Testbench Developers, First Edition. Morgan Kauffman, 2007.

[11] Fredrik Wickberg, “HDL Code Analysis for ASICs In Mobile System”, Jan. 2007.

[12] Questa Clock Domain Crossing Datasheet, April 2017.
[13] [Synopsys]-Spyglass Reset Domain Crossing Verification Datasheet Author: Synopsys.

[14] [Synopsys]-Spyglass Clock Domain Crossing Verification Datasheet Author: Synopsys.

[15] Clock Domain Crossings Technical Paper Author: Cadence.

[16] SpyGlass® Predictive Analyzer User Guide Version 3.8.1, August 2006.

[17] SpyGlass® DC Rules Reference Version 3.8.0.3, July 2006.

[18] Spyglass Lint Verification Datasheet Author: Synopsys.

APPENDIX

The programming language used for the design of the ALU is Verilog and the verification has been done by System Verilog. The transcript for the simulation is given below.

```
# I am in the ALU_GEN
# I AM IN ALU_TX
# value for a= 176,b=157 ,sel=0 and out=x
# PACKET NUMBER 0
# I am in the ALU_GEN
# I AM IN ALU_TX
# value for a= 36,b=31 ,sel=0 and out=x
# PACKET NUMBER 1
# I am in the ALU_GEN
```

```
# I AM IN ALU_TX
# value for a= 246,b=101 ,sel=3 and out=x
# PACKET NUMBER 2
# I am in the ALU_GEN
# I AM IN ALU_TX
# value for a= 87,b=50 ,sel=2 and out=x
# PACKET NUMBER 3
# I am in the ALU_GEN
# I AM IN ALU_TX
# value for a= 136,b=105 ,sel=2 and out=x
# PACKET NUMBER 4
# I am in the ALU_GEN
# I AM IN ALU_TX
# value for a= 142,b=24 ,sel=2 and out=x
# PACKET NUMBER 5
# I am in the ALU_GEN
# I AM IN ALU_TX
# value for a= 194,b=193 ,sel=2 and out=x
# PACKET NUMBER 6
# I am in the ALU_GEN
# I AM IN ALU_TX
# value for a= 254,b=210 ,sel=1 and out=x
# PACKET NUMBER 7
# I am in the ALU_GEN
# I AM IN ALU_TX
# value for a= 238,b=154 ,sel=0 and out=x
# PACKET NUMBER 8
# I am in the ALU_GEN
# I AM IN ALU_TX
# value for a= 239,b=51 ,sel=3 and out=x
# PACKET NUMBER 9
# I am in the ALU_GEN
# I AM IN ALU_TX
# value for a= 253,b=248 ,sel=2 and out=x
# PACKET NUMBER 10
# I am in the ALU_GEN
```

```
# I AM IN ALU_TX
# value for a= 232,b=208 ,sel=3 and out=x
# PACKET NUMBER 11
# I am in the ALU_GEN
# I AM IN ALU_TX
# value for a= 219,b=129 ,sel=0 and out=x
# PACKET NUMBER 12
# I am in the ALU_GEN
# I AM IN ALU_TX
# value for a= 102,b=70 ,sel=1 and out=x
# PACKET NUMBER 13
# I am in the ALU_GEN
# I AM IN ALU_TX
# value for a= 147,b=112 ,sel=1 and out=x
# PACKET NUMBER 14
# I am in the ALU_GEN
# I AM IN ALU_TX
# value for a= 12,b=3 ,sel=2 and out=x
# PACKET NUMBER 15
# alu_scb::run
# alu_cov::run
# new received packet in monitor: a=176 b=157 sel=0 out= 77
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Addition test Passes
# alu_scb::run
# new received packet in monitor: a= 36 b= 31 sel=0 out= 67
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Addition test Passes
# alu_scb::run
# new received packet in monitor: a=246 b=101 sel=3 out=  2
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Division test passes
# alu_scb::run
```

```
# new received packet in monitor: a= 87 b= 50 sel=2 out=254
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Multiplication test Passes
# alu_scb::run
# new received packet in monitor: a=136 b=105 sel=2 out=200
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Multiplication test Passes
# alu_scb::run
# new received packet in monitor: a=142 b= 24 sel=2 out= 80
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Multiplication test Passes
# alu_scb::run
# new received packet in monitor: a=194 b=193 sel=2 out= 66
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Multiplication test Passes
# alu_scb::run
# new received packet in monitor: a=254 b=210 sel=1 out= 44
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Subtraction Test Paased
# alu_scb::run
run
# new received packet in monitor: a=238 b=154 sel=0 out=136
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Addition test Passes
# alu_scb::run
# new received packet in monitor: a=239 b= 51 sel=3 out=  4
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Division test passes
# alu_scb::run
```

```
# new received packet in monitor: a=253 b=248 sel=2 out= 24
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Multiplication test Passes
# alu_scb::run
# new received packet in monitor: a=232 b=208 sel=3 out=  1
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Division test passes
# alu_scb::run
# new received packet in monitor: a=219 b=129 sel=0 out= 92
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Addition test Passes
# alu_scb::run
# new received packet in monitor: a=102 b= 70 sel=1 out= 32
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Subtraction Test Paased
# alu_scb::run
# new received packet in monitor: a=147 b=112 sel=1 out= 35
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Subtraction Test Paased
# alu_scb::run
# new received packet in monitor: a= 12 b=  3 sel=2 out= 36
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Multiplication test Passes
# alu_scb::run
# new received packet in monitor: a= 12 b=  3 sel=2 out= 36
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Multiplication test Passes
# alu_scb::run
# new received packet in monitor: a= 12 b=  3 sel=2 out= 36
```

```
# COVERAGE IS GETTING COMPUTED:
# alu_cov::run
# Multiplication test Passes
# alu_scb::run
```

The System Verilog code for the environment is as follows:

```
//Transaction class
class alu_tx;
rand  logic [7:0] a;
rand  logic [7:0] b;
rand  logic [1:0] sel;
logic [7:0] out ;
function print();
$display ("value for a= %0d,b=%0d ,sel=%0d and out=%0d",a,b,sel,out);
endfunction
constraint a_greater_b {a > b;}
constraint b_not_eqaul_zero {b>0;}
endclass
//driver class
class alu_drv;
alu_tx tx;
 virtual alu_intf vif;
mailbox # (alu_tx) gen2drv;
function new();
this.gen2drv=alu_cfg::gen2drv;
this.vif=alu_cfg::vif;
endfunction
task run();
forever
begin
wait (!vif.rst);
gen2drv.get(tx);
@ (posedge vif.clk)
Begin
vif.a=tx.a;
vif.b=tx.b;
```

```
vif.sel=tx.sel;
end
end
task
endclass
//generator class
typedef class alu_cfg;
class alu_gen;
alu_tx tx;
mailbox #(alu_tx) gen2drv;
function new ();
this.gen2drv=alu_cfg::gen2drv;
endfunction
task run();
begin
for (int i=0;i<256; i++)
begin
tx=new();
assert(tx.randomize());
tx.print();
gen2drv.put(tx);
$display ("PACKET NUMBER %0d ",i);
end
end
endtask
endclass
//monitor class
class alu_mon;
alu_tx tx;
virtual alu_intf vif;
mailbox #(alu_tx) mon2scb;
mailbox #(alu_tx) mon2cov;
function new ();
this.mon2scb=alu_cfg::mon2scb;
this.mon2cov=alu_cfg::mon2cov;
this.vif = alu_cfg::vif;
```

```
endfunction
task run ();
@(posedge vif.clk);
Forever
Begin
while(vif.rst)
@(posedge vif.clk);
tx=new();
#1;
tx.a=vif.a;
tx.b=vif.b;
tx.sel=vif.sel;
@(posedge vif.clk);
tx.out=vif.out;$display ("new received packet in monitor: a=%d b=%d sel=%d out=%d",tx.a,tx.b,tx.sel,tx.out);
mon2scb.put(tx);
mon2cov.put(tx);
end
endtask
endclass
//interface
interface alu_intf (input logic clk, input logic rst);
logic [7:0] a,b;
logic [1:0] sel;
logic [7:0] out;
endinterface
//scoreboard
class alu_scb;
mailbox #(alu_tx) mon2scb;
alu_tx tx;
function new();
this.mon2scb=alu_cfg::mon2scb;
endfunction
task run ();
forever
begin
mon2scb.get(tx);
```

```
case (tx.sel)
2'b00: begin
if (tx.a + tx.b == tx.out)
$display("Addition test Passes");
Else
$display("Addition test FAILS");
End
2'b01:  begin
if (tx.a- tx.b == tx.out)
$display("Subtraction Test Paased");
Else
$display ("Subtraction test FAILS");
End
2'b10:  begin
if (tx.a * tx.b == tx.out)
$display ("Multiplication test Passes");
Else
$display ("Multioplication test FAILS");
end
2'b11: begin
if (tx.a /tx.b == tx.out)
$display ("Division test passes");
Else
$display("Division test FAILS");
end
default: $display("SELECT DOES NOT HAVE A VALID VALUE");
endcase
end
endtask
endclass
// functional coverage class
class alu_cov;
mailbox # (alu_tx) mon2cov;
alu_tx tx;
covergroup alu_cg;
alu_a   :coverpoint tx.a {bins a []={[0:255]};}
```

```
alu_b   :coverpoint tx.b {bins b [50]={[0:100]};}
alu_sel :coverpoint tx.sel {bins sel []={[0:3]};}
endgroup
function new();
alu_cg=new();
this.mon2cov=alu_cfg::mon2cov;
endfunction
task run();
forever
begin
mon2cov.get(tx);
alu_cg.sample();
end
endtask
endclass
//class  for defining static mailbox
class alu_cfg;
static mailbox # (alu_tx) gen2drv=new();
static mailbox # (alu_tx) mon2scb=new();
static mailbox # (alu_tx) mon2cov=new();
static virtual alu_intf vif;
endclass
//environment class
class alu_env;
alu_gen gen;
alu_drv drv;
alu_mon mon;
alu_scb scb;
alu_cov cov;
function new();
gen=new();
drv=new();
mon=new();
scb=new();
cov=new();
endfunction
```

```
task run();
begin
fork
gen.run();
drv.run();
mon.run();
scb.run();
cov.run();
 join
end
endtask
endclass
//top class
module alu_top();
logic clk,rst;
alu_intf intf (clk,rst);
alu_tb tb ();
alu a1 (intf.a,intf.b,intf.sel,clk,rst,intf.out);
initial
begin
clk=0;
end
always
#5 clk=~clk;
initial
begin
rst=1;
#10 rst=0;
end
initial
begin
alu_cfg::vif=intf;
end
endmodule
```

The Verilog code for the ALU Design is as follows:

```
//ALU module
module alu(input [7:0] a,b,
input [1:0] sel,
input clk,rst,
output reg [7:0] out);
always @(posedge clk)
begin
if (rst)
out=0;
else
begin
case(sel)
2'b00: // Addition
out = a + b ;
2'b01: // Subtraction
out = a - b ;
2'b10: // Multiplication
out = a * b;
2'b11: // Division
out = a/b;
default: out= a + b ;
endcase
end
end
endmodule
//alu testbench
program alu_tb();
alu_env env;
initial
begin
env=new();
env.run();
end
endprogram
```