Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electrical & Electronic Engineering

Course Title: Industrial Training

Course No: EEE 4100

Report on
"Industrial Attachment at Siliconova,
Semiconductor Design Engineering & Manufacturing Company."

Submitted By

Jannat Jahan Pretosi

Student ID: 1901167

Department of Electrical and Electronic Engineering
Rajshahi University of Engineering & Technology

Submitted To

Dr. G.K.M. Hasanuzzaman

Associate Professor

Department of Electrical and Electronic Engineering
Rajshahi University of Engineering & Technology

Declaration

This is certified that the attachment work entitled **"Industrial Attachment at Siliconova, Semiconductor Design Engineering & Manufacturing Company."** has been carried out by Jannat Jahan Pretosi, Roll: 1901167 under my supervision in the department of Electrical and Electronic Engineering at Rajshahi University of Engineering & Technology, Rajshahi.

Attachment Supervisor

.....

Dr. G.K.M. Hasanuzzaman

Associate Professor

Department of Electrical and Electronic Engineering

Rajshahi University of Engineering & Technology

Rajshahi, Bangladesh

Acknowledgement

Industrial training bridges the gap between academic theory and workplace reality by letting students explore career interests and gain practical skills for a head start in the job market. I would like to express my inner gratitude to the Almighty Allah for bringing me this far and helping me to successfully complete my internship.

I am sincerely grateful and humble to my respected supervisor, **Dr. G.K.M. Hasanuzzaman** Associate Professor Department of Electrical & Electronic Engineering, Rajshahi University of Engineering & Technology, Bangladesh, for her guidance, encouragement and supervision throughout this industrial attachment.

I would like to acknowledge the invaluable guidance provided by **Professor Dr. Md. Selim Hossain**, the honorable Head of the Department of Electrical & Electronic Engineering, Rajshahi University of Engineering & Technology, Bangladesh.

I want to convey my gratitude to the management of Filament Engineering Ltd. for allowing me to complete my internship at their organization. I am deeply thankful to **Engr. Sirajul Alam Khan**, Founder & CEO of Siliconova.

Lastly, my sincere gratitude goes out to the resolute assistance, continual encouragement, and inspiration that I have received from all of my teachers, friends, and family throughout my academic journey.

Author

Jannat Jahan Pretosi

Undergraduate Student

Department of Electrical and Electronic Engineering

Rajshahi University of Engineering & Technology

Rajshahi, Bangladesh.

Contents

Chapter 1

Introduction

In the ever-evolving landscape of semiconductor design engineering, a comprehensive understanding of Application-Specific Integrated Circuit (ASIC) flow is indispensable. ASIC flow encapsulates a series of meticulous steps that are imperative in the creation of high performance and innovative chip designs. With this in mind, our industrial training at Siliconova Limited provided us with invaluable insights into ASIC flow, alongside an in-depth exploration of associated tools and methodologies such as Verilog and Cadence.

1.1 Objective of this Attachment

The primary objective of our attachment at Siliconova Limited was to gain practical exposure and hands-on experience in ASIC flow. By immersing ourselves in the intricacies of ASIC design, we aimed to augment our theoretical knowledge with real-world applications. Through this attachment, we sought to enhance our proficiency in Verilog, Cadence, and other essential tools utilized in the semiconductor design process.

1.2 Scope and Methodology

Our attachment encompassed a comprehensive exploration of ASIC flow, delving into each step with meticulous detail. From the initial RTL design to the final tape-out, we engaged in practical exercises and projects aimed at honing our skills and understanding. Additionally, we had the opportunity to collaborate with expert engineers at Siliconova, gaining valuable insights and guidance throughout the process.

1.3 Company Profile

Siliconova Limited stands as a leading semiconductor design engineering service provider, renowned for delivering cutting-edge solutions tailored to meet the evolving demands of the electronics industry. With a focus on precision and innovation, Siliconova specializes in every aspect of semiconductor design, from RTL design to tape-out, ensuring optimal performance and reliability in chip designs.

SILICONOVA LIMITED engaged in the design, packaging & testing of semiconductors and semiconductor devices, such as Central Processing Unit (CPU), Microprocessor (MCU), RISC and RISC-V Processors, AI and Knowledge-based Processors, Memory, RAM, Storage, SSD, Ethernet, transistors, integrated circuits (IC), Application Specific Integrated Circuit (ASIC), System on Chip (SOC), Field programable Gate Array (FPGA) and Microchips for Computer Motherboard, Data Center Electronics Devices,

IoT and IIoT Devices, Wireless, Mobile, Networking, Wi-Fi, Small Cells, Wireless Power, Augmented Reality, Location Services, Automotive, 5G, PC computing. They devote themselves to developing high-performance/low-power 32/64 bit processors and their associated SoC to serve the rapidly growing embedded system applications worldwide.

SILICONOVA is a fast-growing company in the field of VLSI in Bangladesh. Their dedication and passion in this field makes them one of the best in this field. Along with it their mentality of providing opportunity to young aspirants really makes them stand out in the mind of youths. Their quality assurance is helping them grow their business.

1.4 Overview of SILICONOVA

SILICONOVA LIMITED is a company dedicated to VLSI field located in Metro Rail Station, Level-09, Genusys Heights, 623 & 624 Begum Rokeya Sharani, Kazipara, Dhaka 1216, Exit D, Dhaka 1216.

The leadership of SILICONOVA is entrusted to Sirajul Alam Khan, who is the CTO and the founder of the company. The company has a team of 24 members with One Director, One Advisor, One Business Advisor, Three Verification Engineers, Four Design Verification Engineers, Eight Physical Design Engineers, Four Layout Design Engineers, and One IC Package Design Engineer. Nine of the team members are from Rajshahi University of Engineering & Technology including The Founder & The CTO Sirajul Alam Khan himself.

1.5 Company Mission and Vision

The mission of Siliconova Limited is to revolutionize the semiconductor industry through unparalleled expertise and innovation. By leveraging cutting-edge technologies and a team of expert engineers, Siliconova aims to deliver superior semiconductor solutions that drive the advancement of electronics worldwide. The vision of Siliconova is to be recognized as a global

leader in semiconductor design engineering, setting new standards of excellence and pushing the boundaries of what is possible in chip design.

1.6 VLSI

VLSI, or Very Large-Scale Integration, represents a pivotal milestone in the evolution of electronics. Its roots trace back to the mid-20th century when the transistor revolutionized the field. In the 1950s and 1960s, the development of integrated circuits (ICs) began to gain momentum. Jack Kilby at Texas Instruments and Robert Noyce at Fairchild Semiconductor independently pioneered the concept of integrating multiple transistors and other components onto a single silicon substrate. This breakthrough laid the groundwork for VLSI. The term "VLSI" started gaining prominence in the 1970s as advancements in semiconductor fabrication techniques enabled the integration of hundreds to thousands of transistors onto a single chip. This period saw the birth of the microprocessor, exemplified by the Intel 4004 in 1971, which packed over 2,000 transistors, heralding the era of VLSI. The late 1970s and 1980s witnessed accelerated growth in VLSI technology, fueled by the introduction of Computer-Aided Design (CAD) tools. These tools empowered engineers to design and simulate complex circuits more efficiently, leading to rapid advancements in chip complexity and performance. Moore's Law, proposed by Intel co-founder Gordon Moore in 1965, predicted a doubling of transistor density roughly every two years. This prediction served as a guiding principle for the semiconductor industry, driving relentless innovation in process technology and chip design. As a result, the number of transistors on a chip grew exponentially, leading to increasingly powerful and compact electronic devices. In the 21st century, VLSI technology continued its march forward with the introduction of advanced process nodes such as 7nm, 5nm, and beyond. These cutting-edge technologies enabled the integration of billions of transistors onto a single chip, paving the way for complex Systems-on-Chip (SoCs) powering smartphones, data centers, artificial intelligence, and more. VLSI technology stands as a testament to human ingenuity and innovation. From its humble beginnings with a few transistors per chip to today's billion-transistor designs, VLSI has transformed the world of electronics, ushering in an era of unprecedented computational power and connectivity. Its journey from concept to reality is a story of relentless pursuit of miniaturization, performance enhancement, and technological excellence.

1.7 History of VLSI

Early History:

- **1950s-1960s:** The advent of the transistor marked the beginning of modern electronics. Initially, circuits were built using discrete components (transistors, resistors, capacitors) on circuit boards.
- **1970s:** With the development of integrated circuits (ICs), multiple transistors could be fabricated on a single silicon chip, significantly reducing size and improving performance.
- **1971:** The invention of the microprocessor by Intel (the Intel 4004) marked a significant milestone in the history of VLSI, integrating thousands of transistors on a single chip.

Emergence of VLSI:

- **Late 1970s:** VLSI emerged as a distinct field with the ability to integrate hundreds of thousands of transistors on a single chip. Companies like Intel, Texas Instruments, and Motorola pioneered VLSI design and manufacturing.
- **1980s:** The introduction of CAD tools for VLSI design accelerated the development process, allowing engineers to design and simulate complex circuits more efficiently.

Progress and Scaling:

- **1980s-1990s:** VLSI technology underwent rapid advancements, driven by Moore's Law, which predicted a doubling of transistor density roughly every two years. This led to the development of increasingly dense and powerful microprocessors and memory chips.
- **2000s:** The transition to sub-micron process technologies (90nm, 65nm, 45nm, etc.) enabled even greater transistor density and improved performance.
- **2010s:** VLSI technology continued to advance with the introduction of FinFET transistors and 3D integration techniques, allowing for greater energy efficiency and performance.
- **2020s:** VLSI technology continued to push the limits of miniaturization, with the introduction of advanced process nodes such as 7nm, 5nm, and beyond. This enabled the integration of billions of transistors on a single chip, facilitating the development of complex SoCs for applications like artificial intelligence, 5G, and high-performance computing.

Chip Complexity:

- **Early Stage:** In the early days of VLSI, chips contained thousands to tens of thousands of transistors. The Intel 4004, for example, had around 2,300 transistors.
- **Today:** Modern VLSI chips can contain billions of transistors. For instance, advanced microprocessors, GPUs, and SoCs for smartphones may contain anywhere from several hundred million to over 50 billion transistors, depending on their complexity and application.

Chapter Two

ASIC Flow

2.1 History of ASIC

Early ASICs used gate array technology. By 1967, Ferranti and Interdesign were manufacturing early bipolar gate arrays. In 1967, Fairchild Semiconductor introduced the Micromatrix family of bipolar diode–transistor logic (DTL) and transistor–transistor logic (TTL) arrays.

Complementary metal–oxide–semiconductor (CMOS) technology opened the door to the broad commercialization of gate arrays. The first CMOS gate arrays were developed by Robert Lipp, in 1974 for International Microcircuits.

Metal–oxide–semiconductor (MOS) standard-cell technology was introduced by Fairchild and Motorola, under the trade names Micromosaic and Polycell, in the 1970s. This technology was later successfully commercialized by VLSI Technology (founded 1979) and LSI Logic (1981).

A successful commercial application of gate array circuitry was found in the low-end 8-bit ZX81 and ZX Spectrum personal computers, introduced in 1981 and 1982. These were used by Sinclair Research (UK) essentially as a low-cost I/O solution aimed at handling the computer's graphics.

Customization occurred by varying a metal interconnect mask. Gate arrays had complexities of up to a few thousand gates; this is now called mid-scale integration. Later versions became more generalized, with different base dies customized by both metal and polysilicon layers. Some base dies also include random-access memory (RAM) elements.

2.2 ASIC Flow

Application-Specific Integrated Circuit (ASIC) flow is a systematic approach to designing custom integrated circuits tailored to specific applications or functions. ASICs are semiconductor devices that are purpose-built to perform a particular task, unlike general-purpose microprocessors. ASIC flow encompasses a series of sequential steps that transform a design concept into a manufacturable ASIC chip. This chapter provides a comprehensive overview of ASIC flow, detailing each step involved in the process.

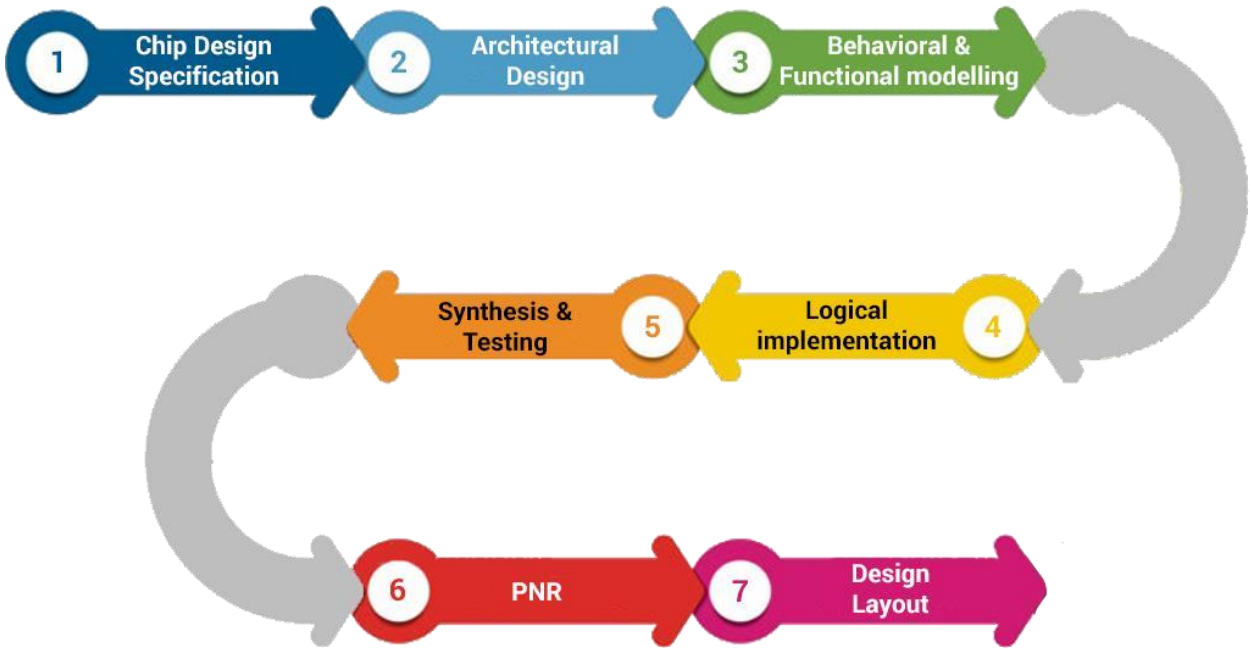

Fig 2.1 ASIC Flow

1. Specification and Architecture Definition:

During the specification and architecture definition phase, the key functionalities, performance requirements, and constraints of the ASIC are established. This phase involves close collaboration between the design team and stakeholders to gather and analyze the requirements. The architectural framework of the ASIC, including block-level partitioning and high-level design decisions, is defined. Additionally, considerations such as power consumption, area constraints, and interface specifications are taken into account. Clear and comprehensive specifications set the foundation for the subsequent stages of ASIC design.

2. RTL Design:

Register Transfer Level (RTL) design is a crucial phase where the architectural description of the ASIC is translated into a hardware description language (HDL) such as Verilog or VHDL. RTL coding involves specifying the behavior of digital logic circuits using registers, combinational logic, and control structures. Designers express the functionality of the ASIC in terms of logical operations, data flows, and control signals. RTL code serves as the starting point for synthesis and subsequent verification stages, guiding the implementation of the ASIC.

3. Synthesis:

Synthesis is the process of converting RTL code into a gate-level netlist, representing the logical structure of the ASIC in terms of standard cells from a library. During synthesis, the RTL code is analyzed, optimized, and mapped to target technology-specific libraries. Optimization techniques such as logic restructuring, technology mapping, and timing optimization are applied to meet

performance, area, and power constraints. The synthesized netlist serves as the input for the subsequent stages of ASIC design, guiding physical implementation and verification efforts.

4. Floor-planning and Partitioning:

Floor-planning and partitioning involve the physical organization of the ASIC's components and the allocation of resources on the silicon die. Floor-planning determines the placement of functional blocks, memories, and I/O pads to optimize for factors such as signal integrity, timing closure, and area utilization. Partitioning involves dividing the design into manageable blocks for ease of implementation and verification. Careful floor-planning and partitioning lay the groundwork for successful place and route, ensuring efficient use of silicon area and meeting design goals.

5. Place and Route:

Place and route is the stage where the synthesized logic cells are physically placed onto the silicon die, and interconnections between them are established. Place and route algorithms optimize the placement of cells to minimize wirelength, reduce signal delays, and meet timing constraints. Routing algorithms determine the paths of interconnect wires while adhering to design rules and constraints imposed by the fabrication process. Place and route plays a critical role in achieving timing closure, signal integrity, and manufacturability goals for the ASIC design.

6. Static Timing Analysis (STA):

Static Timing Analysis (STA) is performed to verify that the ASIC design meets timing requirements and operates correctly under all conditions. STA evaluates the timing characteristics of the design, including setup and hold times, clock skew, and maximum operating frequency. Timing constraints specified during synthesis and floorplanning are verified against the final layout to ensure timing closure. Timing violations, such as setup and hold violations, are identified and resolved iteratively through design optimizations and adjustments.

7. Design Verification:

Design verification is a critical phase aimed at validating the functionality, correctness, and compliance of the ASIC design with specifications. Verification methodologies such as simulation, formal verification, and emulation are employed to ensure that the ASIC behaves as intended under various operating conditions and corner cases. Functional correctness, performance, and compliance with industry standards are rigorously verified to uncover and rectify design bugs and errors before tape-out. Design verification efforts play a crucial role in ensuring the reliability and quality of the final ASIC product.

8. Tape-Out:

Tape-out marks the final stage of ASIC flow, where the design files are prepared for fabrication by the semiconductor foundry. The tape-out process involves generating a set of files, including the final layout database (GDSII), mask data, and design rule check (DRC) reports, which are submitted to the foundry for manufacturing. Tape-out signifies readiness for production and represents the culmination of extensive design efforts. Once tape-out is complete, the ASIC design enters the fabrication process, leading to the production of physical ASIC chips.

In summary, ASIC flow is a comprehensive and iterative process that involves multiple stages, from specification and architecture definition to tape-out. Each stage requires careful planning, design, optimization, and verification to ensure the successful realization of custom integrated circuits. By meticulously executing ASIC flow, semiconductor designers can develop high performance and innovative ASICs that meet the unique requirements of diverse applications and industries.

2.3 Renowned Companies in the Field of ASIC

AMD (Advanced Micro Devices):

AMD, headquartered in Santa Clara, California, USA, stands as a global semiconductor powerhouse renowned for its high-performance computing and graphics solutions. With a focus on innovation, AMD designs and manufactures Application-Specific Integrated Circuits (ASICs) tailored for various applications, including gaming consoles, data centers, and personal computers.

NVIDIA Corporation:

NVIDIA Corporation, also headquartered in Santa Clara, California, USA, is a leading technology company with a primary focus on graphics processing units (GPUs) and AI computing. Renowned for its cutting-edge ASIC designs, NVIDIA produces chips for graphics processing, artificial intelligence, and autonomous vehicles, among other applications.

Intel Corporation:

Intel Corporation, based in Santa Clara, California, USA, is one of the world's largest semiconductor chip manufacturers. With a rich history of innovation, Intel designs and produces ASICs for diverse applications, including data centers, IoT devices, and networking infrastructure, contributing significantly to technological advancements globally.

Samsung Electronics:

Samsung Electronics, headquartered in Suwon, South Korea, is a multinational conglomerate renowned for its electronic products, including semiconductors. Leveraging its expertise, Samsung designs and manufactures ASICs for smartphones, consumer electronics, and various industrial applications, playing a pivotal role in shaping the semiconductor industry.

TSMC (Taiwan Semiconductor Manufacturing Company):

TSMC, based in Hsinchu, Taiwan, stands as the world's largest dedicated independent semiconductor foundry. With a focus on manufacturing excellence, TSMC produces ASICs for a diverse customer base, including fabless semiconductor companies, spanning applications from smartphones to high-performance computing, driving technological innovation forward.

Broadcom Inc.:

Broadcom Inc., headquartered in San Jose, California, USA, is a global technology company offering a broad range of semiconductor and infrastructure software solutions. With expertise in ASIC design and development, Broadcom supplies chips for networking, storage, and wireless communications, contributing to the advancement of connectivity technologies worldwide.

Xilinx Inc. (Acquired by AMD in 2021):

Xilinx Inc., formerly headquartered in San Jose, California, USA, was a leading provider of programmable logic devices and associated technologies. Before its acquisition by AMD, Xilinx offered ASIC solutions for telecommunications, automotive, aerospace, and defense applications, contributing significantly to the advancement of customizable computing solutions.

Qualcomm Incorporated:

Qualcomm Incorporated, based in San Diego, California, USA, is a multinational semiconductor and telecommunications equipment company renowned for its development of wireless technologies. With a focus on innovation, Qualcomm designs and produces ASICs for mobile devices, IoT, and wireless communication infrastructure, driving connectivity and mobility solutions globally.

2.4 ASIC VS FPGA

Option	ASIC	FPGA
Flexibility	ASIC is designed and integrated to perform a certain fixed function with no ability to reprogram it	FPGA is a reconfigurable chip where interconnected logic blocks can have a custom purpose
Performance	Performs better as each function is already set	Reprogrammable nature causes lower performance
Power consumption	Consumes less power due to the functionality-specific design	Consumes more power performing the same functions as ASICs
Time to market	More suitable for long-term projects as the design process has 8 steps and takes a lot of time	Fast time to market due to the quicker design process with 3 main stages
NRE	ASICs provide higher NRE cost, but thanks to functionality layering, it can also be changed or expanded on-premise	FPGAs provide lower NRE as this integrated circuit is comprised of
Design flow	Consists of eight main steps, which result in a more time-consuming design process	Consists of three main stages only ensuring fast time to market

Table 2.1 Comparison between ASIC and FPGA

Chapter Three

Physical Design Flow

3.1 Physical Design

Physical design in the semiconductor industry refers to the process of transforming a logical design, typically described in a hardware description language (HDL) such as Verilog or VHDL, into a physical representation that can be manufactured as an integrated circuit (IC). This process involves translating the logical design into a layout of interconnected components on a silicon wafer, considering factors such as performance, power consumption, area, and manufacturability. In RTL to GDS flow, Physical Design is an important stage.

In physical design, synthesized netlist, design constraints and standard cell library are taken as inputs and converted to a layout (gds file) which should be as per the design rules provided by the foundry. Further, this layout is sent to the foundry for the fabrication of a chip. This whole process of converting the gate-level netlist to layout is termed as physical design.

There are various stages of design, various mandatory checks in each stage and involved various analysis and verifications.In this article, we will see an overall flow of physical design and details of each stage, sanity checks, analysis and verifications will be covered in the coming articles. Here is a basic physical design flow. There are some minor changes in this flow from company to company.

In Physical design, flow start with some set of input files and do the sanity check first once the design is loded into PnR tool. Sanity checks before floor plan are must in order to make sure that netlist, standard cell library and constraint are correct or not. After that floorplan stage starts where the macro placement is done. A good floorplan of design is a critical thing, it decides the overall quality of your design. If floorplan is not well it may lead to several issues in the next stages and it is quite possible that we need to change the floorplan and start again. In general, there are many iterations are required for a physical design engineer to get a quality floorplan. If a block is macro dominant and cell density is high than the floorplan stage is more critical. A good floorplan help to achieve a less congestion and good timing. How to do a good floorplan will be discussed in the other article.

Once the floorplan is done, we need to create the power plan followed by adding antenna diodes, well tap cells, endcap cells, decap cells. What are these cells and why we need will be discussed in the coming article. Generally, this step is called pre-placement stage. Once the preplacement is done we need to again perform a sanity check before the placement stage.

3.2 Physical Design Flow

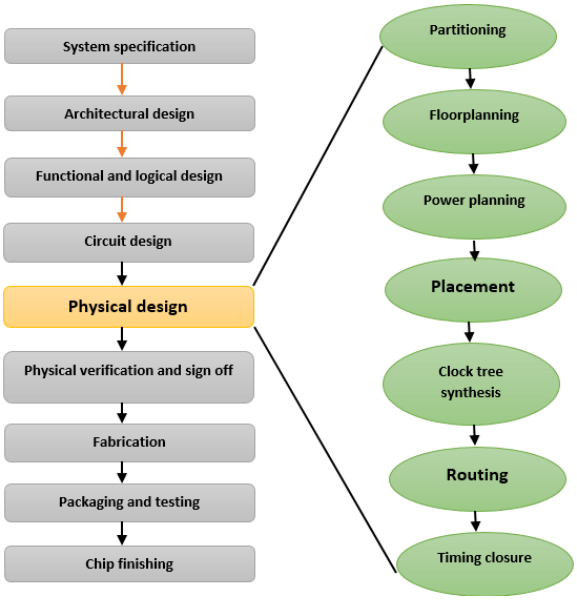

Fig.3.1 Physical Design Flow

3.3 Synthesis

Synthesis in the semiconductor industry refers to the process of converting a high-level hardware description of a digital circuit, typically written in a hardware description language (HDL) such as Verilog or VHDL, into a gate-level representation that can be implemented on a specific hardware platform, such as an application-specific integrated circuit (ASIC) or a field-programmable gate array (FPGA).

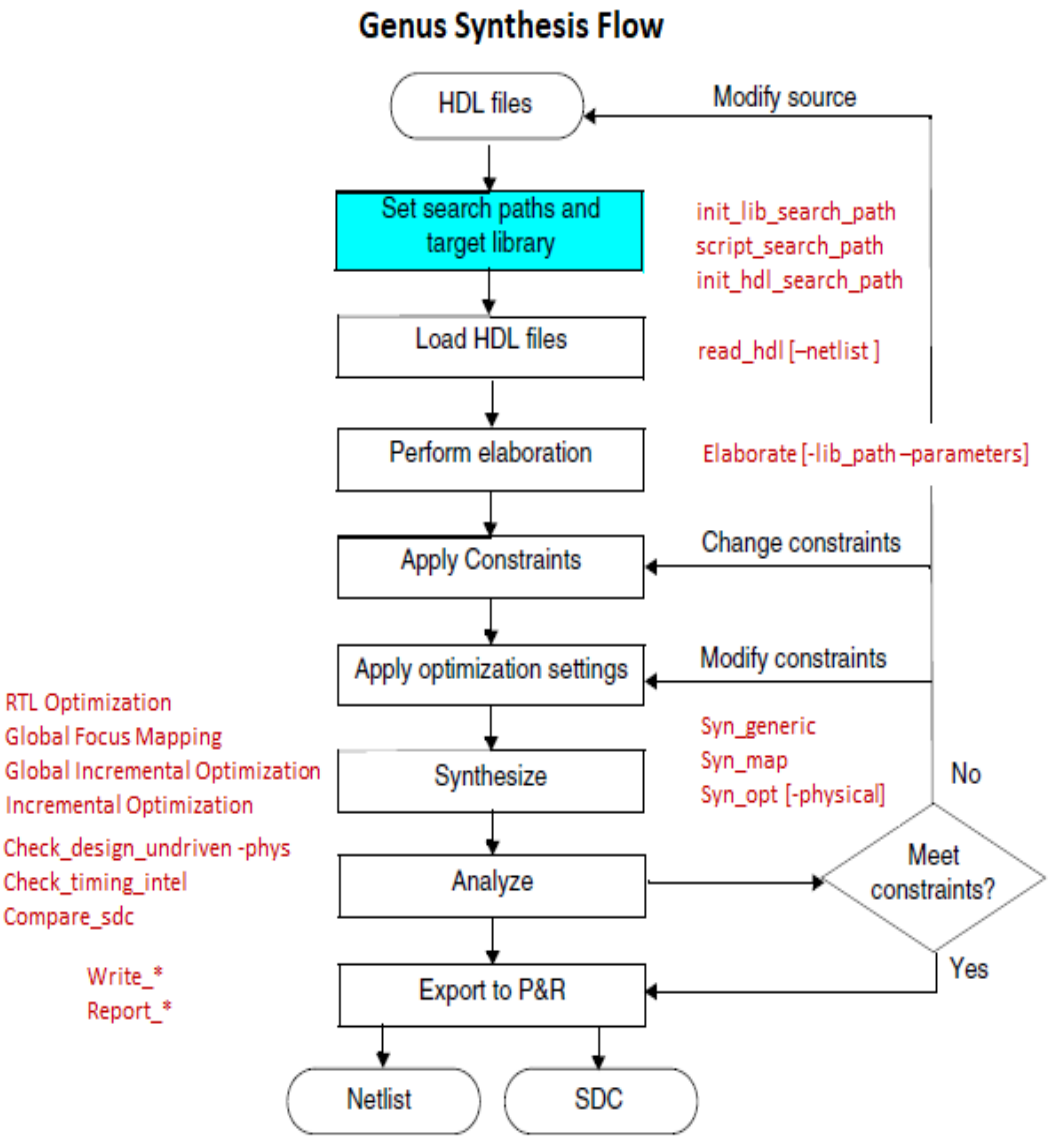

Fig.3.2 Synthesis Workflow

Here's a breakdown of the synthesis process:

1. **Parsing and Analysis:** The synthesis tool parses the HDL code and performs syntax and semantic analysis to understand the design hierarchy, signals, and their interactions.

2. **Optimization**: Once the design is parsed, the synthesis tool performs various optimizations to improve the design's performance, area utilization, and power consumption.

Optimization techniques include logic restructuring, constant propagation, dead code elimination, and resource sharing.

3. **Technology Mapping:** In this step, the synthesis tool maps the abstract logic elements defined in the HDL code to specific cells or components available in the target technology library. This mapping is based on the characteristics of the target technology, such as its available logic gates, flip-flops, and other resources.

4. **Timing Analysis:** Timing analysis is performed to ensure that the synthesized design meets the required timing constraints. The synthesis tool estimates the delay through the logic paths and verifies that the design operates within the specified clock frequency and meets setup and hold time requirements.

5. **Area Estimation:** The synthesis tool estimates the area utilization of the synthesized design, including the number of logic gates, flip-flops, and other resources required to implement the design on the target hardware platform. This information helps designers evaluate the design's scalability and resource utilization.

6. **Constraint Handling:** Synthesis tools allow designers to specify various constraints, such as timing constraints, area constraints, and optimization directives. These constraints guide the synthesis process and ensure that the synthesized design meets the desired performance, area, and power targets.

7. **Output Generation:** Once synthesis is complete, the synthesis tool generates output files that represent the synthesized design in a format suitable for downstream processes, such as place-and-route or FPGA configuration. These output files typically include a netlist describing the synthesized logic and additional reports detailing the design's characteristics, such as timing reports and area reports.

Synthesis is a critical step in the digital design flow, enabling designers to convert abstract hardware descriptions into optimized gate-level representations that can be implemented on specific hardware platforms. By leveraging synthesis tools and methodologies, designers can achieve faster time-to-market, higher performance, and lower power consumption for their digital designs.

3.4 Floor-planning

Floor-planning is a crucial step in the physical design of integrated circuits (ICs), where the goal is to create an optimal layout of the various functional blocks and components on the semiconductor die. It involves determining the physical placement of these blocks to achieve specific design objectives such as performance, power consumption, area utilization, and manufacturability.

Here are the key aspects and considerations involved in Floor-planning:

1. **Hierarchical Organization:** Floor-planning typically starts with a hierarchical organization of the design, where larger functional blocks are grouped together into subsystems or modules. This hierarchical structure helps manage complexity and enables efficient design exploration and optimization.

2. **Partitioning:** Partitioning involves dividing the design into smaller blocks or regions based on functional boundaries, performance requirements, or physical constraints. This partitioning can be based on factors such as clock domains, power domains, or signal integrity considerations.

3. **Block Placement:** Once the design is partitioned, the next step is to determine the physical placement of these blocks within the silicon die. Block placement aims to optimize various metrics such as wire length, signal delay, and power distribution. Techniques such as analytical placement algorithms, simulated annealing, and genetic algorithms are commonly used for block placement.

4. **Pin Placement:** Pin placement involves determining the locations of input/output (I/O) pins or pads for connecting the IC to external devices or other components on the circuit board. Proper pin placement is critical for ensuring signal integrity, minimizing signal delays, and simplifying the routing process.

5. **Power Grid Planning:** Floorplanning also includes planning the distribution of power and ground signals throughout the chip. This involves placing power pads, creating power distribution networks, and optimizing power grid density to minimize voltage drop and ensure reliable operation.

6. **Clock Tree Synthesis (CTS):** In synchronous digital designs, floorplanning includes planning the distribution of clock signals using a clock tree. Clock tree synthesis involves determining the locations of clock sources, clock sinks (flip-flops), and intermediate clock buffers to minimize clock skew and ensure timing closure.

7. S**ignal Routing Channels:** Floorplanning defines the locations and widths of signal routing channels between the functional blocks. These routing channels provide paths for interconnecting the blocks and must be designed to accommodate the required signal density and routing congestion.

8. **Physical Design Constraints:** Floorplanning must adhere to various physical design constraints imposed by the manufacturing process, such as minimum feature sizes, metal layer stack-up, design rule checks (DRC), and design for manufacturability (DFM) guidelines.

9. **Design Exploration and Optimization:** Floorplanning is an iterative process that involves exploring different placement and routing options to optimize the design for performance, area, power, and other metrics. Designers may use floorplanning tools, placement algorithms, and manual adjustments to iteratively refine the floorplan until the desired objectives are met.

Overall, floorplanning plays a critical role in the physical design flow of integrated circuits, influencing factors such as chip size, performance, power consumption, and manufacturability. A well-executed floorplan can significantly impact the success of the overall IC design by balancing competing design objectives and constraints.

3.5 Power-planning

Power planning is a fundamental aspect of physical design in the semiconductor industry, focusing on the distribution and management of power and ground signals throughout an integrated circuit (IC). It ensures that the IC operates reliably and efficiently while minimizing power consumption and mitigating potential issues such as voltage drop and electromigration.

Here's a detailed overview of power planning:

1. **Power Grid Architecture:** Power planning begins with defining the power grid architecture, which includes the placement of power pads (sources) and ground pads (sinks) on the IC's periphery. These pads serve as entry and exit points for power and ground signals, connecting the IC to external power sources and ground planes on the printed circuit board (PCB).

2. **Power Distribution Networks (PDN):** Once the power pads are placed, power planning involves creating the power distribution network (PDN) within the IC. The PDN consists of a hierarchical network of metal power lines (power rails) and vias that distribute power and ground signals to the various functional blocks and components on the IC.

3. **Decoupling Capacitors:** Decoupling capacitors are strategically placed throughout the IC to provide local charge storage and filtering, reducing noise and voltage fluctuations caused by switching activities. Power planning involves determining the locations and sizes of decoupling capacitors based on the dynamic power requirements of the adjacent logic cells.

4. **Power Rail Design:** Power planning includes designing the power rails to minimize voltage drop and ensure uniform power distribution across the IC. This may involve optimizing the width and spacing of power lines, as well as adding additional metal layers or power straps to reduce resistance and impedance.

5. **Clock Domain Power Distribution:** In synchronous digital designs, power planning extends to the distribution of clock signals within each clock domain. Clock gating techniques may be employed to dynamically disable clock signals to unused logic regions, reducing power consumption during idle periods.

6. **Power Integrity Analysis:** Power planning is accompanied by power integrity analysis, which assesses the quality and robustness of the power distribution network. This analysis includes simulations to verify that the PDN can deliver sufficient current without excessive voltage droop or noise, ensuring reliable operation under various operating conditions.

7. **Grounding Strategy:** Power planning also considers the grounding strategy, ensuring that ground signals are distributed effectively to minimize ground bounce and noise. Ground rings or grids may be used to enhance ground distribution and improve signal integrity.

8. **Low-Power Design Techniques:** Power planning may incorporate various low-power design techniques such as voltage scaling, clock gating, power gating, and dynamic voltage and frequency scaling (DVFS) to further reduce power consumption while maintaining performance.

9. **Design Rule Checks (DRC):** Power planning must comply with design rule checks (DRC) to ensure that the power grid meets the specifications and constraints of the semiconductor manufacturing process. This includes checking for minimum width and spacing requirements, via densities, and other layout rules.

10. **Physical Verification:** Once the power planning is complete, physical verification tools are used to perform checks such as electrostatic discharge (ESD) protection,

electromigration analysis, and thermal analysis to ensure the reliability and robustness of the power distribution network.

By carefully planning and optimizing the power distribution within an IC, power planning plays a critical role in achieving efficient, reliable, and high-performance semiconductor designs. It is an essential aspect of the overall physical design flow, influencing factors such as power consumption, signal integrity, and manufacturability.

3.6 Placement stage

After the preplacement, we do the placement where all the standard cells are placed and legalized. There are various steps placement stage which tool performs, these steps will be discussed on a later article. Once the placement is done we need to perform the optimization for better timing and congestions.

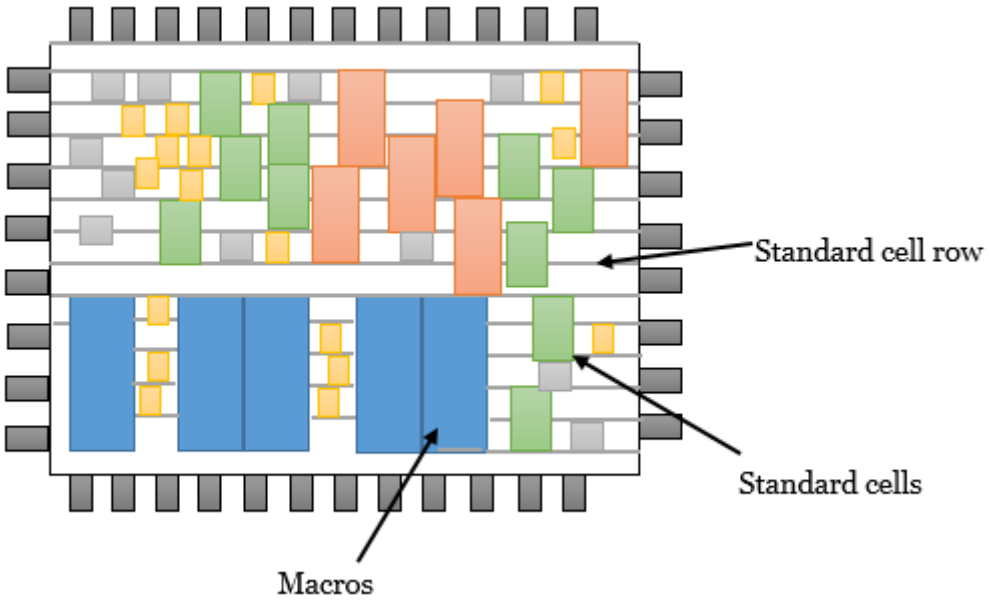

Fig. 3.3 Placement of Cells

3.7 CTS Stage

Before the Clock Tree Synthesis (CTS) stage the clock is ideal. CTS is a step in which clock is distributed to all the synchronous elements in the design. Before start CTS we need to do sanity checks that the inputs of CTS is proper or not. In CTS there are basically two steps first build a clock tree and then balance the skew of the clock tree. Quality of CTS is very important in order

to meet the timing requirements. A separate article will be done on CTS. After CTS we need to analyze the quality of the clock tree, timing and congestion.

3.8 Route Stage:

Route stage comes once the Clock tree is built and routed. In routing, there are basically two stage global routing and detail routing. Power nets and Clock nets are already routed, In this stage, we need to route the data nets.

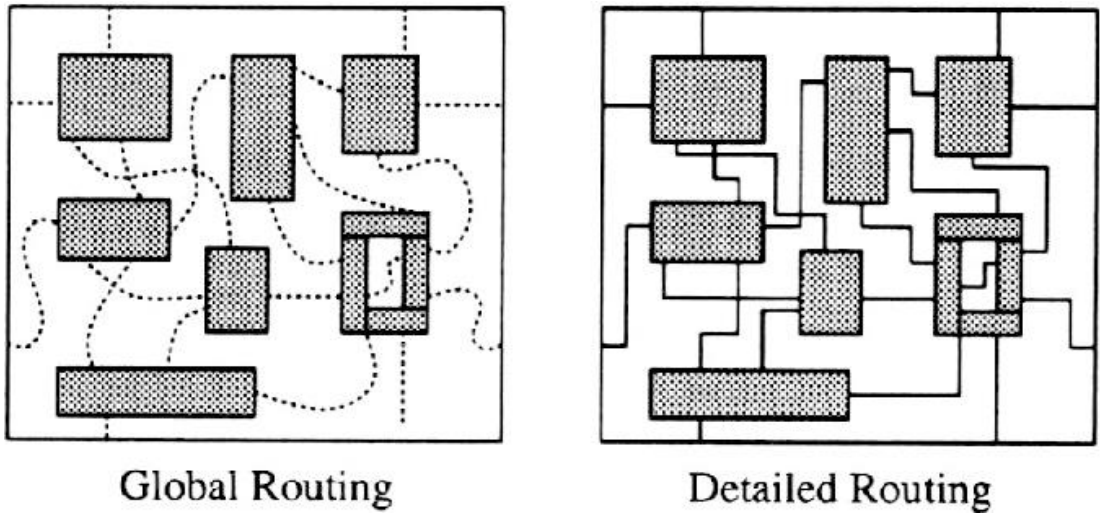

Fig. 3.4 Types of Routing

3.9 Signoff Stage

Once routing is done we need to insert fillers cells followed by metal fill and then Power signoff, timing signoff, and physical verification. Once all these steps are done in finally we stream out the layout in the form of gds or OASIS file which is called tape out. A detail discussion on each stage will be on coming articles.

Chapter 4

Layout Design

4.1 MOSFET

A Metal Oxide Semiconductor Field-effect Transistor (MOSFET, MOS-FET, or MOS FET) is a field-effect transistor (FET with an insulated gate) where the voltage determines the conductivity of the device.

It is used for switching or amplifying signals. The ability to change conductivity with the amount of applied voltage can be used for amplifying or switching electronic signals.

Structure:

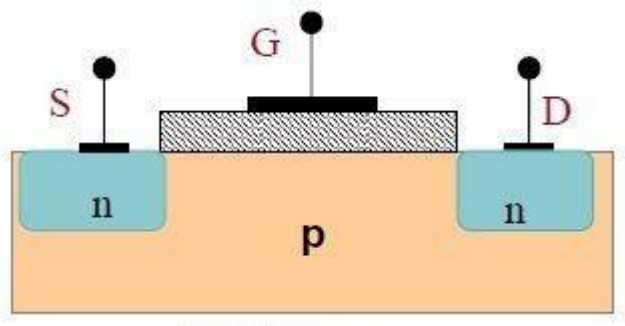

Fig. 4.1 MOSFET Structure

It is a four-terminal device with **Source (S)**, **Drain (D), Gate (G)**, and body (B) terminals. The body (B) is frequently connected to the source terminal, reducing the terminals to three. It works by varying the width of a channel along which charge carriers flow (electrons or holes).

The charge carriers enter the channel at the source and exit via the drain. The width of the channel is controlled by the voltage on an electrode called Gate which is located between the source and the drain. It is insulated from the channel near an extremely thin layer of metal oxide.

A metal-insulator-semiconductor field-effect transistor or **MISFET** is a term almost synonymous with MOSFET. Another synonym is **IGFET** for the insulated-gate field-effect transistor.

MOSFET works in two modes-

1. **Depletion Mode:** The transistor requires the Gate-Source voltage (VGS) to switch the device “OFF”. The depletion-mode MOSFET is equivalent to a “Normally Closed” switch.

2. **Enhancement Mode:** The transistor requires a Gate-Source voltage(VGS) to switch the device “ON”. The enhancement mode MOSFET is equivalent to a “Normally Open” switch. **P-Channel MOSFET**

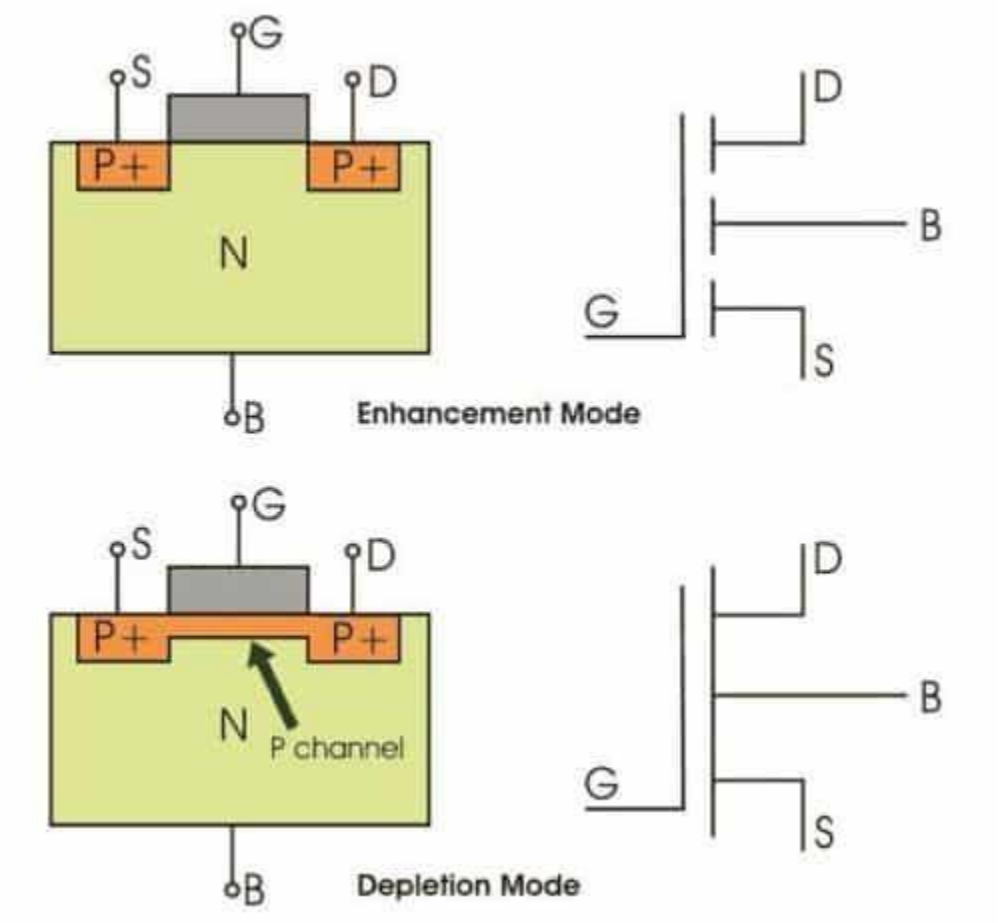

Fig. 4.2 P Channel MOSFET Depletion and Enhancement Mode

The drain and source are heavily doped p+ region and the substrate is in n-type. The current flows due to the flow of positively charged holes, and that’s why known as p-channel MOSFET.

When we apply negative gate voltage, the electrons present beneath the oxide layer experience repulsive force and are pushed downward into the substrate, the depletion region is populated by the bound positive charges which are associated with the donor atoms.

The negative gate voltage also attracts holes from the P+ source and drain region into the channel region.

N-Channel MOSFET

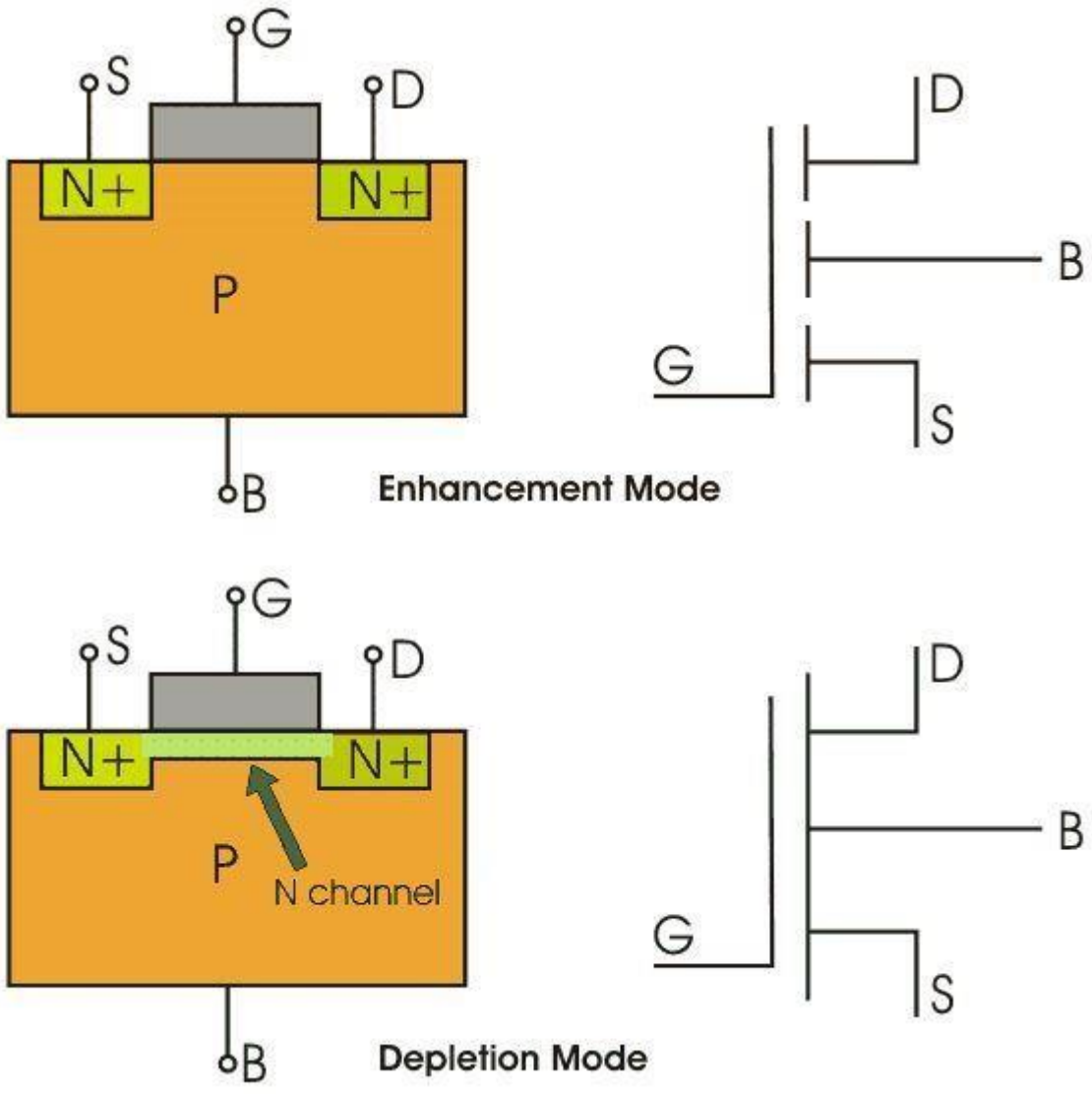

Fig. 4.3 N Channel MOSFET Depletion and Enhancement Mode

The drain and source are heavily doped N+ region and the substrate is p-type. The current flows due to the flow of negatively charged electrons and that's why known as n-channel MOSFET.

When we apply the positive gate voltage, the holes present beneath the oxide layer experience repulsive force, and the holes are pushed downwards into the bound negative charges which are associated with the acceptor atoms.

The positive gate voltage also attracts electrons from the N+ source and drain region into the channel thus an electron-rich channel is formed.

MOSFET Working Operation

The working principle of a MOSFET depends upon the MOS capacitor. The MOS capacitor is the main part of MOS-FET. The semiconductor surface at the below oxide layer is located between the source and drain terminals. It can be inverted from p-type to n-type by applying positive or negative gate voltages.

When we apply positive gate voltage, the holes present under the oxide layer experience a repulsive force, and holes are pushed downward with the substrate.

The depletion region is populated by the bound negative charges that are associated with the acceptor atoms. The electrons reach, and the channel is formed. The positive voltage also attracts electrons from the n+ source and drain regions into the channel.

Now, if a voltage is applied between the drain and source, the current flows freely between the source and drain and the gate voltage controls the electrons in the channel. If we apply negative voltage, a hole channel will be formed under the oxide layer.

MOSFET Current Equation

- **For Saturation Region**: $I_{DS} = \frac{1}{2} k_n \cdot \left(V_{gs} - V_{th}\right)^2$
- **For Linear Region**: $I_{DS} = k_n \cdot \left(V_{gs} - V_{th} - \frac{V_{DS}}{2}\right) \cdot V_{ds}$

4.2 CMOS (Complementary Metal Oxide Semiconductor)

The main advantage of CMOS over NMOS and BIPOLAR technology is the much smaller power dissipation. Unlike NMOS or BIPOLAR circuits, a Complementary MOS circuit has almost no static power dissipation. Power is only dissipated in case the circuit actually switches. This allows integrating more CMOS gates on an IC than in NMOS or bipolar technology, resulting in much better performance. Complementary Metal Oxide Semiconductor transistor consists of P-channel MOS (PMOS) and N-channel MOS (NMOS). Please refer to the link to know more about the fabrication process of CMOS transistor.

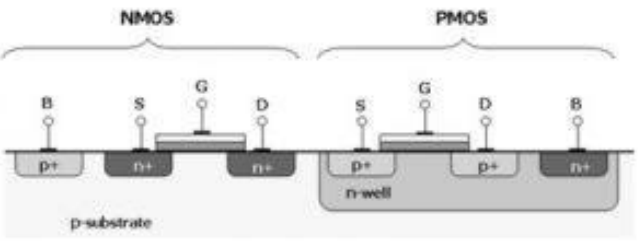

Fig 4.4 CMOS (Complementary Metal Oxide Semiconductor)

NMOS

NMOS is built on a p-type substrate with n-type source and drain diffused on it. In NMOS, the majority of carriers are electrons. When a high voltage is applied to the gate, the NMOS will conduct. Similarly, when a low voltage is applied to the gate, NMOS will not conduct. NMOS is considered to be faster than PMOS, since the carriers in NMOS, which are electrons, travel twice as fast as the holes.

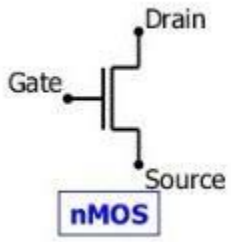

Fig 4.5 NMOS Transistor

PMOS

P- channel MOSFET consists of P-type Source and Drain diffused on an N-type substrate. The majority of carriers are holes. When a high voltage is applied to the gate, the PMOS will not conduct. When a low voltage is applied to the gate, the PMOS will conduct. The PMOS devices are more immune to noise than NMOS devices.

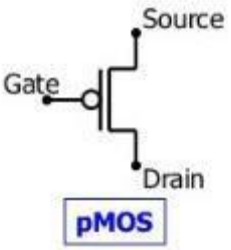

Fig 4.6 PMOS Transistor

CMOS Working Principle

In CMOS technology, both N-type and P-type transistors are used to design logic functions. The same signal which turns ON a transistor of one type is used to turn OFF a transistor of the other type. This characteristic allows the design of logic devices using only simple switches, without the need for a pull-up resistor.

In CMOS logic gates a collection of n-type MOSFETs is arranged in a pull-down network between the output and the low voltage power supply rail (Vss or quite often ground). Instead of the load resistor of NMOS logic gates, CMOS logic gates have a collection of p-type MOSFETs in a pullup network between the output and the higher-voltage rail (often named Vdd).

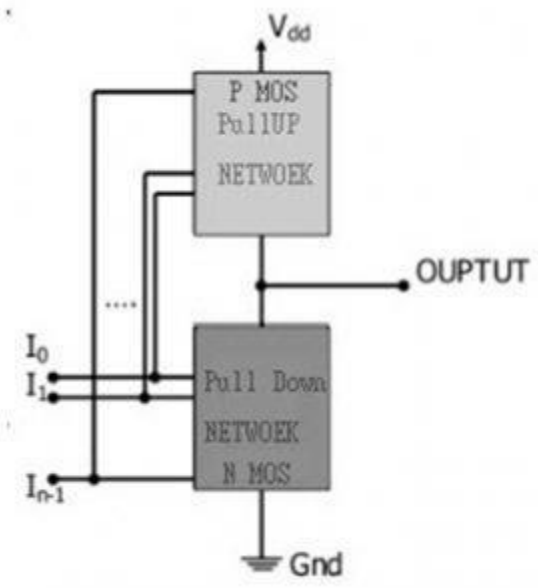

Fig 4.7 CMOS using Pull Up & Pull Down

Thus, if both a p-type and n-type transistor have their gates connected to the same input, the p-type MOSFET will be ON when the n-type MOSFET is OFF, and vice-versa. The networks are arranged such that one is ON and the other OFF for any input pattern as shown in the figure below.

CMOS offers relatively high speed, low power dissipation, high noise margins in both states, and will operate over a wide range of source and input voltages (provided the source voltage is fixed). Furthermore, for a better understanding of the Complementary Metal Oxide Semiconductor working principle, we need to discuss in brief CMOS logic gates as explained below.

CMOS Inverter

The inverter circuit as shown in the figure below. It consists of PMOS and NMOS FET. The input A serves as the gate voltage for both transistors.

The NMOS transistor has input from Vss (ground) and the PMOS transistor has input from Vdd. The terminal Y is output. When a high voltage (~ Vdd) is given at input terminal (A) of the inverter, the PMOS becomes an open circuit, and NMOS switched OFF so the output will be pulled down to Vss.

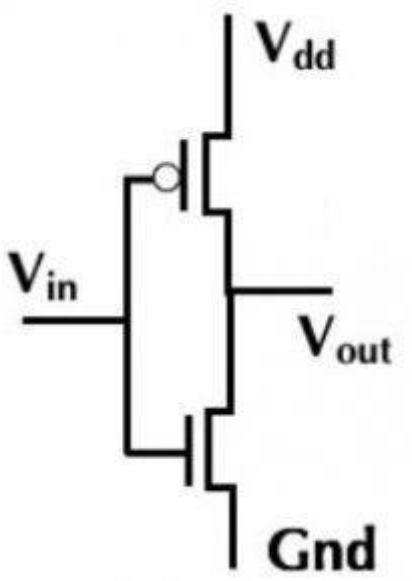

Fig 4.8 CMOS Inverter

When a low-level voltage (<Vdd, ~0v) applied to the inverter, the NMOS switched OFF and PMOS switched ON. So the output becomes Vdd or the circuit is pulled up to Vdd.

INPUT	LOGIC INPUT	OUTPUT	LOGIC OUTPUT
0 v	0	Vdd	1
Vdd	1	0 v	0

Table 4.1 CMOS Inverter Logic Gate

4.3 Stick Diagram

In VLSI (Very Large Scale Integration) design, a stick diagram is a graphical representation used to depict the layout of basic building blocks or components within an integrated circuit (IC). It is a simplified and abstract representation that helps designers to visualize the physical layout of the circuit and understand its connectivity and relative placement.

Here's how stick diagrams are typically constructed:

1. Basic Shapes: Stick diagrams use basic geometric shapes such as rectangles, squares, circles, and lines to represent various components and interconnections within the circuit.

2. Wire Connections: Lines or wires are used to represent interconnections between components. These lines indicate the routing of signals between different parts of the circuit.
3. Transistor Symbols: Transistors are represented by rectangles or squares, with lines indicating the connections to the transistor's terminals (source, drain, gate for MOSFETs).
4. Contact and Via Representation: Contacts and vias, which are used to connect metal layers or different layers within the IC, are often represented by dots or small circles at the intersections of lines.
5. Orientation and Alignment: Components and wires in stick diagrams are typically aligned in a grid-like fashion to maintain consistency and aid readability.

Stick diagrams are useful during the early stages of the IC design process because they provide a quick and intuitive way to explore different layout possibilities and understand the spatial relationships between components. However, stick diagrams are highly abstract and do not capture all the details of the physical layout, so they are usually supplemented with more detailed representations as the design progresses. These more detailed representations include layout schematics, circuit diagrams, and ultimately the full physical layout of the integrated circuit.

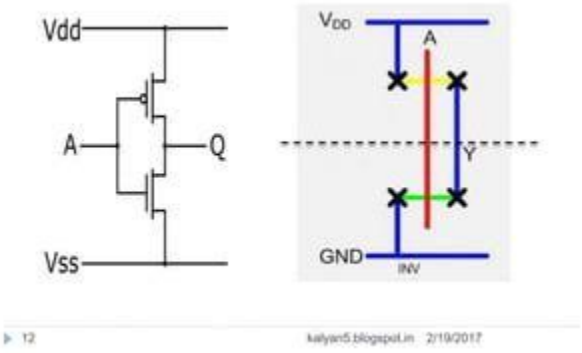

Fig 4.9 CMOS Inverter Stick Diagram

4.4 Layout Design Flow

1. **Specification and Architecture:** This initial stage involves gathering requirements from the customer or project stakeholders and defining the overall architecture of the ASIC. This includes functionality, performance targets, power constraints, and any other relevant specifications.
2. **Floorplanning:** Floorplanning involves partitioning the chip area and allocating space for different functional blocks based on their size, connectivity, and performance requirements. It also includes placement of I/O pads and power distribution network planning.

3. **Placement:** In this stage, the synthesized gates are placed onto the chip floorplan. Placement tools optimize for factors such as timing, signal integrity, and congestion.

4. **Design Rule Check (DRC):** DRC verifies that the layout adheres to the manufacturing rules specified by the foundry. DRC checks for violations such as minimum feature size, spacing, and overlap violations.
5. **Routing:** Routing involves connecting the placed gates using metal layers to implement the interconnections specified by the netlist. Routing tools optimize for factors such as signal integrity, timing, and congestion.
6. **Physical Verification:** This stage involves performing additional checks such as Antenna Rule Check (ARC), Metal Density Check, and Electrical Rule Check (ERC) to ensure the layout meets all necessary criteria for manufacturability and reliability.

Chapter 5

Cadence Software Suites

5.1 Introduction to Cadence Software

Cadence Design Systems, Inc. stands as a preeminent provider of electronic design automation (EDA) solutions, offering a comprehensive suite of software, hardware, and services crucial for modern electronic system and semiconductor device design.

5.2 Cadence Software suite

Cadence's software portfolio encompasses a spectrum of tools tailored to address diverse facets of the design lifecycle:

1. **OrCAD:** Renowned for its prowess in schematic capture, PCB design, and simulation, OrCAD furnishes an intuitive interface facilitating schematic creation, PCB layout, and essential signal integrity and electrical analysis.
2. **Allegro:** As Cadence's flagship PCB design tool, Allegro extends sophisticated functionalities catering to high-speed and high-density PCB designs. This encompasses constraint-driven design, routing, placement, and comprehensive manufacturing preparation.
3. **Virtuoso:** Virtuoso emerges as Cadence's definitive suite for custom IC design, offering a versatile platform for designing analog, digital, and mixed-signal ICs. It integrates schematic capture, layout, and simulation tools.
4. **Incisive:** Incisive represents Cadence's veritable bastion for simulation and verification, providing a robust infrastructure for functional verification, digital simulation, and hardware emulation to ascertain compliance with functional requirements and performance benchmarks.
5. **Palladium:** Palladium assumes significance as Cadence's hardware emulation platform, serving as a pivotal resource for verifying the functionality and performance of intricate SoCs prior to fabrication. Its capacity for accelerating simulation speed via specialized hardware is instrumental.
6. **Xcelium:** Catering to the rigors of high-performance digital simulation, Xcelium embodies Cadence's solution for handling complex digital designs adeptly. It encompasses features for debugging, coverage analysis, and low-power simulation.
7. **Genus:** Genus signifies Cadence's RTL synthesis tool, facilitating the translation of high-level RTL designs into optimized gate-level netlists. Its optimization algorithms enhance design performance, area efficiency, and power consumption.

8. **Innovus:** As Cadence's physical implementation tool, Innovus is indispensable for floor-planning, placement, and routing of digital designs. Leveraging advanced algorithms, it achieves timing closure and optimizes power and area considerations effectively.

5.3 Key Features and Capabilities

The notable features of Cadence Software Suites are:

1. **Advanced Automation:** Cadence software integrates sophisticated automation mechanisms to streamline design workflows and enhance productivity, encompassing automated routing, constraint-driven design methodologies, and design rule verification.
2. **Integration:** The seamless integration of Cadence tools facilitates a cohesive transition across various stages of the design process, fostering synergy and reducing errors.
3. **Performance Optimization:** Cadence tools incorporate cutting-edge algorithms tailored for performance optimization, spanning timing closure, power management, and signal integrity analysis, thereby enabling designers to adhere to stringent performance criteria and design constraints.
4. **Verification and Validation:** Cadence provides a comprehensive suite of verification and validation tools, including simulation, emulation, and formal verification techniques, to ensure designs conform to functional specifications and requirements meticulously.
5. **Collaboration and Teamwork:** Cadence software facilitates collaborative endeavors by enabling concurrent design work among multiple engineers, thereby fostering efficient collaboration and expediting design iterations.
6. **Support and Training:** Cadence extends robust support and training resources, encompassing documentation, online forums, training courses, and technical assistance, to empower engineers in leveraging their software suite optimally.

In summary, Cadence software epitomizes a cornerstone in the arsenal of electronics and semiconductor companies, offering a suite of advanced tools and features indispensable for navigating the complexities of modern electronic design with precision and efficacy.

5.4 Design of an Inverter using Cadence Virtuoso

During the attachment at Siliconova, we were introduced to Cadence Virtuoso and a simple project of designing a CMOS Inverter Layout was demonstrated.

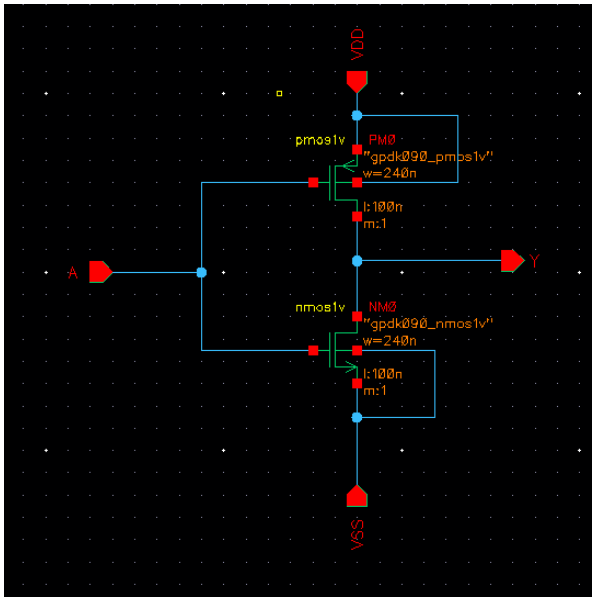

Fig. 5.1 Schematic Design of an Inverter using Cadence Virtuoso

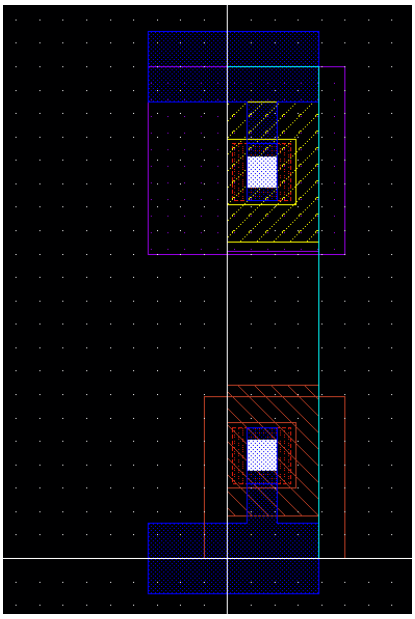

Fig. 5.2 Tap-Cell Design of an Inverter

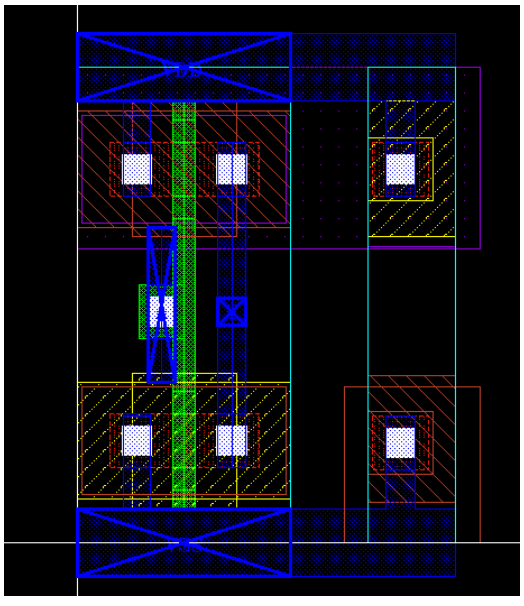

Fig. 5.3 Layout Design of an Inverter

Chapter 6

Introduction to Verilog

6.1 HARDWARE DESCRIPTION LANGUAGE (HDL)

It is a language used for describing a digital system like a network switch or a microprocessor or a memory or a flip−flop. It means, by using a HDL we can describe any digital hardware at any level. Designs, which are described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits.

SL.No.	Hardware Description Language	Software Language
01	HDL defines the structure and behavior of electronic circuits and mostly, digital logic circuits.	Software language writes a set of instructions to allow a CPU to perform a particular task.
02	It defines the behavior of digital circuits.	It helps to develop a variety of applications.
03	It is more complex to work with.	It is not as complex to work with.
04	This design is based on the creation and use of textual-based descriptions of circuits.	This is used to create executable software applications that will operate on a suitable processor.
05	It is a language which is having syntactic and semantic support for supporting the temporal behavior and spatial structure of hardware.	It is a language that can translate machine instructions and execute them on a computer.
06	There is no such facility for language selection in HDL.	Depending on the application, a programmer can choose a language. You have a lot of options when it comes to software languages.
07	The career prospects with HDL-Hardware Engineer, Electronic	Working as a web developer, web designer, data science specialist, QA
08	Hardware Engineer, or Embedded Engineer and other positions.	Manager and other positions are all possible with software languages.
09	Examples: Verilog and VHDL.	Examples: Java, C, C++, etc.

Table 6.1 Difference between HDL and Software Language

6.2 RTL Coding

In digital circuit design, RTL coding is a design abstraction that represents a synchronous digital circuit's behavior in terms of:

- **Data flow** between hardware registers
- **Logical operations** performed on that data

As it provides a high-level description of a circuit, designers are free to concentrate on usefulness rather than being mired down in minutiae such as transistor-level implementation.

6.3 Key Concepts in RTL Verilog:

Verilog is a hardware description language (HDL) used for designing and modeling digital circuits. It's widely used in the field of digital design for describing electronic systems, such as integrated circuits and FPGA-based logic designs. Here's a brief description:

1. **Conciseness:** Verilog allows designers to succinctly describe complex digital circuits using a concise syntax. This makes it easier to understand and maintain large designs.

2. **Behavioral and Structural Constructs:** Verilog supports both behavioral and structural modeling. Behavioral modeling describes the functionality of the circuit, while structural modeling describes the interconnection between hardware components.

3. **Modules:** Verilog designs are organized into modules, which are encapsulated units of functionality. Modules can be instantiated multiple times and connected together to build larger systems.

4. **Hierarchical Design:** Verilog supports hierarchical design, allowing designers to break down complex systems into smaller, more manageable modules. This promotes modularity and reusability.

5. **Simulation and Synthesis:** Verilog can be used for simulation and synthesis. Simulation allows designers to verify the correctness of their designs through software-based testing, while synthesis translates the Verilog code into a netlist of gates that can be implemented in hardware.

6. **Event-Driven Simulation:** Verilog simulators use an event-driven model, where changes in signal values trigger events that propagate through the circuit. This allows for efficient simulation of digital systems.

7. **Parallel Execution:** Verilog is inherently parallel, allowing multiple operations to occur simultaneously. This makes it suitable for describing parallel hardware architectures.

8. **Data Types:** Verilog supports a variety of data types, including scalar and vector types, as well as user-defined types. This allows designers to represent different types of signals and data in their designs.

Overall, Verilog provides a powerful and flexible platform for digital design, enabling designers to efficiently create and verify complex digital systems.

6.4 Benefits of RTL Coding:

- **Abstraction:** Enables designers to work at a higher level, making circuits more manageable and reusable.
- **Synthesis:** RTL code can be fed into logic synthesis tools, which automatically convert it into a netlist of lower-level gates or FPGA cells for implementation in hardware.
- **Simulation:** RTL code can be simulated using hardware description language (HDL) simulators to verify its functionality before synthesis.

6.5 Introduction to System Verilog

System Verilog is an extension of the Verilog hardware description language (HDL) with additional features aimed at improving verification productivity, design clarity, and reusability. It encompasses both design and verification aspects, making it a comprehensive language for digital system modeling.

Here's a detailed description of System Verilog:

i) **Enhanced Data Types:**
- System Verilog introduces enhanced data types such as dynamic arrays, associative arrays, queues, and user-defined types (typedef).
- Dynamic arrays allow for flexible memory allocation during simulation, making it easier to work with variable-sized data structures.
- Associative arrays (also known as hash tables) provide efficient lookup and storage based on user-defined keys.
- Queues are specialized arrays optimized for FIFO (First-In-First-Out) data storage and manipulation.

ii) Object-Oriented Programming (OOP):
- System Verilog supports object-oriented programming (OOP) concepts such as classes, objects, inheritance, and polymorphism.
- Classes allow designers to encapsulate data and behaviors into reusable units, promoting modularity and reusability.
- Inheritance enables the creation of derived classes that inherit properties and behaviors from base classes, facilitating code reuse and hierarchy.
- Polymorphism allows objects of different types to be treated uniformly, enhancing flexibility and extensibility.

iii) Constrained Randomization:

- System Verilog introduces constrained randomization, a powerful verification methodology for generating realistic and diverse stimulus.
- Constraints define the properties and relationships of random variables, guiding the randomization process to produce valid test scenarios.
- Randomization facilitates comprehensive testing by exploring different corner cases and scenarios that may not be covered by directed testing alone.

iv) **Assertion-Based Verification (ABV):**

- System Verilog includes built-in support for assertion-based verification (ABV), enabling designers to specify properties and constraints directly in the code.
- Assertions express design requirements, constraints, and expected behavior, allowing for automated formal verification and runtime checking during simulation.
- ABV enhances design confidence by detecting errors and violations early in the verification process, reducing debug time and improving overall verification quality.

v) **Concurrency and Parallelism:**

- System Verilog introduces constructs for describing concurrent and parallel behavior, including fork-join blocks, parallel blocks, and the fork-join none construct.
- Concurrent blocks allow for the concurrent execution of procedural statements, improving simulation efficiency and modeling complex behavior.
- Parallel blocks enable the parallel execution of multiple statements or processes, facilitating the modeling of parallel hardware architectures and algorithms.

vi) Direct Programming Interface (DPI):

- System Verilog provides a direct programming interface (DPI) for interfacing with foreign languages such as C/C++.
- DPI enables seamless integration of Verilog/System Verilog code with software components, allowing for co-simulation, hardware-software co-verification, and system-level modeling.
- DPI facilitates design verification by enabling the reuse of existing software libraries and leveraging the performance advantages of hardware acceleration.

Overall, System Verilog enhances the capabilities of traditional Verilog by incorporating advanced features for design, verification, and system-level modeling. It provides a unified platform for hardware and verification engineers to develop complex digital systems efficiently and reliably.

6.6 OOP Concepts in System Verilog Coding

In System Verilog, Object-Oriented Programming (OOP) concepts are integrated into the language to enhance design and verification capabilities. Here's a detailed description of how OOP concepts are implemented in System Verilog:

1. Classes and Objects:

- System Verilog supports the definition of classes, which are user-defined data types that encapsulate both data (attributes) and methods (functions or tasks).

- Objects are instances of classes, representing individual entities with their own unique set of data and behavior.
- Classes in System Verilog can have constructors and destructors, allowing initialization and cleanup operations when objects are created and destroyed.

2.Encapsulation:

- Encapsulation in System Verilog involves bundling data and methods together within a class, while restricting access to the internal implementation details from outside the class.
- Access specifiers (public, protected, private) control the visibility of class members, allowing designers to enforce encapsulation and data hiding.
- Encapsulation promotes modularity, reusability, and maintainability by isolating implementation details and exposing only the necessary interfaces.

3.Inheritance:

- Inheritance allows a class (subclass or derived class) to inherit properties and methods from another class (superclass or base class).
- System Verilog supports single and multiple inheritance, where a subclass can inherit from one or more base classes.
- Derived classes can extend or override inherited methods, providing flexibility in customizing behavior while leveraging existing functionality.

4.Polymorphism:

- Polymorphism enables objects of different types to be treated uniformly through a common interface, allowing for code reuse and flexibility.
- System Verilog supports polymorphism through virtual methods and dynamic dispatch.
- Virtual methods declared in base classes can be overridden in derived classes, and the appropriate method is dynamically dispatched based on the runtime type of the object.

5.Abstract Classes and Interfaces:

- System Verilog allows the declaration of abstract classes and interfaces, which cannot be instantiated directly but serve as templates for concrete implementations.
- Abstract classes define a common interface and may contain one or more pure virtual methods (methods without an implementation).
- Interfaces define a set of methods without any implementation details, providing a contract for interacting with objects.

6.Static and Dynamic Arrays:

- System Verilog supports static arrays (arrays with fixed size) and dynamic arrays (arrays with variable size), which can be used to represent collections of data within classes.
- Arrays can be used to model data structures such as queues, stacks, and matrices, facilitating complex data manipulation and storage.

By incorporating OOP concepts, System Verilog promotes code organization, reusability, and scalability in both design and verification domains. Designers can leverage classes, inheritance, encapsulation, and polymorphism to create modular, extensible, and maintainable codebases for complex digital systems.

6.7 Comparative Analysis on Verilog and System Verilog

A comparative analysis on Verilog and System Verilog is given as follows:

SL No.	**Verilog**	**System Verilog**
1.	Verilog is a Hardware Description Language (HDL).	System Verilog is a combination of both Hardware Description Language (HDL) and Hardware Verification Language (HVL).
2.	Verilog language is used to structure and model electronic systems.	System Verilog language is used to model, design, simulate, test and implement electronic system.
3.	It supports structured paradigm.	It supports structured and object oriented paradigm.
4.	Verilog is based on module level testbench.	System Verilog is based on class level testbench.
5.	It is standardized as IEEE 1364.	It is standardized as IEEE 1800-2012.
6.	Verilog is influenced by C language and Fortran programming language.	System Verilog is based on Verilog, VHDL and c++ programming language.
7.	It has file extension .v or .vh	It has file extension .sv or .svh
8.	It supports Wire and Reg datatype.	It supports various datatypes like enum, union, struct, string, class.
9.	It is based on hierarchy of modules.	It is based on classes.
10.	It was begun in 1983 as proprietary language for hardware modelling.	It was originally intended as an extension to Verilog in the year 2005.

Table 6.2 Difference between Verilog and System Verilog

6.8 Rules of Coding in Verilog RTL

Verilog's detailed rules encompass various aspects, here's a breakdown of the major ones:

1. Lexical Structure:

- **Case Sensitivity:** Verilog is case-sensitive. reg and REG are entirely different.
- **Keywords:** Keywords are reserved words with specific meanings and are always lowercase (e.g., module, always, if).
- **Identifiers:** User-defined names for variables, signals, and modules must start with a letter underscore (_) and can contain letters, numbers, and underscores.
- **Comments:** Single-line comments begin with // and extend to the end of the line. Multi-line comments use /* and */.
- **Operators:** Verilog supports various operators for arithmetic, logical, comparison, bitwise, and reduction operations.
- **Separators:** Whitespace (spaces, tabs, newlines) separates tokens except within strings.

2. Data Types:

- **Nets (wires):** Represent physical wires connecting modules. Declared with wire. Values propagate immediately upon change. Useful for combinational logic.
- **Registers:** Hold data and update on clock edges. Declared with reg. Useful for sequential logic.
- **Integers:** Signed (int) or unsigned (integer) for whole numbers. Can have a specific width (int 4).
- **Real Numbers:** Represented with real for floating-point numbers.
- **Logical:** Represent Boolean values (logic, reg). Can be 1, 0, X (unknown), or Z (high impedance).
- **Arrays:** Collection of elements of the same type. Defined with square brackets [size-1:0].

3. Modules:

- **Structure:** The basic building block of Verilog design. Defined with module <name> (...).
- **Ports:** Module interface for communication. Declared with input, output, or inout and data type.
- **Statements:** Verilog code resides within modules and defines functionality.

4. Statements:

- **Blocking Assignments (=):** Assigns a value to a variable on the current time step. Useful for combinational logic.
- **Non-Blocking Assignments (<=):** Schedules a value to be assigned to a variable at the next time step. Used for sequential logic with clock edges.
- **Conditional Statements (if, else):** Control program flow based on conditions.
- **Case Statements:** Multi-way branching based on compared values.
- **Loops (for, while):** Execute statements repeatedly.
- **Procedural Blocks (always):** Contain concurrent statements that execute continuously. Used to model sequential behavior with clock edges.

5. Other Rules:

- **Signal Initialization:** Signals must be initialized with valid values (0, 1, X, or Z) during module instantiation.
- **Concurrency:** Verilog allows for concurrent execution of statements within an always block.
- **Time Constructs:** Verilog offers constructs like # (delay) and forever (infinite loop) for timing behavior.
- **System Verilog Extensions:** While not core Verilog, System Verilog adds features for advanced design and verification.

6.9 Verilog Ports

Port is an essential component of the Verilog module. Ports are used to communicate for a module with the external world through input and output.

It communicates with the chip through its pins because of a module as a fabricated chip placed on a PCB.

Every port in the port list must be declared as ***input, output*** or ***input-output***. All ports declared as one of them is assumed to be wire by default to declare it, or else it is necessary to declare it again. Ports, also referred to as pins or terminals, are used when wiring the module to other modules.

The rules for naming identifiers are as follows −

- Identifier names are unique.
- Cannot use a keyword as identifiers.
- Identifier has to begin with a letter or underscore (_).
- It should not contain white space.
- Special characters are not allowed.
- Identifiers can consist of only letters, digits, or underscore.
- Only 31 characters are significant.
- They are case sensitiv

The data storage and transmission elements found in digital hardware are represented using a set of Verilog Hardware Description Language (HDL) data types. The purpose of Verilog HDL is to design digital hardware.

Data types in Verilog are divided into **NETS** and **Registers**. These data types differ in the way that they are assigned and hold values, and also they represent different hardware structures.

Number Specification:

Sized Numbers:

Size --> no. of bits in the number; only decimal
Base format --> for decimal : ‘d or ‘D
Hex :’h or ‘H
Binary : ‘b or ‘B
Octal : ‘o or ‘O

Number -> 0 to 9(if deci)
0 to F(if hex)
0 to 8(if oct)
Example:
4‘b1011 //4 bit binary no.
12’habf //12 bit hex no.
16’d255 //16 bit decimal no.

Unsized Numbers:

Numbers that are specified without are decimal numbers by default.

Example:

23456 //32 bit decimal no. by default
‘hc3 //32 bit hex no. by default
‘o21 //32 bit oct no. by default

X or Z values:

X ---> denotes unknown value

Z ---> denotes a high impedance value

Example:

12’h13X //12 bit hex no.;4 LSBs are unknown
6’hx //6 bit hex no.;all 6 bits are unknown
32’bz //32 bit high impedance number

Negative Numbers:

Example:

-6’d3 //8 bit negative no. stored as 2’s complement of 3
-6sd3 //used for performing signed integer math
4’d-2 //illegal specification: ‘-’ is not allowed between and

6.10 Rules of Coding in System Verilog

System Verilog builds upon the foundation of Verilog, adding features for enhanced design, verification, and reusability. Here's a breakdown of some key detailed rules specific to System Verilog:

1. Classes and Inheritance:

- System Verilog introduces object-oriented concepts like classes for data structures and modules.
- Classes can inherit properties and functionality from parent classes, promoting code reuse.

2. Interfaces:

- Interfaces define a set of methods (functions) that a class must implement, promoting modularity and verification.

3. Packages:

- Organize code into hierarchical packages for better structure and namespace management.

4. Constrains:

- System Verilog allows expressing design constraints using constrain and assert statements for formal verification.

5. Randomization:

- Powerful randomization features enable generation of random test data for verification.

6. Functional Coverage:

- System Verilog supports specifying and monitoring functional coverage to ensure test completeness.

7. Verification Methodologies:

- Supports methodologies like Universal Verification Methodology (UVM) for efficient and scalable verification.

8. Advanced Data Types:

- Extends data types including enums, structs, and unions for complex data modeling.

9. Concurrent Assertions:

- Enables expressing assertions (checks) that can run concurrently with the design for comprehensive verification.

10. Interfaces with Other Languages:

- System Verilog allows interfacing with languages like C and C++ for integration with other verification tools.

Important Considerations:

- System Verilog syntax largely overlaps with Verilog, but some constructs have different interpretations.
- Not all tools fully support all System Verilog features. Check tool documentation for compatibility.

In digital circuit design, RTL coding is a design abstraction that represents a synchronous digital circuit's behavior in terms of:

- **Data flow** between hardware registers
- **Logical operations** performed on that data

As it provides a high-level description of a circuit, designers are free to concentrate on usefulness rather than being mired down in minutiae such as transistor-level implementation.

Verilog supports 4 levels of abstraction namely,

1. **Switch Level:** This is the lowest level of abstraction in Verilog, where individual transistors and switches are explicitly modeled. Switch-level modeling is rarely used directly by designers due to its complexity and the level of detail required. It's typically used in specialized cases where very detailed analysis or design is necessary, such as for analog or mixed-signal circuits.
2. **Structural Level/Gate Level:** At this level of abstraction, hardware is described using interconnected modules and components. Each module represents a specific hardware component, and the interconnections between modules represent the connections between components in the actual hardware. Structural-level Verilog is commonly used for hierarchical design, where larger systems are built by connecting smaller modules together.
3. **Data Flow Level:** Data flow modeling describes the behavior of hardware in terms of the flow of data through the system. In Verilog, data flow modeling is done using continuous assignments, where the value of signals is updated continuously based on the values of other signals. Data flow modeling is particularly useful for describing combinational logic circuits, where the output depends only on the current values of the inputs.
4. **Behavioral Level:** This is the highest level of abstraction in Verilog, where hardware behavior is described in terms of algorithms and procedures. Behavioral-level modeling focuses on specifying what the hardware should do, rather than how it should be

implemented. It uses constructs like procedural blocks (always, initial) and functions to describe the behavior of the hardware. Behavioral-level modeling is often used during the early stages of design for system level modeling and verification.

Each level of abstraction in Verilog has its own strengths and weaknesses, and the choice of level depends on factors such as design complexity, design goals, and the stage of the design process. Designers often use a combination of abstraction levels to model different parts of the system at the appropriate level of detail.

Most Important Verilog Keywords:

1. **module**: A building block in Verilog that defines a hardware module. It is always terminated with **endmodule**. The module keyword is followed by the circuit name and a port list, where each port can be either an input or an output.
2. **assign**: Creates combinational logic. It assigns a value to a wire or register.
3. **case**: Defines conditional branching based on the value of an expression.
4. **while**: Defines loop constructs.
5. **wire** and **reg**: These are data types in Verilog. wire represents a continuous assignment, while reg represents a register.
6. Logical operators: Keywords like **and**, **or**, and **nand** are used for logical operations.
7. System tasks and functions: Verilog includes built-in system tasks and functions that perform specific actions, such as $display, $monitor, and $finish.

Chapter 7

Verilog Code and Tasks

7.1 EDA Playground

EDA Playground is a free web application that allows users to edit, simulate (and view waveforms), synthesize, and share their HDL code. Its goal is to accelerate the learning of design and testbench development with easier code sharing and with simpler access to simulators and libraries.

Currently, EDA Playground runs open source and/or free simulation and synthesis EDA tools. It supports several HDLs such as SystemVerilog, VHDL, MyHDL, and Migen. Verification engineers may play with several libraries and methodologies, such as UVM, OVL, SVUnit, and cocotb. UVM is currently one of the most popular verification methodologies. Here is a simple Verilog design and UVM testbench. For synthesis, EDA Playground uses the open source Yosys and VTR flows.

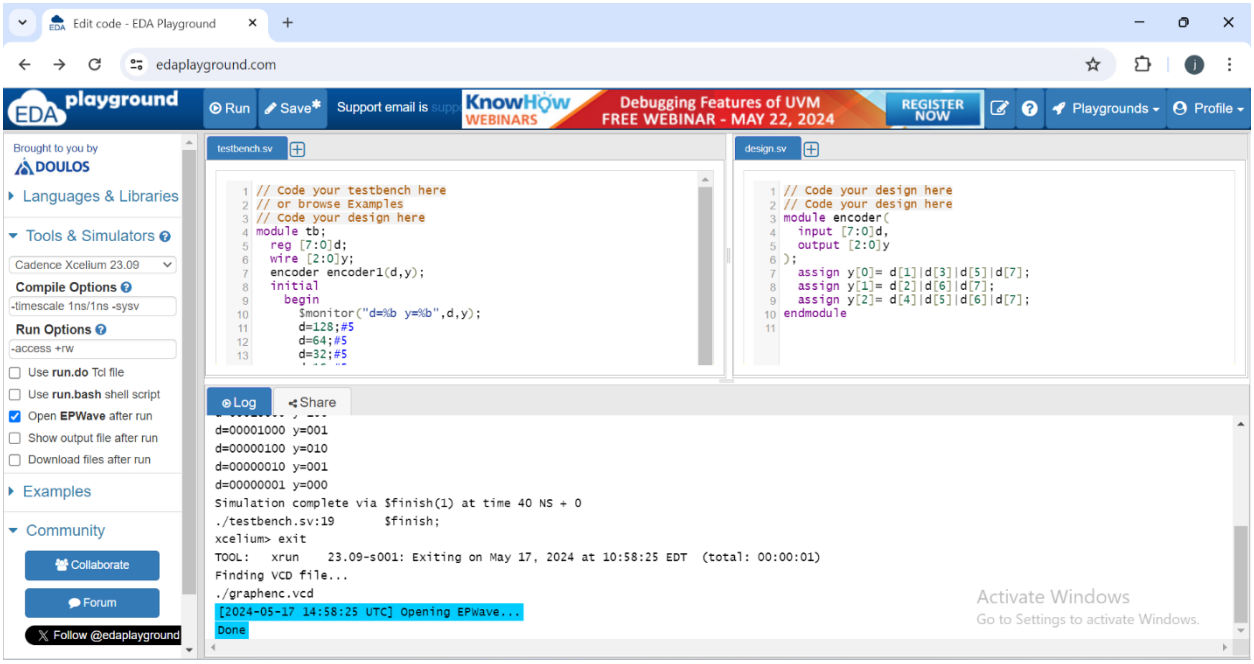

Fig.7.1 EDA Playground Website for Verilog and System Verilog

7.2 Combinational Circuits

2:1 MUX:

Verilog Design:

```
// Code your design here
module mux2x1(i0,i1,s,y);
 input i0,i1,s;
 output y;

 assign y=s?i1:i0; //s=1(i1) s=0(i0)

endmodule
```

Test Bench:

```
// Code your testbench here
// or browse Examples
module testbench();
 reg i0,i1,s;
 wire y;

 mux2x1 dut(i0,i1,s,y);
 initial begin
  repeat(5) begin
   {i0,i1,s}=$random; //random
   #2ns;
  end
 end
 initial begin
  $monitor("i0=%b i1=%b s=%b y=%b", i0,i1,s,y);
 end
 initial begin
```

```
    $dumpfile("graphmux2x1.vcd");
    $dumpvars;
  end
endmodule
```

Waveform:

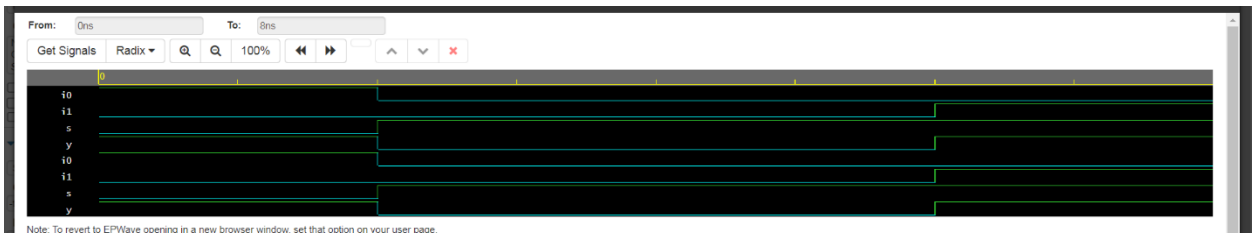

Fig.7.2 Waveform of 2:1 Multiplexer using Verilog Code

4:1 MUX using 2:1 MUX:

Verilog Design:

```
// Code your design here
module mux_4to1(f,i,s);
  input [3:0]i;
  input [1:0]s;
  output f;
  wire t1,t2;
  mux_2to1 m1(t1,i[0],i[1],s[0]);
  mux_2to1 m2(t2,i[2],i[3],s[0]);
  mux_2to1 m3(f,t1,t2,s[1]);
endmodule

module mux_2to1(z,x,y,s);
  input x,y;
  input s;
  output reg z;
  always @(*)
    if(s==0)
```

```
    z=x;
  else
    z=y;
  endmodule
```

Test Bench:

```
// Code your testbench here
// or browse Examples
module tb();
  reg[3:0]i;
  reg[1:0]s;
  wire f;
  mux_4to1 dut(f,i,s);
  initial
    begin
      s=0;i=6;//6=4'b0110
      #1 s=1;
      #1 s=2;
      #1 s=3;
      #1 $finish;
    end
  initial begin
    $dumpfile("graphmux4x1.vcd");
    $dumpvars;
  end
endmodule
```

Waveform:

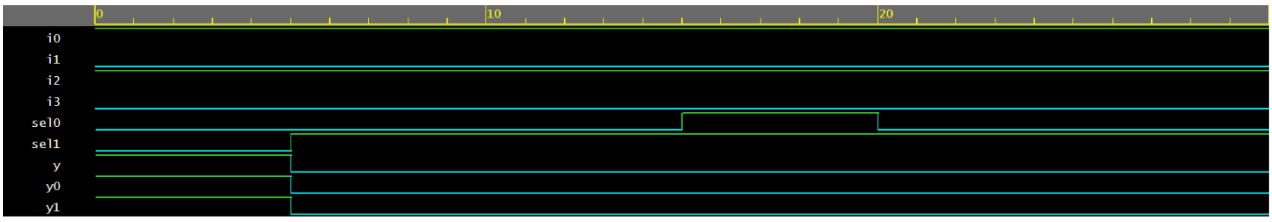

Fig.7.3 Waveform of 4:1 Multiplexer using 2:1 Multiplexer using Verilog Code

Encoder:

Verilog Design:

```
// Code your design here
module encoder(
  input [7:0]d,
  output [2:0]y
);
  assign y[0]= d[1]|d[3]|d[5]|d[7];
  assign y[1]= d[2]|d[6]|d[7];
  assign y[2]= d[4]|d[5]|d[6]|d[7];
endmodule
```

Test Bench:

```
// Code your testbench here
// or browse Examples
module tb;
  reg [7:0]d;
  wire [2:0]y;
  encoder encoder1(d,y);
  initial
    begin
      $monitor("d=%b y=%b",d,y);
      d=128;#5
      d=64;#5
      d=32;#5
      d=16;#5
      d=8;#5
      d=4;#5
      d=2;#5
      d=1;#5
      $finish;
```

```
  end
 initial begin
  $dumpfile("graphenc.vcd");
  $dumpvars;
 end
endmodule
```

Waveform:

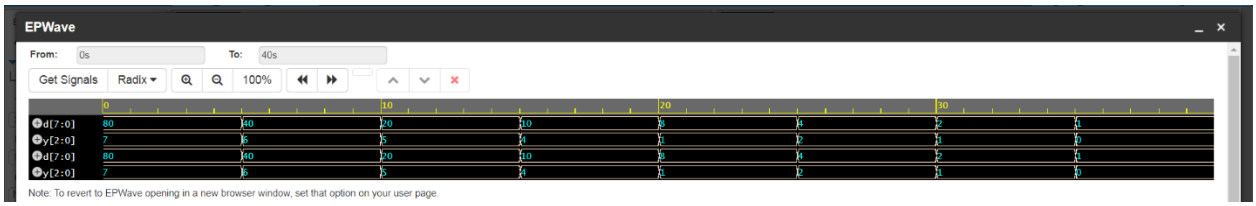

Fig.7.4 Waveform of Encoder using Verilog Code

Priority Encoder:

Verilog Design:

```
//Code design begins here

module priencoder(
 input[3:0]a,
 output[1:0]y
);
 assign y[0]=a[3]|a[2];
 assign y[1]=a[3]|(~a[2])&a[1];
endmodule
```

Test Bench:

```
// Code your testbench here
// or browse Examples
module tb;
 reg [3:0]a;
 wire [1:0]y;
```

```
 priencoder priencoder1(a,y);
 initial
  begin
   $monitor("a=%b, y=%b",a,y);
   a=4'b0000;#1
   a=4'b0001;#1
   a=4'b0010;#1
   a=4'b0100;#1
   a=4'b1000;#1
   $finish;
  end
 initial begin
  $dumpfile("graphmux2x1.vcd");
  $dumpvars;
 end
endmodule
```

Waveform:

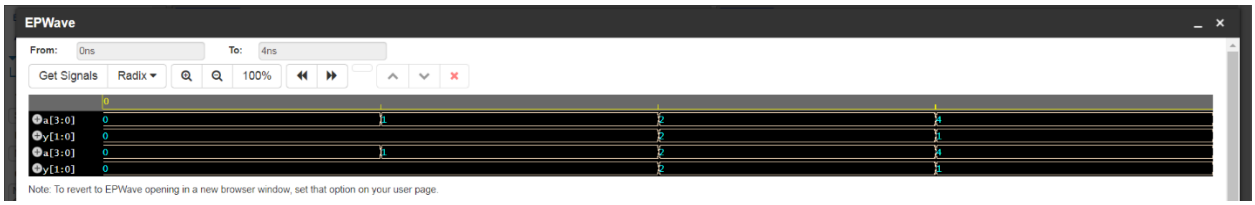

Fig.7.5 Waveform of Priority Encoder using Verilog Code

Full Adder:

Verilog Design:

```
// Code your design here
module fulladder(
 input a,b,c,
```

```
  output sum,carry
);
  assign sum=a^b^c;
  assign carry=(a*b)+(b*c)+(c*a);
endmodule
```

Test Bench:

```
// Code your testbench here
// or browse Examples
module tb;
  reg a,b,c;
  wire sum,carry;
  fulladder fulladder1(a,b,c,sum,carry);
  initial
    begin
      $dumpfile("file.vcd");
      $dumpvars(1);
      a=0;b=0;c=0;
      #5
      a=0;b=0;c=1;
      #5
      a=0;b=1;c=0;
      #5
      a=0;b=1;c=1;
      #5
      a=1;b=0;c=0;
      #5
      a=1;b=0;c=1;
      #5
      a=1;b=1;c=0;
      #5
      a=1;b=1;c=1;
      #5
```

```
    $finish;
  end
endmodule
```

Waveform:

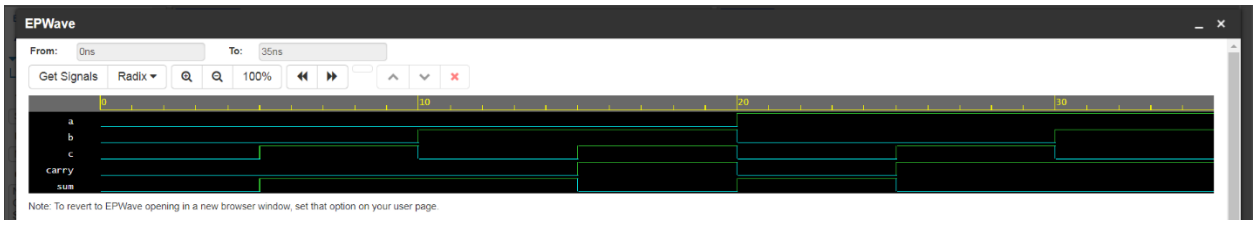

Fig.7.6 Waveform of Full Adder using Verilog Code

BCD 7 Segment:

Verilog Design:

```
// Code your design here
module seven(
 input [3:0]bcd,
 output reg [6:0]seg
);
 always@(bcd) begin
  case(bcd)
   0:seg=7'b1111110;
   1:seg=7'b0110000;
   2:seg=7'b1101101;
   3:seg=7'b1111001;
   4:seg=7'b0110011;
   5:seg=7'b1011011;
   6:seg=7'b1011111;
   7:seg=7'b11100000;
   8:seg=7'b11111111;
   9:seg=7'b11111011;
   default:seg=7'b11111111;
  endcase
```

```
  end
endmodule
Test Bench:
// Code your testbench here
// or browse Examples
module tb;
  reg [3:0]bcd;
  wire [6:0]seg;
  integer i;
  seven seven1(bcd,seg);
  initial
    begin
      $monitor("bcd=%b, seg=%b",bcd,seg);
      for (i=0;i<16;i=i+1)begin
        bcd=i;
      #10;
    end
  end
  initial begin
    $dumpfile("graphbcd7.vcd");
    $dumpvars;
  end
endmodule
```

Waveform:

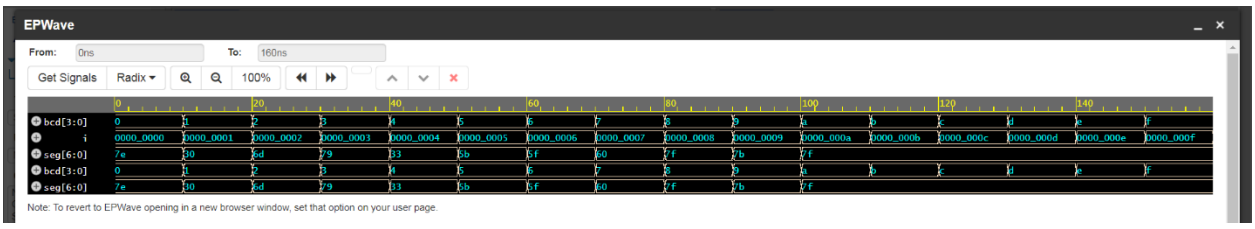

Fig.7.7 Waveform of BCD to 7 Segment using Verilog Code

7.3 Sequential Circuits:

D Flip Flop with Enable, Reset:

Verilog Design:

```
// Code your design here
module dff(
  input rst,clk,d,
  output reg y
);
  always@(posedge clk)begin
    if(rst==1)begin
      y=0;
    end
    else begin
      y<=d;
    end
  end
endmodule
```

Test Bench:

```
// Code your testbench here
// or browse Examples
module tb;
  reg rst,clk,d;
  wire y;
  dff dff1(rst,clk,d,y);
  initial
    begin
      $dumpfile("file.vcd");
      $dumpvars(1);
```

```
    clk=1;
    forever #10 clk=~clk;
  end
 initial
  begin
    rst=1;d=0;#10
    rst=1;d=1;#10
    rst=0;d=0;#10
    rst=0;d=0;#10
    rst=0;d=1;#10
    rst=0;d=1;#10
    $finish;
  end
endmodule
```

Waveform:

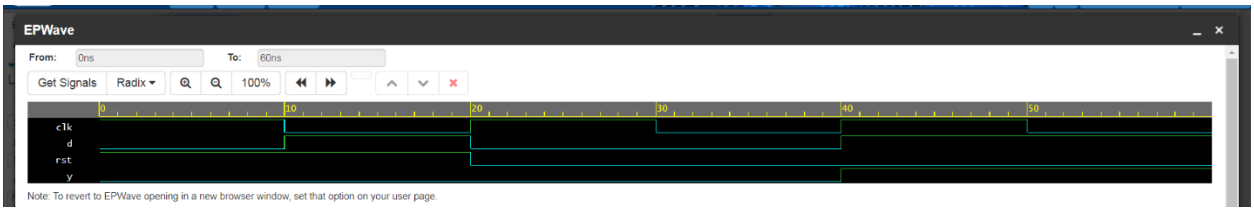

Fig.7.8 Waveform of D Flip-Flop with Enable and Reset using Verilog Code

JK Flip Flop:

Verilog Design:

```
// Code your design here
module jkflip(
 input rst,clk,j,k,
 output reg q,qbar
);
 always@(posedge clk)begin
  if(rst==1)begin
    q=0;
```

```
    qbar=~q;
   end
   else
    if(j==0 && k==0)begin
     q<=q;
     qbar<=qbar;
    end
   else if(j==0 && k==1)begin
    q<=0;
    qbar<=~q;
   end
   else if(j==1 && k==0)begin
    q<=1;
    qbar<=~q;
   end
   else begin
    q<=~q;
    qbar<=~qbar;
   end
  end
endmodule
```

Test Bench:

```
// Code your testbench here
// or browse Examples
module tb;
 reg rst,clk,j,k;
 wire q,qbar;
 jkflip jkflip1(rst,clk,j,k,q,qbar);
 initial
  begin
   $dumpfile("file.vcd");
```

```
    $dumpvars(1);
    clk=1;
    forever #10 clk=~clk;
  end
 initial
  begin
    rst=1;
    j=0;k=0;#5
    rst=0;
    j=0;k=0;#5
    j=0;k=0;#5
    j=0;k=1;#5
    j=0;k=1;#5
    j=1;k=1;#5
    j=1;k=1;#5
    j=1;k=1;#5
    j=1;k=1;#5
    $finish;
  end
endmodule
```

Waveform:

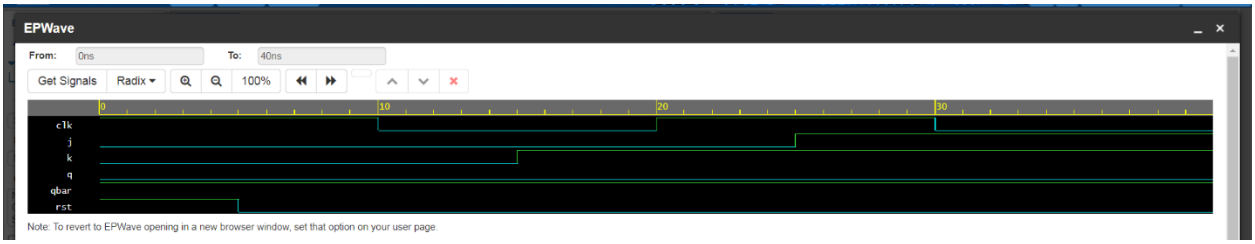

Fig.7.9 Waveform of JK Flip-Flop using Verilog Code

Ring Counter:

Verilog Design:

```
// Code your design here
module ring(clk,rst,count);
```

```
 input clk,rst;
 output reg[3:0] count =4'b0000;

 always @(posedge clk) begin
  if(rst==1)
   count<=4'b1000;
  else begin
   count[3]<=count[0];
   count[2]<=count[3];
   count[1]<=count[2];
   count[0]<=count[1];
  end
 end
endmodule
```

Test Bench:

```
// Code your testbench here
// or browse Examples
module ring_tb();
 reg clk,rst;
 wire [3:0] count;

 ring dut (clk,rst,count);

 initial begin
  clk=0;
  forever #10 clk=~clk;
 end
  initial begin
   $dumpfile("file.vcd");
   $dumpvars(0,dut);
   $monitor("Time=%0t clk=%b rst=%b count=%b",$time,clk,rst,count);
```

```
    #0 rst =1;
    #30 rst =0;

    repeat(20) begin
      @(posedge clk);
    end
    $finish();

  end
endmodule
```

Waveform:

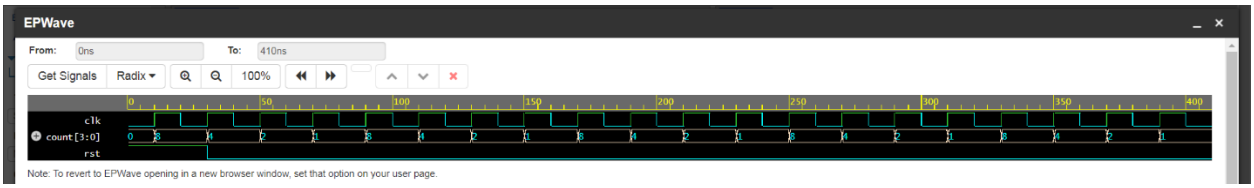

Fig.7.10 Waveform of Ring Counter using Verilog Code

Johnson Counter:

Verilog Design:

```
// Code your design here
module johnson(clk,rst,count);

 input clk,rst;
 output reg[3:0] count;

 always @(posedge clk)
  begin
   if(rst==1)
     count<=4'b0;
   else
     begin
```

```
        count<={!count[0],count[3:1]};
      end
   end
endmodule
```

Test Bench:

```
// Code your testbench here
// or browse Examples
module tb();

 reg clk,rst;
 wire [3:0] count;

 johnson dut (clk,rst,count);

 always #5 clk=!clk;
 initial
   begin
     clk=0;
     rst=1;
     @(negedge clk);
     rst=0;
   end

 initial
   begin
     $dumpfile("dump.vcd");
     $dumpvars();

     repeat(20)
       @(negedge clk);
     $finish();
   end
endmodule
```

Waveform:

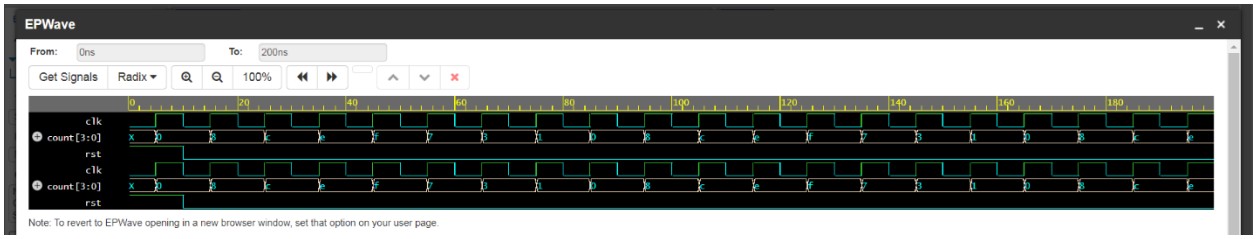

Fig.7.11 Waveform of Johnson Counter using Verilog Code

Up and Down Counter:

Verilog Design:

```
// Code your design here
module up_down_counter(q,up_down,clk,reset);
 output [3:0]q;
 input up_down,clk,reset;
 reg[3:0]q=0;
 always@(posedge clk)
  begin
   if(reset)
     begin
       q<=4'b0;
     end
   else if (up_down)
     begin
       q<=q+1;
     end
   else begin
```

```
        q<=q-1;
      end
    end
endmodule
```

Test Bench:

```
// Code your testbench here
// or browse Examples
module test;
 reg up_down,clk,reset;
 wire[3:0]q;
 up_down_counter test(q,up_down,clk,reset);
 initial begin
  $dumpfile("test.vcd");
  $dumpvars(1);
  up_down=0;
  clk=0;
  reset=0;
  #200 $finish;
 end
 always #80 up_down=~up_down;
 always #8 clk=~clk;
endmodule
```

Waveform:

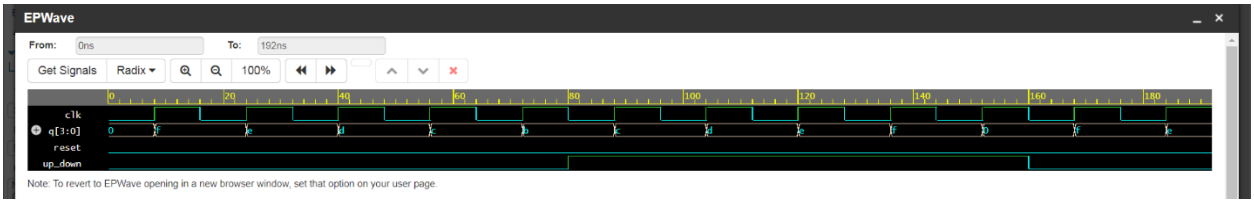

Fig.7.12 Waveform of Up Down Counter using Verilog Code

SISO Design:

Verilog Design:

```
// Code your design here
module siso(
  input si,clk,rst,
  output reg so
);
  reg [3:0]temp;
  always @(posedge clk)begin
    if(rst==1) begin
      temp=4'b0;
    end
    else begin
      temp=temp>>1;
      temp[3]<=si;
      so<=temp[0];
    end
  end
endmodule
```

Test Bench:

```
// Code your testbench here
// or browse Examples
module tb;
  reg si,clk,rst;
  wire so;
  siso siso1(si,rst,clk,so);
  initial begin
    clk=1;
    $dumpfile("file.vcd");
    $dumpvars(1);
    forever #5 clk=~clk;
  end

  initial begin
```

```
    rst=1;
    si=0;#10;
    rst=0;
    si=1;#10;
    si=0;#10;
    si=1;#10;
    si=0;#10;
    $finish;
  end
endmodule
```

Waveform:

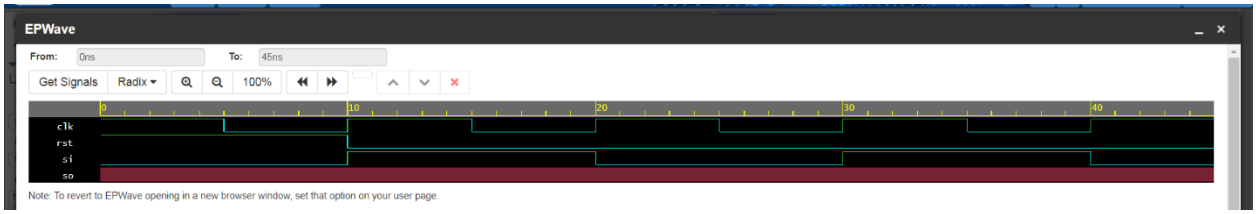

Fig.7.13 Waveform of SISO Design using Verilog Code

PISO Design:

Verilog Design:

```
// Code your design here
module piso(
  input [3:0]pi,
  input clk,load,
  output reg so
);
  reg [3:0]temp;
  always@(posedge clk)begin
    if(load!=1)begin;
      temp<=pi;
    end
```

```
   else begin
     so<=temp[0];
     temp=temp>>1;
   end
  end
endmodule
```

Test Bench:

```
// Code your testbench here
// or browse Examples
module tb;
  reg[3:0]pi;
  reg clk,load;
  wire so;
  piso piso1(clk,pi,load,so);
  initial
    begin
      clk=1;
      $dumpfile("file.vcd");
      $dumpvars(1);
      forever #5 clk=~clk;
    end
  initial
    begin
      load=0;
      pi=1010;#10
      load=1010;#10
      load=1010;#10
      load=1010;#10
      load=1010;#10
      load=0;
      pi=1101;#10
      load=1;#40
```

```
        $finish;
    end
endmodule
```

Waveform:

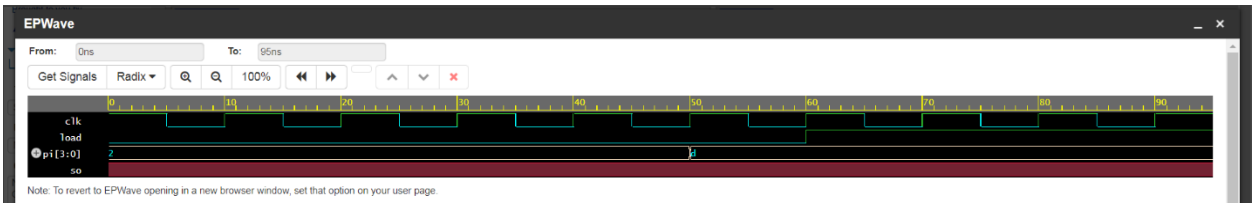

Fig.7.14 Waveform of PISO Design using Verilog Code

Chapter 8

Conclusion

8.1 Conclusion

In conclusion, my industrial training at SILICONOVA has been an enriching experience filled with invaluable opportunities for growth and learning. The friendly and supportive environment fostered a conducive atmosphere for exploration and development. Throughout the internship, I encountered numerous opportunities to delve into the intricate world of ASIC manufacturing, gaining insights into cutting-edge technologies and industry practices. The exposure to real-world projects and hands-on assignments, such as RTL coding, equipped me with practical skills and knowledge that are highly relevant in today's competitive landscape.

Moreover, the experience illuminated the vast array of career prospects and possibilities within the ASIC industry, showcasing a promising future for those passionate about semiconductor design and innovation. As the world progresses towards a more digitalized future, the demand for skilled professionals in ASIC design continues to soar, making it an exciting field to pursue.

Furthermore, the transition to an online internship format seamlessly accommodated the evolving dynamics of remote work, underscoring SILICONOVA's adaptability and commitment to fostering learning opportunities regardless of the circumstances. Despite the virtual setting, the engagement and support from mentors and colleagues remained unwavering, ensuring a fulfilling and productive experience.

Overall, my time at SILICONOVA has been a remarkable journey marked by growth, learning, and invaluable experiences. I am immensely grateful for the opportunity to be part of such a dynamic and innovative company, and I look forward to applying the knowledge and skills acquired during my internship to contribute meaningfully to the field of ASIC manufacturing in the future.