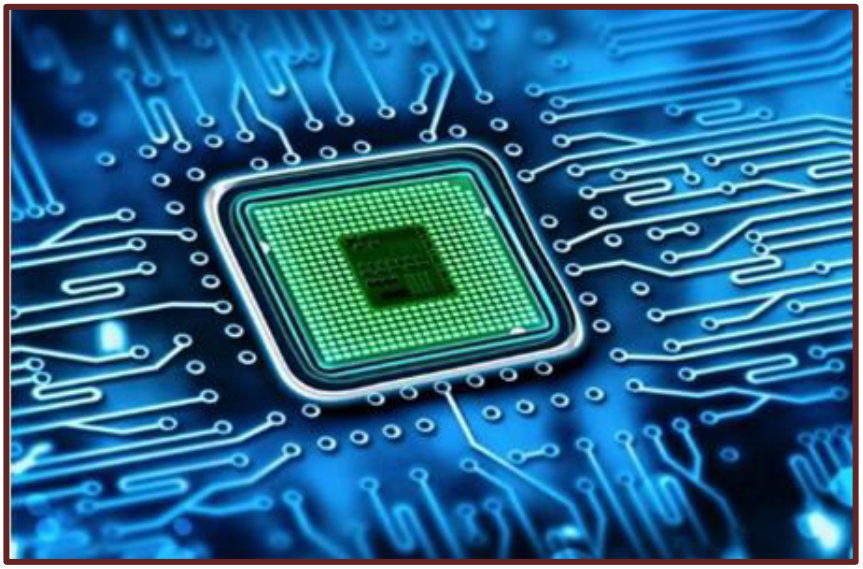
MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(AN AUTONOMOUS INSTITUTION)
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
Accredited by NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

LABORATARY MANUAL OF

DIGITAL CMOS IC DESIGN LAB

III YEAR B.TECH II-SEM ECE (R24)



A.Y.2026-27

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

PREPARED BY

Ms. PRANALI SURKAR, Asst. Prof
Mrs. R. BABITHA, Asst. Prof

INDEX

MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(AN AUTONOMOUS INSTITUTION)
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
Accredited by NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that this manual is a Bonafide record of practical work carried out in the **Digital CMOS IC Design Laboratory** in **B.Tech. (Electronics and Communication Engineering) VI Semester** programme during the **academic** year **2026-2027**.

This manual has been prepared by **Ms. Pranali Surkar (Assistant Professor), Mrs. R. Babitha (Assistant Professor),** Department of Electronics and Communication Engineering, with my/our own efforts and to the best of our knowledge.

Signature of Lab Faculty **Signature of HOD**

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

PREFACE

This laboratory lays the foundation for the Electronics and Communication Engineering students during Third year of their course.

In Digital CMOS IC Design Lab students will design the circuits of digital system and implement them in Cadence tool. Students will know how to write Verilog programming of digital circuits and simulate using Cadence software tool. After performing all the experiments included in this Laboratory, it is hoped the student receives good training to handle any electronic equipment available in electronics field.

By
Ms. Pranali Surkar
Mrs. R. Babitha

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ACKNOWLEDGEMENT

It was really a good experience, working with Design CMOS IC Design laboratory. First we would like to thank **Dr. N. Srinivas, Professor, HOD** of Department of Electronics and Communication Engineering, Marri Laxman Reddy Institute of technology & Management for his concern and giving the technical support in preparing the document.

We are deeply indebted to and gratefully acknowledge the constant support and valuable patronage of **Dr. Ravi Prasad, Dean (Academics), Marri Laxman Reddy Institute of Technology & Management,** for giving us this wonderful opportunity to prepare the **Electronic Devices and circuits Laboratory Manual.**

We express our hearty thanks to **Dr. R.Murali Prasad, Principal,** Marri Laxman Reddy Institute of technology & Management**,** for timely corrections and scholarly guidance.

We express our hearty thanks to **Dr. P. Sridhar**, Director, Marri Laxman Reddy Institute of technology & Management, for timely corrections and scholarly guidance.

At last, but not the least I would like to thanks the entire ECE Department faculty those who had inspired and helped us to achieve our goal.

By
Ms. Pranali Surkar
Mrs. R. Babitha

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

GENERAL INSTRUCTIONS

1. Students are instructed to come to Digital CMOS IC Design Laboratory on time. Late comers are not entertained in the lab
2. Students should be punctual to the lab. If not, the conducted experiments will not be repeated.
3. Students are expected to come prepared at home with the experiments which are going to be performed.
4. Students are instructed to display their identity cards before entering into the lab.
5. Students are instructed not to bring mobile phones to the lab.
6. Any damage/loss of equipment like mouse, keyboard, CPU etc., during the lab session, it is student's responsibility and penalty or fine will be collected from the student.
7. Students should update the records and lab observation books session wise. Before leaving the lab, the student should get his lab observation book signed by the faculty.
8. Students should submit the lab records by the next lab to the concerned faculty members in the staffroom for their correction and return.
9. Students should not move around the lab during the lab session.
10. If any emergency arises, the student should take the permission from faculty member concerned in written format.
11. The faculty members may suspend any student from the lab session on disciplinary grounds.
12. Never copy the output from other students. Write down your own outputs.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Vision of the Institute

To be a globally recognized institution that fosters innovation, excellence, and leadership in education, research, and technology development, empowering students to create sustainable solutions for the advancement of society.

Mission of the Institute

To foster a transformative learning environment that empowers students to excel in engineering, innovation, and leadership.

To produce skilled, ethical, and socially responsible engineers who contribute to sustainable technological advancements and address global challenges.

To develop future leaders through innovative research, strong industry collaboration, and meaningful community engagement..

Quality Policy

The management is committed in assuring quality service to all its stakeholders, students, parents, alumni, employees, employers, and the community.

Our commitment and dedication are built into our policy of continual quality improvement by establishing and implementing mechanisms and modalities ensuring accountability at all levels, transparency in procedures, and access to information and actions.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Vision of the Department

To provide quality technical education in Electronics and Communication Engineering through research, innovation, striving for global recognition in specified domain, leadership, and sustainable societal solutions.

Mission of the Department

- To create a transformative learning environment that empowers students in electronics and communication engineering, fostering excellence in technical skills and leadership.
- To drive innovation through research, deliver a transformative education grounded in ethical principles, and nurture the development of professionals
- To cultivate strong industry partnerships, and engaging actively with the community for societal and technological progress.

Program educational Objectives (PEOs)

PEO 1: Have Successful career in Industry

Graduates will excel in the Electronics and Communication industry with a strong foundation in technical expertise, continuous learning, and innovation.

PEO 2: Show Excellence in higher studies/Research

Graduates will excel in higher studies and research in Electronics and Communication Engineering (ECE) through a combination of rigorous academic dedication, cutting-edge innovation, and a deep understanding of emerging technologies.

PEO 3: Show Good Competency towards Entrepreneurship

Graduates will have to show good competency towards entrepreneurship in the field of Electronics and Communication Engineering, one must demonstrate an in-depth understanding

of emerging technologies, market trends, and the ability to innovate within this rapidly evolving industry.

Program Outcomes (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

1. Analyze and design analog & digital circuits or systems for a given specification and function.
2. Implement functional blocks of hardware-software co-designs for signal processing and communication applications.

COURSE DESCRIPTOR

DIGITAL COMS IC DESIGN LABORATORY

1	Department	ELECTRONICS & COMMUNICATION ENGINEERING							
2	Course Name	DIGITAL COMS IC DESIGN LABORATORY							
3	Course Code	2460479							
4	Year/Semester	III/II							
5	Regulation	MLRS-R24							
6	Structure of the course	Theory				Practical			
		Lecture	Tutorials	Practical	Credit	L	T	P	C
		0	0	0	0	0	0	2	1
7	Type of course	BS	HS	ES	PC	PE	OE	PS	MC
		×	×	×	✓	×	×	×	×
8	Course Offered	Odd Semester		×	Even Semester				✓
9	Total lecture, tutorial and practical hours for this course Offered (16 weeks of teaching per semester)								
	Lectures: 0 Hours	Tutorials: 0 hours		Practical: 32 hours					
10	Course Coordinator								
11	Date Approved by BOS								
12	Course Webpage	www.mlritm.ac.in/							

13		Level	Course Code	Semester	Prerequisites
	Prerequisites/ Co-requisites	3	2230423	II-I	Basics of VLSI Design. (Digital System Design)

14. Course Overview:

This course provides hands-on experience in CMOS digital design using HDL and FPGA platforms. Students learn MOS/CMOS characteristics, design and analysis of combinational and

sequential circuits and development of complex systems such as ALUs, FSMs, and memories. The course also emphasizes CMOS layout design, verification checks and practical implementation of digital circuits for real-world applications.

15. Course Objectives:

The students will try to learn:

- MOSFET and CMOS characteristics for digital circuit design
- Combinational and sequential CMOS circuits and evaluate their performance.
- Layouts of CMOS circuits and validate them using DRC, LVS, and RC checks
- Application of HDL to design, simulate and synthesize digital systems such as ALUs, FSMs and memories
- Implementation and verify designs on FPGA/CPLD platforms for practical applications.

16. Course Outcomes:

CO1	Analyze MOSFET and CMOS characteristics and basic digital logic design.
CO2	Design and implement combinational circuits such as adders, multipliers and encoders/decoders
CO3	Develop sequential circuits including flip-flops, counters, shift registers and sequence detectors.
CO4	Simulate and synthesize complex systems like ALUs, FSMs and memories using HDL
CO5	Perform layout verification checks and implement CMOS circuits on FPGA/CPLD

17. Employability Skills:

Example: Communication skills / Programming skills / Project based skills/
In the CMOS IC Design Lab, students gain practical skills in designing,simulating and synthesizing digital circuits using tools like Cadence. They also develop problem-solving, teamwork, and communication skills essential for careers in VLSI and semiconductor industries.

18. Content Delivery / Instructional Methodologies:

✓	 Day to Day lab evaluation	✓	 Demo Video	✓	 Viva Voice questions	x	 Open Ended Experiments
x	 Competitions	x	 Hackathons	✓	 Certifications	✓	Probing Further Questions

19. Evaluation Methodology:

Each laboratory will be evaluated for a total of 100 marks consisting of 40 marks for Continuous Internal Evaluation (CIE) and 60 marks for semester end lab examination. Out of 40 marks for internal evaluation:

- A write-up on day-to-day experiment (aim, components/procedure, expected outcome) which shall be evaluated for 10 marks
- 10 marks for viva-voce/ tutorial/ case study/ application/ poster presentation.
- Internal practical examination shall be evaluated for 10 marks.
- The remaining 10 marks are for Laboratory Project (Design/ Software / Hardware Model/ App Development/ Prototype).

Table 1: CIE marks distribution

Component				
Type of Assessment	Day to Day performance and viva voice examination	Final internal lab assessment	Laboratory Report / Project and Presentation	Total Marks
CIE marks	20	10	10	40

Continuous Internal Evaluation (CIE): Two CIE exams shall be conducted at the 8th week and 16th week of the semester; the average of the two CIEs will be taken into account. The CIE exam is conducted for 10 marks.

The Semester End Examination shall be conducted with an external examiner and the laboratory teacher. The external examiner shall be appointed from the other colleges which will be decided by the Head of the institution.

In the Semester End Examination held for 3 hours, total 60 marks are divided and allocated as shown below:

- 10 marks for write-up
- 15 for experiment/program
- 15 for evaluation of results
- 10 marks for presentation on another experiment/program in the same laboratory course and
- 10 marks for viva-voce on concerned laboratory course.

20. Course content:

CO 1	Design and realization of all basic logic gates
	Design of a 4-bit Binary-to-Gray Code Converter.
CO 2	Design and realization of all basic logic gates
	Design and implementation of an 8-bit Adder.
	Design and implementation of a 4-bit Multiplier.
	Design and implementation of an Arithmetic Logic Unit (ALU)
	Design of a 4-bit Binary-to-Gray Code Converter.
	Design of a 4-bit Comparator.
	Design and implementation of a 4-bit Parity Generator and Checker.
CO 3	Design and implementation of a Universal Shift Register.
	Design and realization of Flip-Flops: SR, D, JK, and T.
	Design and implementation of Ripple Counters (Mos-10 and Mod-12)
	Design and simulation of a Finite State Machine (Moore/Mealy Model)
	Design and implementation of Memory Elements.
CO 4	Design and implementation of an 8-bit Adder.
	Design and implementation of a 4-bit Multiplier.

	Design and implementation of an Arithmetic Logic Unit (ALU)
	Design and implementation of a Universal Shift Register.
	Design of a 4-bit Comparator.
	Design and realization of Flip-Flops: SR, D, JK, and T.
CO 5	Design and realization of all basic logic gates
	Design and implementation of an 8-bit Adder.
	Design and implementation of a 4-bit Multiplier.
	Design of a 4-bit Comparator.
	Design and implementation of Ripple Counters (Mos-10 and Mod-12)
	Design and simulation of a Finite State Machine (Moore/Mealy Model)

21. Course Plan:

The course plan is meant as a guideline. Probably there may be changes.

S. No	Topics to be covered	CO's	Reference
1	Course Description on Outcome Based Education (OBE): Course Objectives, Course Outcomes (CO), Program Outcomes (PO) and CO-PO Mapping	--	--
2	Design and realization of all basic logic gates	CO1,2,5	T2: 1.4.2, 1.4.3, 1.4.4
3	Design and implementation of an 8-bit Adder.	CO2,4,5	T2: 1.8.4
4	Design and implementation of a 4-bit Multiplier.	CO2,4,5	T2: 12.5.1
5	Design of an 8-to-3 Encoder (with and without priority) and a 2-to-4 Decoder.	CO2,4,5	T2: 2.1, 2.2,2.3
6	Design and implementation of an Arithmetic Logic Unit (ALU)	CO2,4	T2: 12.5.1
7	Design of a 4-bit Binary-to-Gray Code Converter.	CO1,2	T2: 2.4
8	Design and implementation of a Universal Shift Register.	CO3,4	T2: 12.5.1

9	Design of a 4-bit Comparator.	CO2,4,5	T2: 2.5
	Design and realization of Flip-Flops: SR, D, JK, and T.	CO3,4	T2: 4.3,4.4,4.5,4.6
	Design and implementation of Ripple Counters (Mos-10 and Mod-12)	CO3,4,5	T2: 11.5
	Design and simulation of a Finite State Machine (Moore/Mealy Model)	CO3,4,5	T2:5.1,5.2,5.3
	Design and implementation of Memory Elements.	CO3,4	T1: 5.4,5.5
	Design and implementation of a 4-bit Parity Generator and Checker.	CO2,4	T2:2.6,2.7
	Design of a Sequence Detector using Finite State Machine.	CO3,4,5	T2: A.6.1

TEXTBOOKS:

1. Philip E. Allen and Douglas R. Holberg, "CMOS Analog Circuit Design," Oxford University Press, International 2nd Edition, 2010

2. Neil H. E. Weste, David Harris, Ayan Banerjee, "CMOS VLSI DESIGN-A circuits and Systems Perspective", 4th Edition, Pearson, 2nd Edition 2015

22. Experiments for Enhanced Learning:

S.No	Design Oriented Experiments
1	Design and implement 4x1 multiplexer.
2	Verify De'morgans Theorem for 2 variables.

23. PROGRAM OUTCOMES & PROGRAM SPECIFIC OUTCOMES:

PO 1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering

fundamentals, and engg. specialization to the solution of complex engineering problems.
PO 2: Problem analysis: Identify, formulate, research literature, and analyze engineering problems to arrive at substantiated conclusions using first principles of mathematics, natural, and engineering sciences.
PO 3: Design/development of solutions: Design solutions for complex engineering problems and design system components, processes to meet the specifications with consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO 4: Conduct investigations of complex problems: Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO 5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO 6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO 7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO 8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO 9: Individual and team work: Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
PO 10: Communication: Communicate effectively with the engineering community and with society at large. Be able to comprehend and write effective reports documentation. Make effective presentations, and give and receive clear instructions.

PO 11: Project management and finance: Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team. Manage projects in multidisciplinary environments.
PO 12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.
Program Specific Outcomes
PSO 1: Analyze and design analog & digital circuits or systems for a given specification and function.
PSO 2: Implement functional blocks of hardware-software co-designs for signal processing and communication applications.

24. HOW PROGRAM OUTCOMES ARE ASSESSED:

Program Outcomes		Strength	Proficiency Assessed by
PO1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and engg. specialization to the solution of complex engineering problems.	3	CIE/SEE/ Viva-Voce/ Day to Day Performance / Project & Presentation
PO2	Problem analysis: Identify, formulate, research literature, and analyze engineering problems to arrive at substantiated conclusions using first principles of mathematics, natural, and engineering sciences.	3	CIE/SEE/ Viva-Voce/ Day to Day Performance / Project & Presentation
PO3	Design/development of solutions: Design solutions for complex engineering problems and design system components, processes to meet the specifications with consideration for the public health and safety, and the cultural, societal, and environmental considerations.	3	CIE/SEE/ Viva-Voce/ Day to Day Performance / Project & Presentation
PO4	Conduct investigations of complex problems: Use	3	CIE/SEE/ Viva-

	research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.		Voce/ Day to Day Performance / Project & Presentation
PO5	Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.	3	CIE/SEE/ Viva-Voce/ Day to Day Performance / Project & Presentation
PO12	Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.	3	CIE/SEE/ Viva-Voce/ Day to Day Performance /Project & Presentation

25. HOW PROGRAM SPECIFIC OUTCOMES ARE ASSESSED:

Program Outcomes		Strength	Proficiency Assessed by
PSO1	Analyze and design analog & digital circuits or systems for a given specification and function.	3	CIE/SEE/ Viva-Voce/ Day to Day Performance / Project & Presentation
PSO2	Implement functional blocks of hardware-software co-designs for signal processing and communication applications.	3	CIE/SEE/ Viva-Voce/ Day to Day Performance / Project & Presentation

3 = High; 2 = Medium; 1 = Low

26. MAPPING OF EACH CO WITH PO(s),PSO(s):

Course Outc	PROGRAM OUTCOMES													
	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO1	PSO 2

ome s														
CO1	✓	✓	✓	✓	✓	-	-	-	-	-	-	✓	✓	✓
CO2	✓	✓	✓	✓	✓	-	-	-	-	-	-	✓	✓	✓
CO3	✓	✓	✓	✓	✓	-	-	-	-	-	-	✓	✓	✓
CO4	✓	✓	✓	✓	✓	-	-	-	-	-	-	✓	✓	✓
CO5	✓	✓	✓	✓	✓	-	-	-	-	-	-	✓	✓	✓

27. JUSTIFICATIONS FOR CO – PO / PSO MAPPING - DIRECT:

Course Outcomes	PO'S/ PSO'S	Justification for mapping (Students will be able to)	No. of Key Competencies
CO1	PO1	1. Application of scientific principles and methodologies. 2. Application of specialized engineering knowledge in complex engineering problems	2
	PO2	1. Recognizing and defining complex engineering problems or opportunities. 2. Structuring and abstracting the problem for systematic analysis. 3. Applying mathematical, natural, and engineering sciences in problem-solving. 4. Planning and conducting experiments for problem analysis. 5. Implementing and testing solutions through experimentation.	5
	PO3	1. Use creativity to develop innovative engineering solutions.	1
	PO4	1. Develop essential laboratory and workshop skills to carry out experimental investigations and gather reliable data. 2. Follow appropriate codes of practice and industry standards when analyzing and interpreting experimental data.	2

	PO5	1. Develop engineering solutions using modern tools across various disciplines. 2. Identify appropriate prediction and modelling tools for diverse engineering applications. 3. Utilize IT tools in engineering analysis, design, and decision-making. 4. Implement simulation tools in different engineering fields.	4
	PO12	1. Pursue professional, Academic, Global certifications. 2. Begin and work towards advanced programs to further deepen knowledge in engineering and related areas. 3. Stay updated on industry trends and emerging technologies to remain relevant in the field. 4. Learn at least 2–3 new significant skills annually to ensure continuous growth and development. 5. Dedicate time for formal training for a standard duration of training each year. 6. Engage in ongoing self-improvement efforts to enhance both personal and professional growth. 7. Be adaptable to technological changes by actively pursuing new learning opportunities and challenges.	7
	PSO1	1. Analyze response of a circuit or system. 2. Design of a circuit or system for a given specification. 3. Understand and apply circuit or system specifications accurately.	3
	PSO2	1. Proficiency in the use of software tools for circuit design. 2. Hardware-software integration in analog and digital systems.	2

CO2	PO1	1. Application of scientific principles and methodologies. 2. Application of specialized engineering knowledge in complex engineering problems.	2
	PO2	1. Recognizing and defining complex engineering problems or opportunities. 2. Structuring and abstracting the problem for systematic analysis. 3. Applying mathematical, natural, and engineering sciences in problem-solving. 4. Planning and conducting experiments for problem analysis. 5. Implementing and testing solutions through experimentation.	5
	PO3	1. Use creativity to develop innovative engineering solutions.	1
	PO4	1. Develop essential laboratory and workshop skills to carry out experimental investigations and gather reliable data. 2. Follow appropriate codes of practice and industry standards when analyzing and interpreting experimental data.	2
	PO5	1. Develop engineering solutions using modern tools across various disciplines. 2. Identify appropriate prediction and modelling tools for diverse engineering applications. 3. Utilize IT tools in engineering analysis, design, and decision-making. 4. Implement simulation tools in different engineering fields.	4
	PO12	1. Pursue professional, Academic, Global certifications. 2. Begin and work towards advanced programs to further deepen knowledge in engineering and related areas. 3. Stay updated on industry trends and emerging	7

		technologies to remain relevant in the field. 4. Learn at least 2–3 new significant skills annually to ensure continuous growth and development. 5. Dedicate time for formal training for a standard duration of training each year. 6. Engage in ongoing self-improvement efforts to enhance both personal and professional growth. 7. Be adaptable to technological changes by actively pursuing new learning opportunities and challenges.	
	PSO1	1. Analyze response of a circuit or system 2. Design of a circuit or system for a given specification 3. Understand and apply circuit or system specifications accurately.	3
	PSO2	1. Develop Operational block diagrams 2. Proficiency in the use of software tools for circuit design. 3. Hardware-software integration in analog and digital systems	3
CO3	PO1	1. Application of scientific principles and methodologies. 2. Application of specialized engineering knowledge in complex engineering problems	2
	PO2	1. Recognizing and defining complex engineering problems or opportunities. 2. Structuring and abstracting the problem for systematic analysis. 3. Applying mathematical, natural, and engineering sciences in problem-solving. 4. Planning and conducting experiments for problem analysis. 5. Implementing and testing solutions through experimentation.	5
	PO3	1. Use creativity to develop innovative engineering solutions.	1

	PO4	1. Develop essential laboratory and workshop skills to carry out experimental investigations and gather reliable data. 2. Apply fundamental engineering principles to analyze and interpret key engineering processes and challenges.	2
	PO5	1. Develop engineering solutions using modern tools across various disciplines. 2. Identify appropriate prediction and modeling tools for diverse engineering applications. 3. Utilize IT tools in engineering analysis, design, and decision-making. 4. Implement simulation tools in different engineering fields.	4
	PO12	1. Pursue professional, Academic, Global certifications. 2. Begin and work towards advanced programs to further deepen knowledge in engineering and related areas. 3. Stay updated on industry trends and emerging technologies to remain relevant in the field. 4. Learn at least 2–3 new significant skills annually to ensure continuous growth and development. 5. Dedicate time for formal training for a standard duration of training each year. 6. Engage in ongoing self-improvement efforts to enhance both personal and professional growth. 7. Be adaptable to technological changes by actively pursuing new learning opportunities and challenges.	7
	PSO1	1. Analyze response of a circuit or system 2. Design of a circuit or system for a given specification 3. Understand and apply circuit or system specifications accurately.	3
	PSO2	1. Develop Operational block diagrams 2. Proficiency in the use of software tools for	3

		circuit design. 3. Hardware-software integration in analog and digital systems	
CO4	PO1	1. Application of scientific principles and methodologies. 2. Application of specialized engineering knowledge in complex engineering problems	2
	PO2	1. Recognizing and defining complex engineering problems or opportunities. 2. Structuring and abstracting the problem for systematic analysis. 3. Applying mathematical, natural, and engineering sciences in problem-solving. 4. Planning and conducting experiments for problem analysis. 5. Implementing and testing solutions through experimentation.	5
	PO3	1. Use creativity to develop innovative engineering solutions.	1
	PO4	1. Develop essential laboratory and workshop skills to carry out experimental investigations and gather reliable data. 2. Apply fundamental engineering principles to analyze and interpret key engineering processes and challenges.	2
	PO5	1. Develop engineering solutions using modern tools across various disciplines. 2. Identify appropriate prediction and modeling tools for diverse engineering applications. 3. Utilize IT tools in engineering analysis, design, and decision-making. 4. Implement simulation tools in different engineering fields.	4
	PO12	1. Pursue professional, Academic, Global certifications. 2. Begin and work towards advanced programs to further deepen knowledge in engineering and	7

		related areas. 3. Stay updated on industry trends and emerging technologies to remain relevant in the field. 4. Learn at least 2–3 new significant skills annually to ensure continuous growth and development. 5. Dedicate time for formal training for a standard duration of training each year. 6. Engage in ongoing self-improvement efforts to enhance both personal and professional growth. 7. Be adaptable to technological changes by actively pursuing new learning opportunities and challenges.	
	PSO1	1. Analyze response of a circuit or system. 2. Design of a circuit or system for a given specification. 3. Understand and apply circuit or system specifications accurately. 4. Knowledge of analog and digital signal processing techniques.	4
	PSO2	1. Develop Operational block diagrams 2. Proficiency in the use of software tools for circuit design. 3. Hardware-software integration in analog and digital systems	3
CO5	PO1	1. Application of scientific principles and methodologies. 2. Application of specialized engineering knowledge in complex engineering problems.	2
	PO2	1. Recognizing and defining complex engineering problems or opportunities. 2. Structuring and abstracting the problem for systematic analysis. 3. Applying mathematical, natural, and engineering sciences in problem-solving. 4. Planning and conducting experiments for problem analysis. 5. Implementing and testing solutions through experimentation.	5

	PO3	1. Use creativity to develop innovative engineering solutions.	1
	PO4	1. Develop essential laboratory and workshop skills to carry out experimental investigations and gather reliable data. 2. Apply fundamental engineering principles to analyze and interpret key engineering processes and challenges.	2
	PO5	1. Develop engineering solutions using modern tools across various disciplines. 2. Identify appropriate prediction and modeling tools for diverse engineering applications. 3. Utilize IT tools in engineering analysis, design, and decision-making. 4. Implement simulation tools in different engineering fields.	4
	PO12	1. Pursue professional, Academic, Global certifications. 2. Begin and work towards advanced programs to further deepen knowledge in engineering and related areas. 3. Stay updated on industry trends and emerging technologies to remain relevant in the field. 4. Learn at least 2–3 new significant skills annually to ensure continuous growth and development. 5. Dedicate time for formal training for a standard duration of training each year. 6. Engage in ongoing self-improvement efforts to enhance both personal and professional growth. 7. Be adaptable to technological changes by actively pursuing new learning opportunities and challenges.	7

	PSO1	1. Analyze response of a circuit or system 2. Design of a circuit or system for a given specification 3. Understand and apply circuit or system specifications accurately. 4. Knowledge of analog and digital signal processing techniques.	4
	PSO2	1. Develop Operational block diagrams 2. Proficiency in the use of software tools for circuit design. 3. Hardware-software integration in analog and digital systems	3

28. TOTAL COUNT OF KEY COMPETENCIES FOR CO – (PO, PSO) MAPPING:

Course Outcomes	PROGRAM OUTCOMES												PSOs	
	1	2	3	4	5	6	7	8	9	10	11	12	1	2
	4	10	10	10	4	5	4	4	10	5	10	8	4	4
CO1	2	5	1	2	4	-	-	-	-	-	-	7	3	2
CO2	2	5	1	2	4	-	-	-	-	-	-	7	3	3
CO3	2	5	1	2	4	-	-	-	-	-	-	7	3	3
CO4	2	5	1	2	4	-	-	-	-	-	-	7	4	3
CO5	2	5	1	2	4	-	-	-	-	-	-	7	4	3

29. PERCENTAGE OF KEY COMPETENCIES FOR CO – (PO/ PSO):

Course Outcomes	PROGRAM OUTCOMES												PSOs	
	1	2	3	4	5	6	7	8	9	10	11	12	1	2
	4	10	10	10	4	5	4	4	10	5	10	8	4	4
CO 1	50	50	10	20	100	-	-	-	-	-	-	87.5	75	50
CO 2	50	50	10	20	100	-	-	-	-	-	-	87.5	75	75
CO 3	50	50	10	20	100	-	-	-	-	-	-	87.5	75	75
CO 4	50	50	10	20	100	-	-	-	-	-	-	87.5	100	75
CO 5	50	50	10	20	100	-	-	-	-	-	-	87.5	100	75

30. COURSE ARTICULATION MATRIX (PO – PSO MAPPING):

CO'S and PO'S, CO'S and PSO'S on the scale of 0 to 3, 0 being no correlation, 1 being the low correlation, 2 being medium correlation and 3 being high correlation.

0 - 0≤ C≤ 5% – No correlation, 2 - 40 % <C< 60% –Moderate

1-5 <C≤ 40% – Low/ Slight 3 - 60% ≤ C < 100% – Substantial /High

Course Outcomes	PROGRAM OUTCOMES												PSOs	
	PO 1	PO 2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO 2
CO 1	2	2	1	1	3	-	-	--	-	-	-	3	3	2
CO 2	2	2	1	1	3	-	-	-	-	-	-	3	3	3
CO 3	2	2	1	1	3	-	-	-	-	-	-	3	3	3
CO 4	2	2	1	1	3	-	-	-	-	-	-	3	3	3
CO 5	2	2	1	1	3	-	-	-	-	-	-	3	3	3
TOTAL	**10**	**10**	**5**	**5**	**15**	**-**	-	-	-	-	-	**15**	**15**	**14**

AVERAGE	2	2	1	1	3	-	-	-	-	-	-	3	3	2.8

31. ASSESSMENT METHODOLOGY DIRECT:

CIE Exams	✓	SEE	✓	Laboratory Practices	✓
Certification	--	Viva-Voce/PPT/ Project	✓	Open Ended Experiments	-

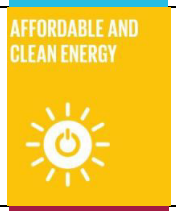
32. ASSESSMENT METHODOLOGY INDIRECT:

✓	Course End Survey (CES)

33. RELEVANCE TO SUSTAINABILITY GOALS:

The Digital CMOS IC Design Laboratory supports sustainability goals by enabling the development of energy-efficient communication systems and fostering innovation in digital technologies, reducing environmental impact and promoting sustainable connectivity.

1		
2		
3		
4		Provides practical exposure to modern IC design tools and methodologies. Enhances skills in low-power and high-performance circuit design. Builds awareness of eco-friendly engineering practices.

5		
6		
7		CMOS technology is inherently low-power, reducing energy consumption in digital systems. Lab experiments (ALU, counters, FSMs) help students design power-efficient circuits. Promotes development of energy-saving electronic devices.
8		
9		Encourages innovation in VLSI and semiconductor technologies. Hands-on design of digital systems (encoders, registers, memory) supports modern chip design practices. Prepares students for sustainable semiconductor industry growth.
10		
11		CMOS circuits are the backbone of smart cities, IoT, and communication systems. Lab skills contribute to designing intelligent and sustainable digital infrastructure.

12		Efficient CMOS designs minimize power wastage and heat dissipation. Reduces the need for excessive cooling and energy resources. Promotes optimal use of electronic materials and resources.
13		Low-power CMOS circuits reduce overall carbon footprint of electronic systems. Supports development of green electronics.
14		
15		
16		
17		

Signature of Course Coordinator **HOD**

Name & Designation

2460479: DIGITAL CMOS IC DESIGN LABORATORY

III Year B.Tech. ECE II – Sem. **L T P C**

0 0 2 1

Course Overview:

This course provides hands-on experience in CMOS digital design using HDL and FPGA platforms. Students learn MOS/CMOS characteristics, design and analysis of combinational and sequential circuits and development of complex systems such as ALUs, FSMs, and memories. The course also emphasizes CMOS layout design, verification checks and practical implementation of digital circuits for real-world applications.

Pre-requisites: Basics of VLSI Design

Course Objectives:

The students will try to learn

- MOSFET and CMOS characteristics for digital circuit design
- Combinational and sequential CMOS circuits and evaluate their performance.
- Layouts of CMOS circuits and validate them using DRC, LVS, and RC checks
- Application of HDL to design, simulate and synthesize digital systems such as ALUs, FSMs and memories
- Implementation and verify designs on FPGA/CPLD platforms for practical applications.

Course Outcomes:

After successful completion of the course, students shall be able to

- Analyze MOSFET and CMOS characteristics and basic digital logic design.
- Design and implement combinational circuits such as adders, multipliers and encoders/decoders
- Develop sequential circuits including flip-flops, counters, shift registers and sequence detectors.
- Simulate and synthesize complex systems like ALUs, FSMs and memories using HDL
- Perform layout verification checks and implement CMOS circuits on FPGA/CPLD

List of Experiments:

Digital CMOS IC Design- Simulate it using any HDL tool or equivalent and implement by FPGA

1. Design and realization of all basic logic gates
2. Design and implementation of an 8-bit Adder.
3. Design and implementation of a 4-bit Multiplier.
4. Design of an 8-to-3 Encoder (with and without priority) and a 2-to-4 Decoder.
5. Design and implementation of an Arithmetic Logic Unit (ALU)
6. Design of a 4-bit Binary-to-Gray Code Converter.
7. Design and implementation of a Universal Shift Register.
8. Design of a 4-bit Comparator.
9. Design and realization of Flip-Flops: SR, D, JK, and T.
10. Design and implementation of Ripple Counters (Mos-10 and Mod-12)
11. Design and simulation of a Finite State Machine (Moore/Mealy Model)
12. Design and implementation of Memory Elements.
13. Design and implementation of a 4-bit Parity Generator and Checker.
14. Design of a Sequence Detector using Finite State Machine.

Open Ended Experiments

1. Design and implement 4x1 multiplexer.
2. Verify De'morgans Theorem for 2 variables.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DIGITAL CMOS IC DESIGN LABORATORY

Virtual lab details

Name of the Virtual Lab: DLD with Verilog Virtual Laboratory

Virtual Lab Host Institute: IIIT Hyderabad

URL/Link to Lab: https://dldv-iiith.vlabs.ac.in/Introduction.html

Academic Year: 2026-27

Semester: VI

List of Experiments Available in Virtual Lab

1. Design of Adder circuit using Verilog
2. Design of Multiplexer using Verilog
3. Design of ALU using Verilog
4. Design of Counter using Verilog
5. Design of Comparator using Verilog
6. Design of Latch and Flip Flops using Verilog
7. Design of Register using Verilog
8. Design of Multiplier using Verilog

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
DIGITAL CMOS IC DESIGN LABORATORY

LAB PLANNER

S.No	Experiment	CO	Virtual Lab Availabilty	Date planned	Date conducted
1	Design and realization of all basic logic gates	1,2,5	Yes		
2	Design and implementation of an 8-bit Adder.	2,4,5	Yes		
3	Design and implementation of a 4-bit Multiplier.	2,4,5	Yes		
4	Design and implementation of an Arithmetic Logic Unit (ALU)	2,4	Yes		
5	Design of a 4-bit Binary-to-Gray Code Converter.	1,2	No		
6	Design and implementation of a Universal Shift Register.	3,4	No		
7	**MID-I**				
8	Design of a 4-bit Comparator.	2,4,5	Yes		
9	Design and realization of Flip-Flops: SR, D, JK, and T.	3,4	Yes		
10	Design and implementation of Ripple Counters (Mos-10 and Mod-12)	3,4,5	Yes		

11	Design and simulation of a Finite State Machine (Moore/Mealy Model)	3,4,5	No		
12	Design and implementation of Memory Elements.	3,4	No		
13	Design and implementation of a 4-bit Parity Generator and Checker.	2,4	No		
14	**MID-II**				

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DIGITAL CMOS IC DESIGN LABORATORY

1. RUBRICS FOR DAY TO DAY EVALUATION

Parameter	Max Marks	Level-1 (Very Poor)	Level-2 (Poor)	Level-3 (Average)	Level-4 (Good)	Level-5 (Excellent)
Observation Book	05	No observations or irrelevant data. (0–1)	Incomplete or incorrect data. (2)	Basic values with some errors. (3)	Mostly correct with good format. (4)	Fully correct, clear, and well-formatted. (5)
Record Writing	05	Not submitted. (0-1)	Submitted but mostly incomplete. (2)	Submitted with some missing/wrong parts. (3)	Submitted with minor issues. (4)	Fully complete, correct algorithm & flowchart. (5)
Result	05	No result or major errors. (0–1)	Result partially obtained. (2)	Acceptable result with limited error. (3)	Near-correct result and reasonable error. (4)	Accurate result. (5)
Viva-Voce	05	Did not answer any questions. (1)	Answered very few questions. (2)	Answered some questions with help. (3)	Answered most questions correctly. (4)	Answered all questions accurately. (5)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DIGITAL CMOS IC DESIGN LABORATORY

2.RUBRICS FOR INTERNAL EVALUATION

Criterion	Max Marks	Level-1 *(Very Poor)*	Level-2 *(Poor)*	Level-3 *(Average)*	Level-4 *(Good)*	Level-5 *(Excellent)*
Design/Tool/Apparatus Selection	2 Marks	Incorrect tool/design and no reasoning. **(0)**	Tool/design selection attempted with unclear logic. **(0.5)**	Satisfactory selection with partial justification. **(1)**	Correct selection and proper analysis with few errors. **(1.5)**	Smart selection with accurate, relevant analysis. **(2)**
Execution (Code/Debug/Run) /Analysis/Method Used	4 Marks	Did not attempt or completely failed to execute. **(0)**	Attempted but unable to proceed or with major errors. **(1)**	Partial execution with some logic/syntax errors. **(2)**	Mostly correct execution with minimal help. **(3)**	Fully correct and independently executed program. **(4)**
Results& Documentation	2 Marks	Incomplete or poorly presented. **(0)**	Basic structure but lacks clarity or formatting. **(0.5)**	Complete but generic or with formatting issues. **(1)**	Well-structured and mostly clear. **(1.5)**	Well-organized, professional, and engaging documentation. **(2)**
Viva-Voce (Understanding of Concepts)	2 Marks	No understanding; could not answer questions. **(0)**	Answered a few with difficulty. **(0.5)**	Answered half the questions with basic clarity. **(1)**	Good understanding with confident answers. **(1.5)**	Answered all questions with clarity and depth. **(2)**

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
DIGITAL CMOS IC DESIGN LABORATORY

3. RUBRICS FOR SEMESTER END EXAMINATIONS

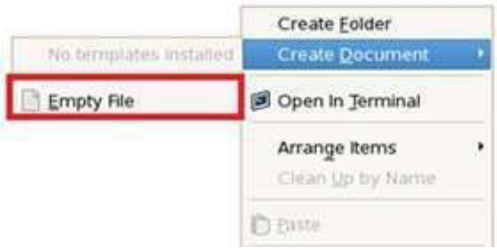
Criterion	Max Marks	Level-1 (Very Poor (*0–2 marks)*	Level-2 (Poor) *(3–4 marks)*	Level-3 (Average) *(5–6 marks)*	Level-4 (Good) *(7–9 marks)*	Level-5 (Excellent) *(10–12 marks)*
Preparedness for the Experiment	12 marks	No clarity on objective or procedure. Unable to explain basics.	Limited idea of the objective/procedure. Needed prompting.	Has basic understanding; minor gaps in concept or preparation.	Well-prepared, with clear understanding of steps and background.	Fully prepared with strong conceptual clarity and confident explanation.
Performance in the Laboratory	12 marks	Unable to perform experiment. Relied entirely on examiner's help.	Performed with multiple errors and constant support.	Performed with some errors; required occasional help.	Performed mostly independently with minimal support.	Performed independently, efficiently, and with precision.
Calculations & Graphs	12 marks	No or incorrect calculations. Graphs missing or irrelevant.	Multiple calculation errors. Graphs/plots inaccurate or poorly labeled.	Calculations partially correct. Graphs present but with some flaws.	Correct calculations and graphs with minor errors.	Accurate calculations and well-labeled graphs with proper interpretation.
Results & Error Analysis	12 marks	No result or invalid result. No error analysis attempted.	Incorrect result with vague or no error discussion.	Acceptable result. Error analysis attempted but limited.	Correct result with sound error discussion.	Accurate result with detailed and relevant error analysis.
Viva-Voce (Subject Knowledge)	12 marks	Unable to answer any questions. No conceptual understanding.	Answered few questions with poor logic.	Answered half of the questions with average understanding.	Answered most questions with clarity and confidence.	Answered all questions with depth, clarity, and reasoning.

Steps to use Cadence Tool

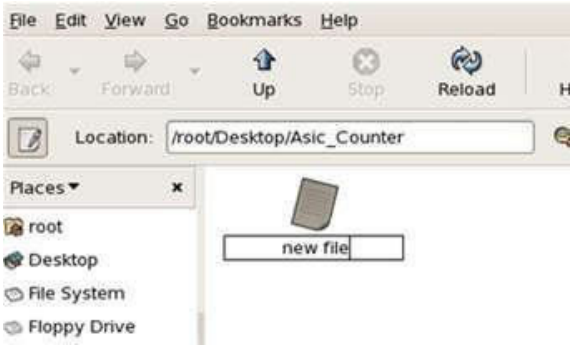
. Digital Simulation flow

Create a folder in any location. To create a folder right click and select the Create Folder option.
Note: Do not use space in folder name or filename.
It will create a folder like below and rename it to your requirement.

After creating the folder enter into the location and create a Verilog file specified below or copy the file to the location.



This creates a file shown below



Name the file as gates.v

Double click on gates.v file (or open it with gedit) and type your Verilog code specified below.

*gates.v gates_tb.v

```
`timescale 1ns/1ps

module tb();

reg a,b;
wire c,d,e,f,g,h,i;

gates uut (.a(a),.b(b),.c(c),.d(d),.e(e),.f(f),.g(g),.h(h),.i(i));

initial begin

a=0;b=0;#100;
a=0;b=1;#100;
a=1;b=0;#100;
a=1;b=1;#100;
a=0;b=0;#100;
end
endmodule
```

Follow similar procedure to create testbench file

Save the files and they should look like below window

*gates.v gates_tb.v

```
`timescale 1ns/1ps

module tb();

reg a,b;
wire c,d,e,f,g,h,i;

gates uut (.a(a),.b(b),.c(c),.d(d),.e(e),.f(f),.g(g),.h(h),.i(i));

initial begin

a=0;b=0;#100;
a=0;b=1;#100;
a=1;b=0;#100;
a=1;b=1;#100;
a=0;b=0;#100;
end
endmodule
```

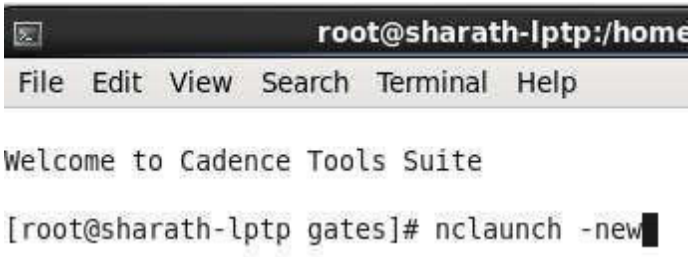
Right click in the same folder and give open in terminal Re check the location using "pwd " command.

Now invoke the Cadence tools using below commands from the terminal
csh

source /cad/cshrc

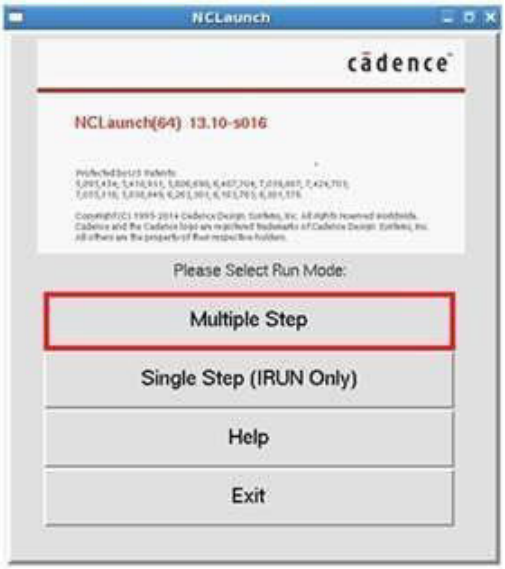
Now a welcome note appears on the terminal (if the commands are properly executed).

Now we will launch the Incisive tool for performing Simulation. Use the below command to invoke the tool.



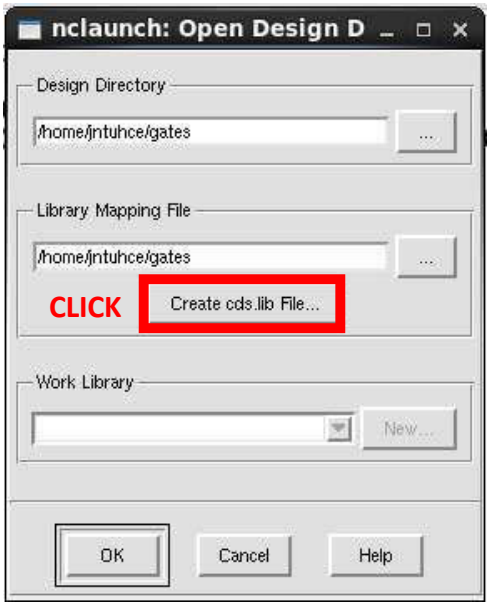
nclaunch -new

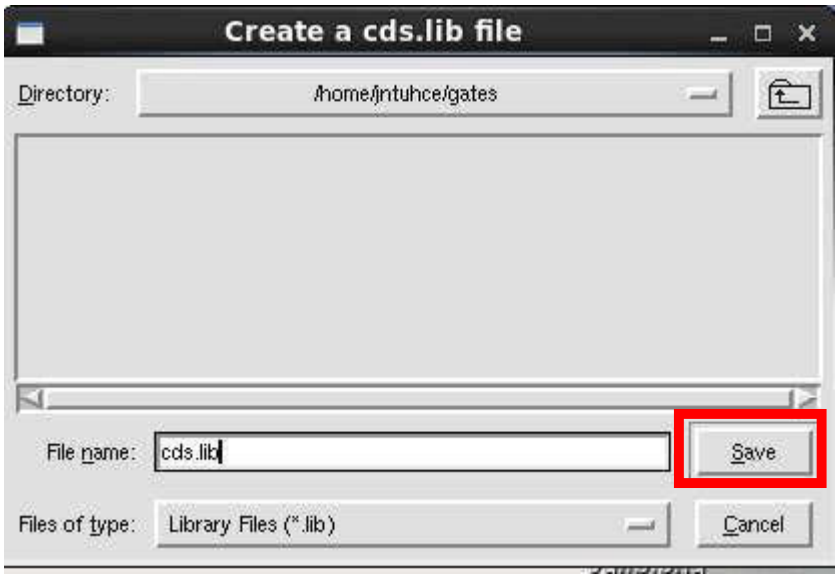
It will invoke the nclaunch window for functional simulation we can compile, elaborate and simulate



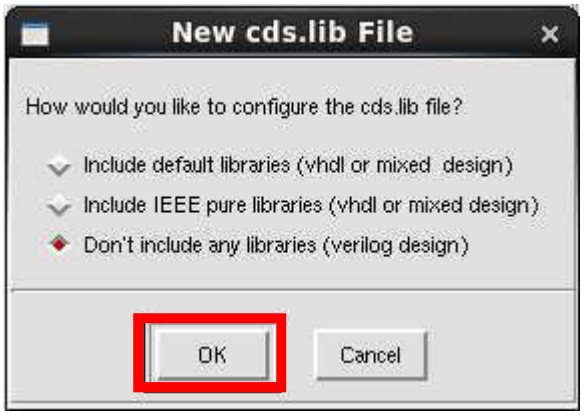
the design using Multistep option.

Create the cds.lib and hdl.var files for to Compile, elaborate and simulate the design and test bench, Click on Create cds.lib File option as shown below.

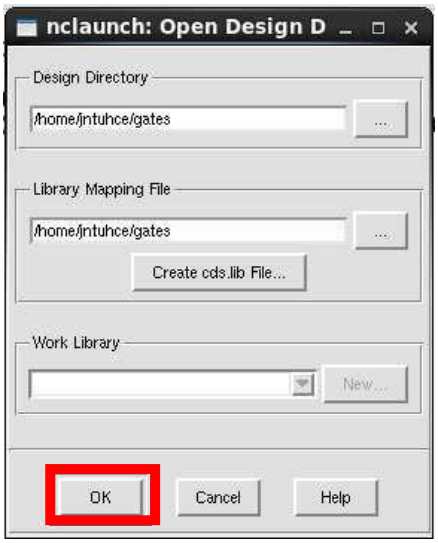




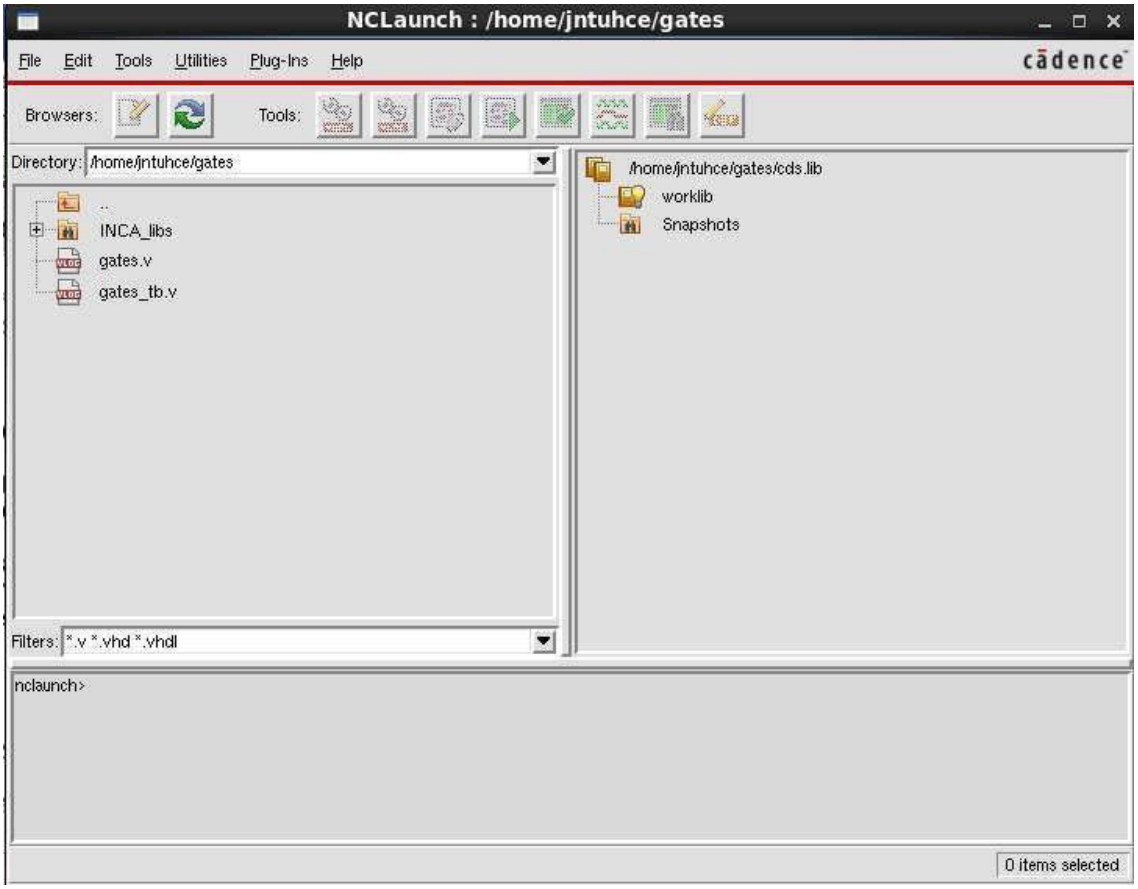
Click save option



Select OK



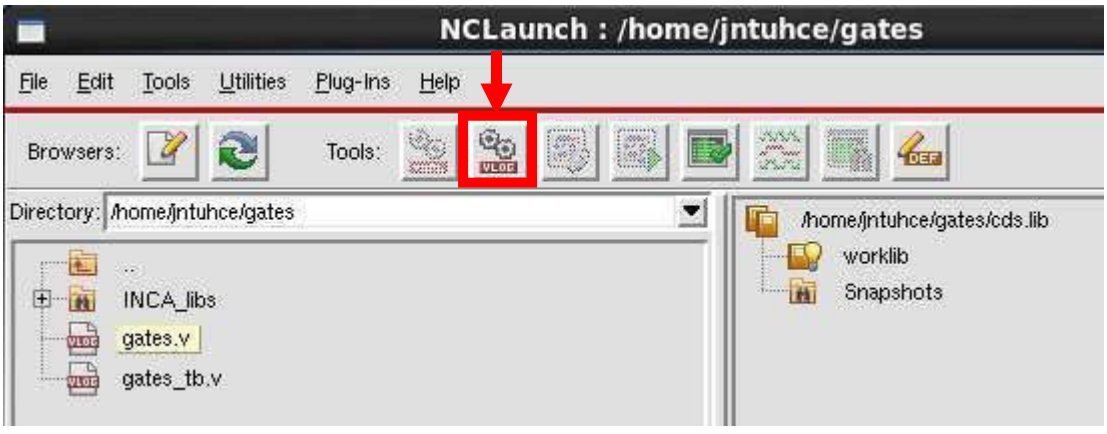
Select ok
You can see the below window after giving ok



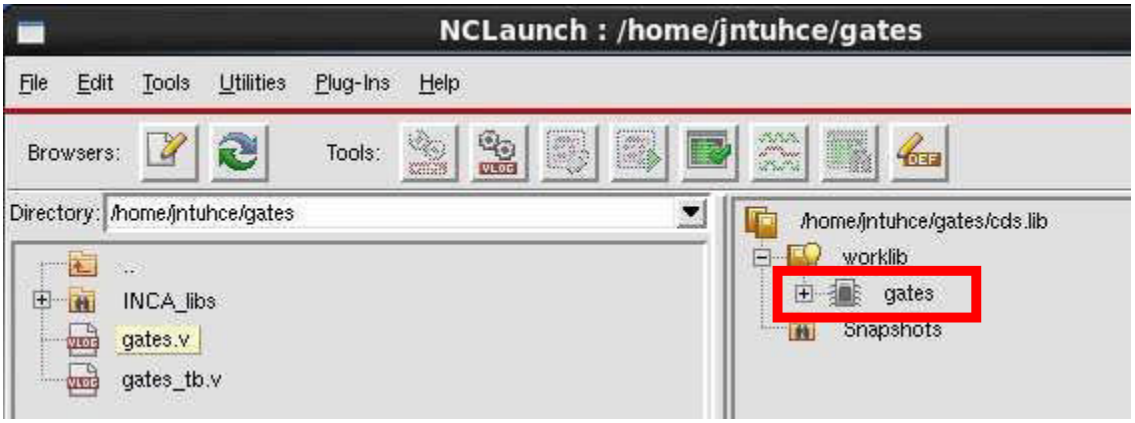
Left side you can see the HDL files, Right side of the window has worklib and snapshots directories listed.

Worklib is the directory where all the compiled codes are stored while Snapshot will have output of elaboration which in turn goes for simulation

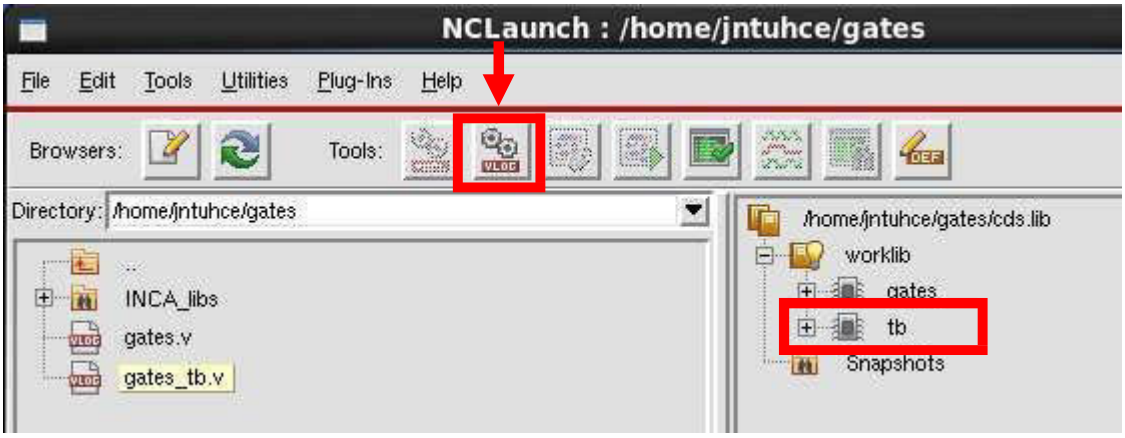
Compilation:



left side select the file and in Tools : launch verilog compiler with current selection will get enable. Click it to compile the code



After compilation it will come under worklib you can see in right side window.

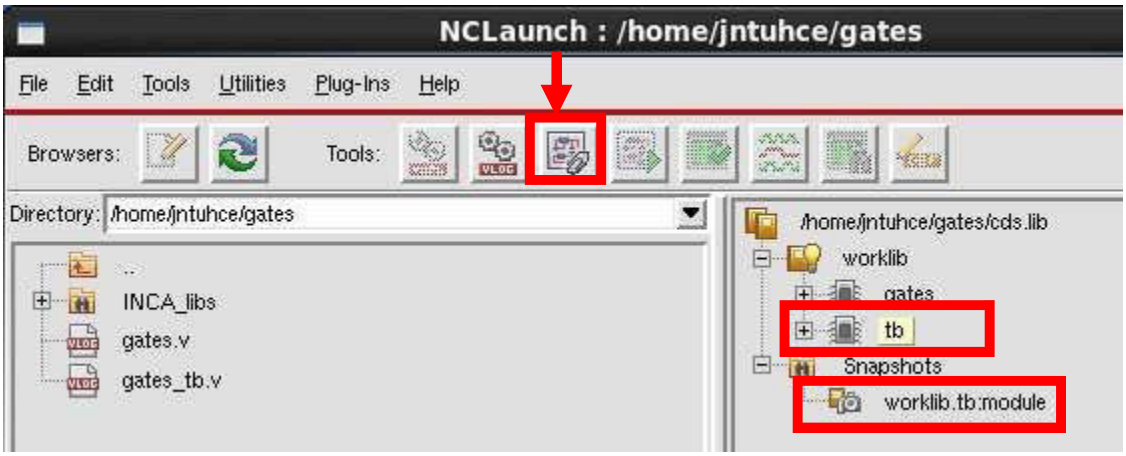


Select the test bench and compile it. It will come under worklib. Under Worklib you can see the module and testbench. Next is to elaborate the design.

Elaboration:

select the testbench file under worklib and in Tools : launch elaborator with current selection will get enable. select the elaborator to elaborate the design.

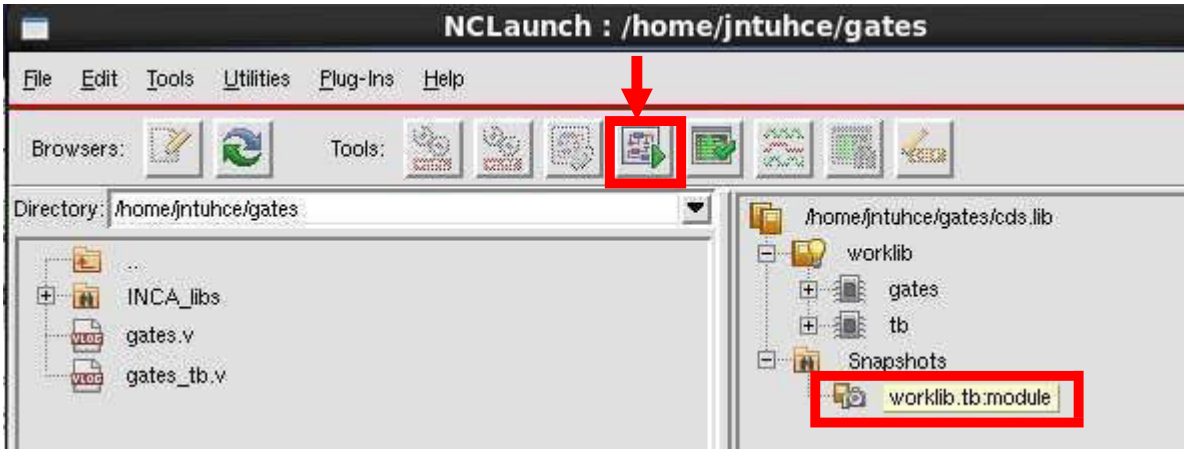
Choose the test bench and elaborate the design



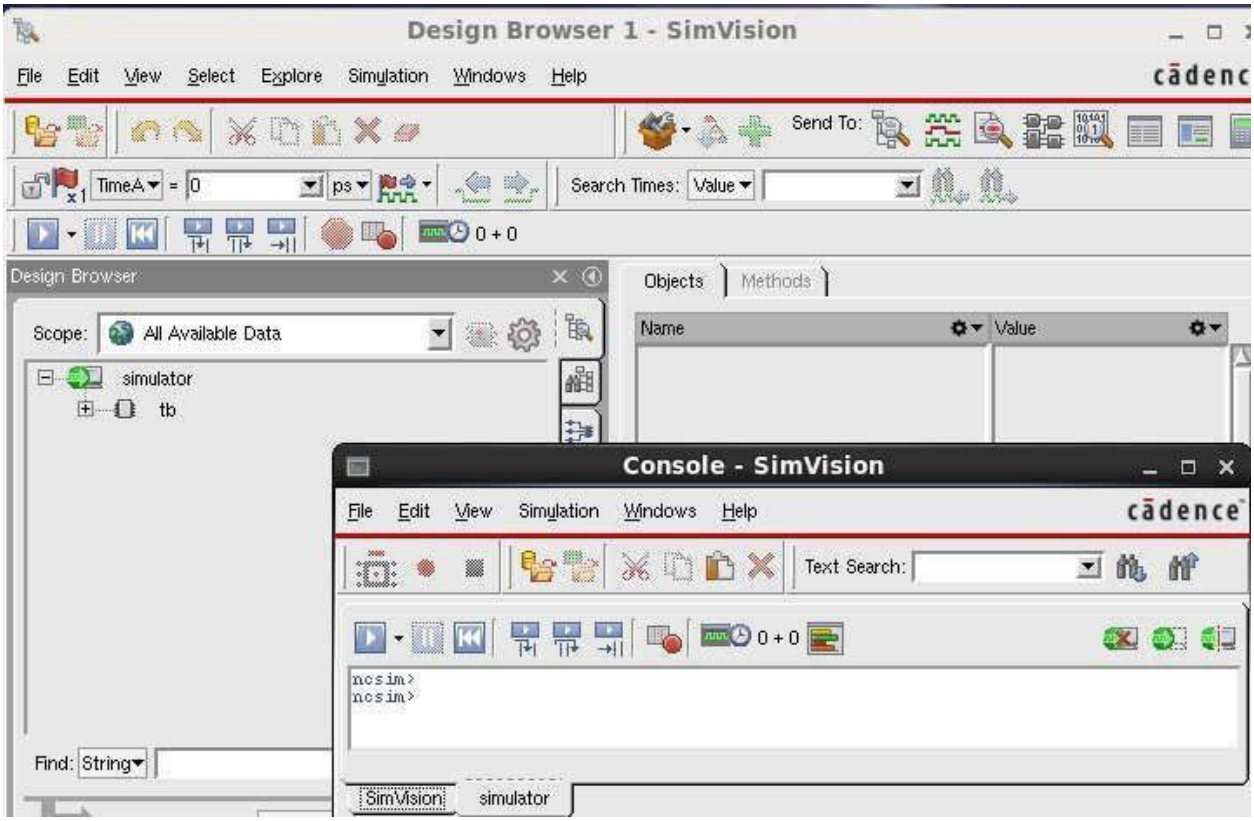
After elaboration the file will come under snapshots.

Simulatiuon:

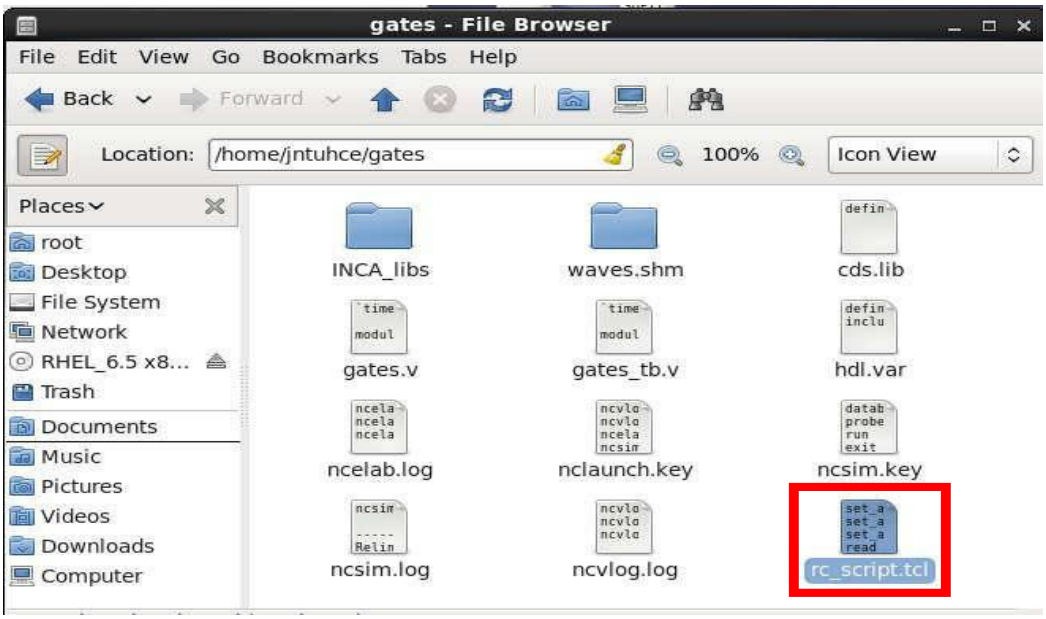
Select the testbench file under snapshot and in Tools : Launch simulator with current selection will get enable.

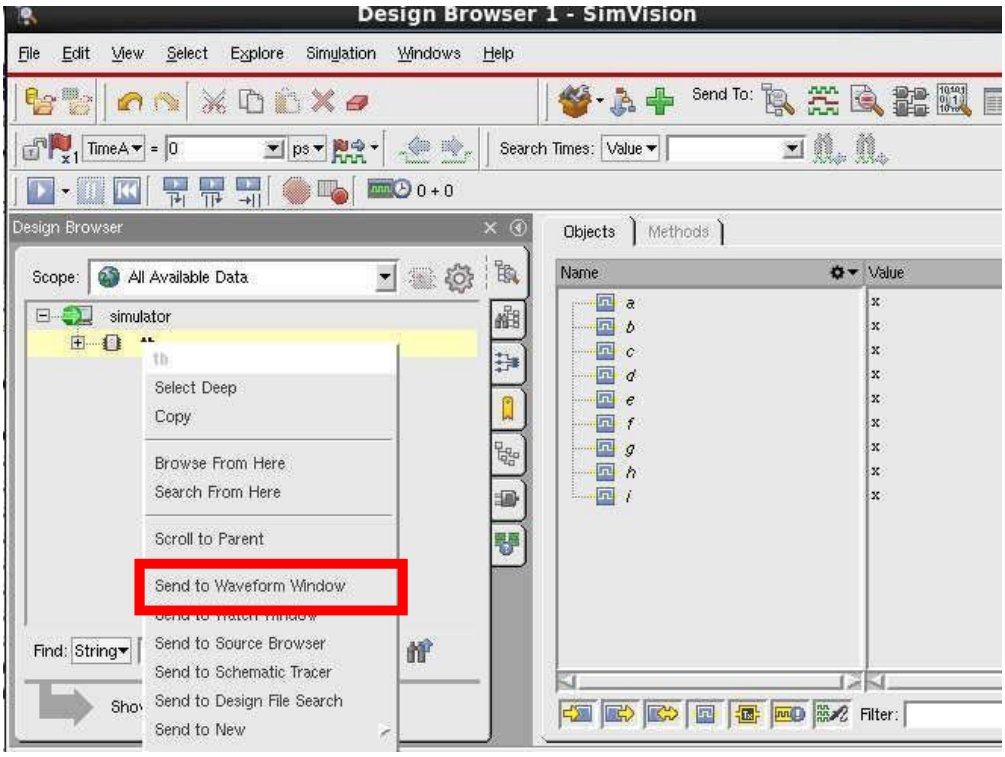


select simulator to simulate the design. After simulation you will get the two windows like below image.



you will get the two windows Design Browser and Simvision .In design browser you can see the test bench in left side window.

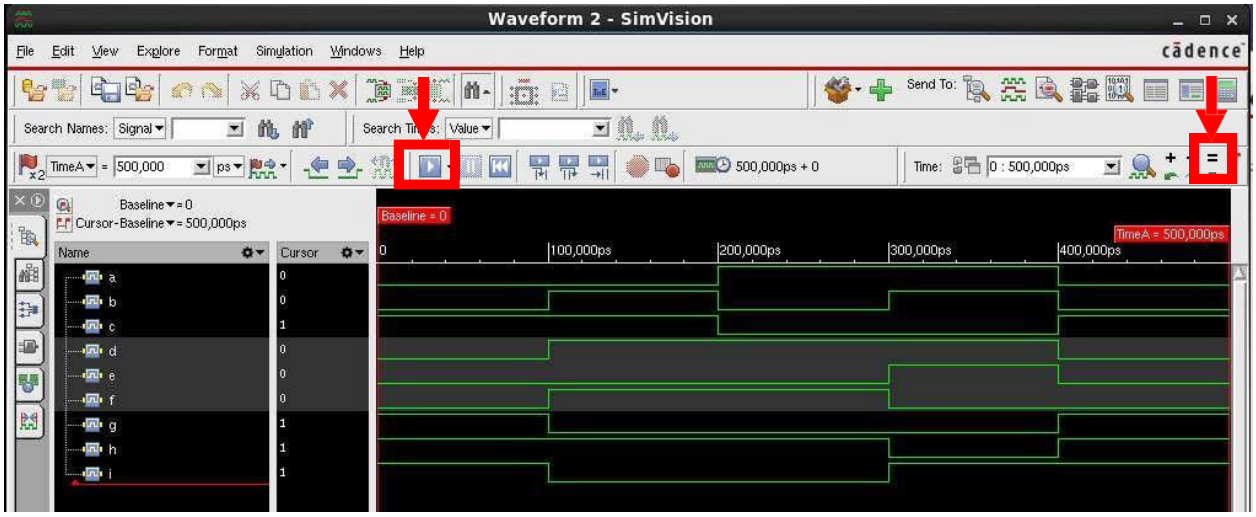




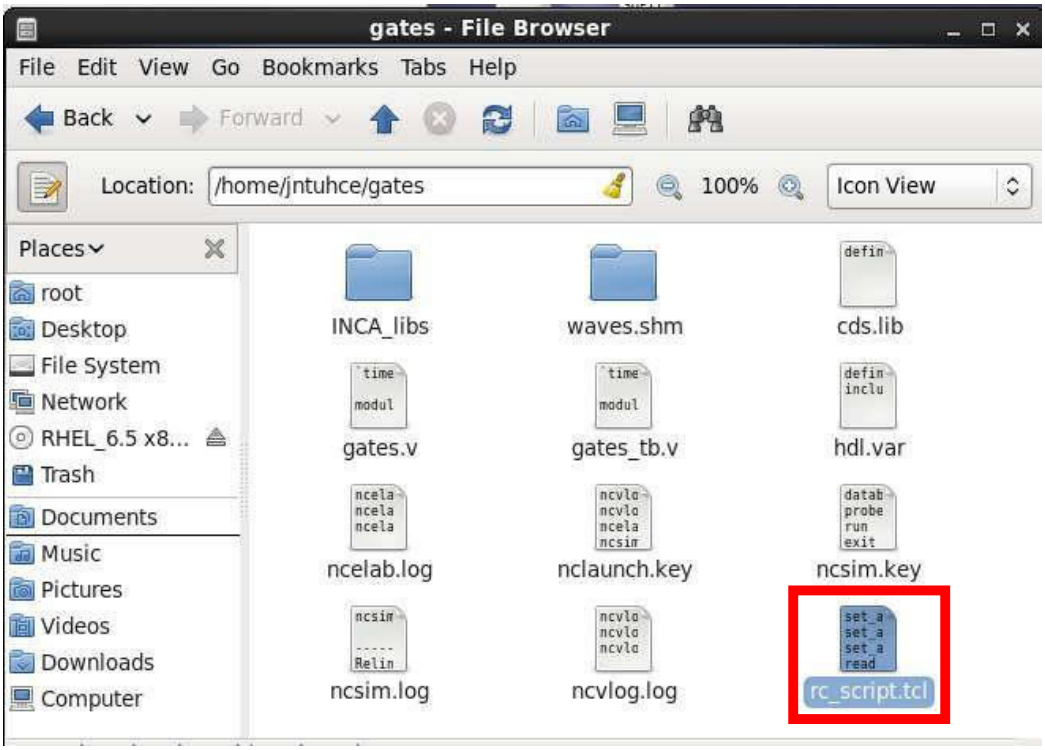
select the test bench for the gates and Right click it. Select the send to waveform window or select the waveform icon

you can see the waveform window after that click the run tool to see the functional simulation for the gates

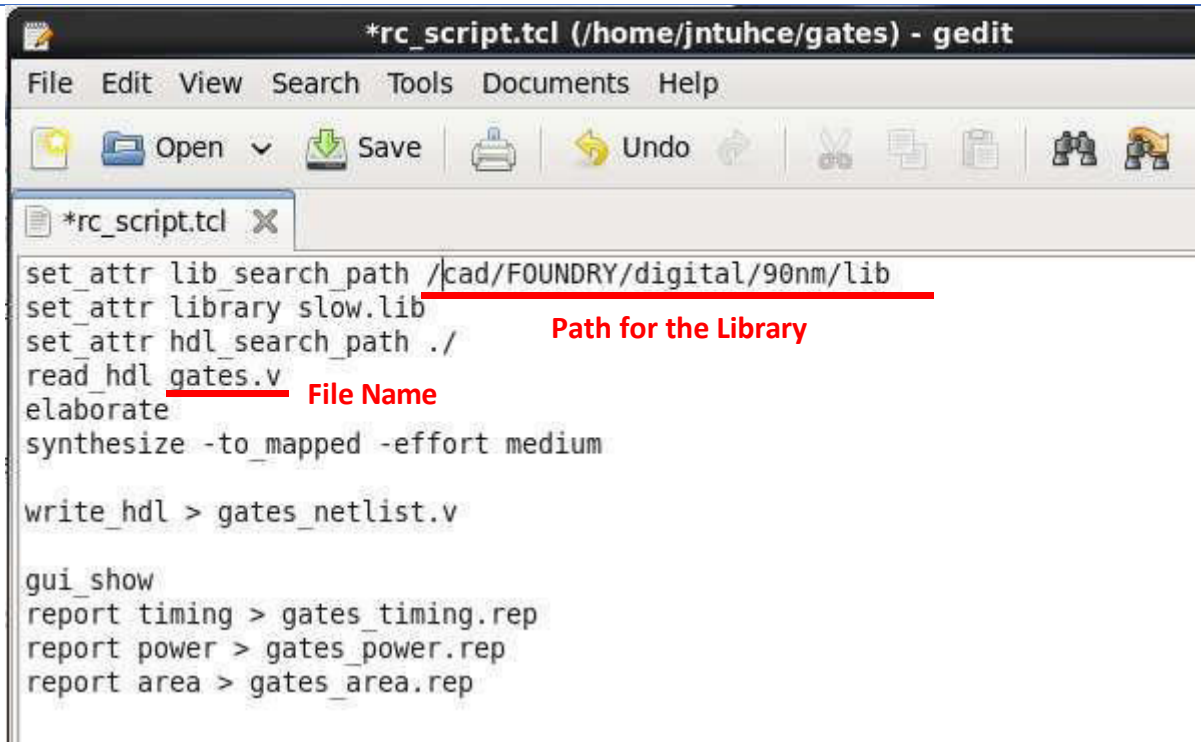
Synthesis Flow:



Synthesis will be done using RTL Compiler. It is a script language called Tool Command Language(TCL)



Inside the run.tcl file we have to mention the commands like below image.

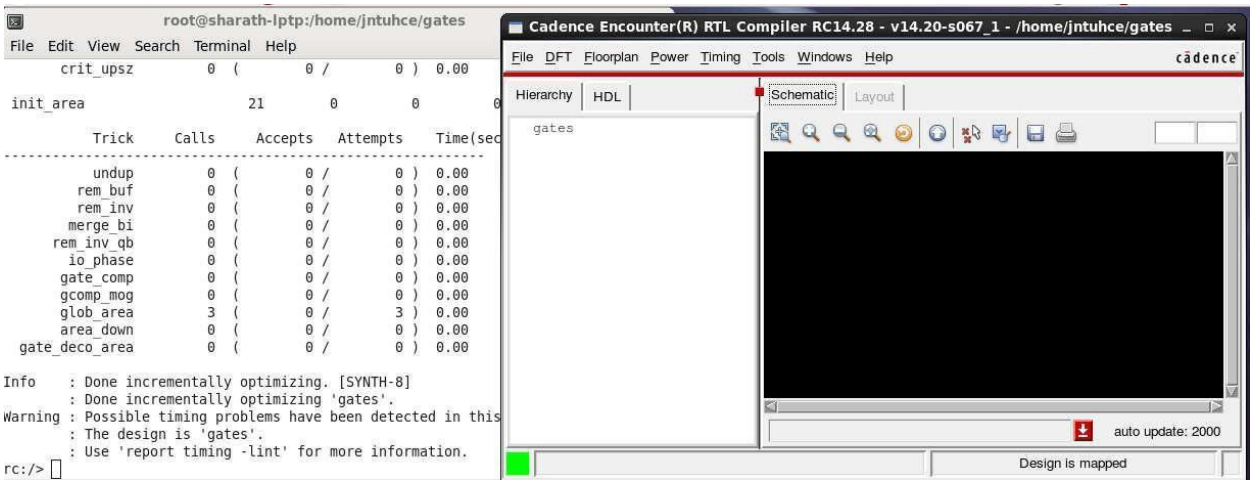


Script file explained below

- Give the path of the library w.r.t to the directory you are in using the command: set_attribute lib_search_path
- Give the path of the RTL files with respect to the directory you are in using the below command: set_attribute hdl_search_path
- Read the library from the directory specified in giving the path for the library files in First line using the command: set_attribute library (slow.lib) is the name of the library file in the directory --library.
- Read the RTL files from the directory specified in the second line. The RTL files are in the directory name : read_hdl gates.v
- Now Elaborate the design using : elaborate command.
- Synthesize the circuit using the command: synthesize -to_mapped -effort medium.
- Timing could be check using : report timing. □ Similarly for Gates : report gates.
- Check area using : report area.
- Check Power dissipation using : report power. It will generate the reports
- Write the hdl code in terms of library components for the synthesized circuit using the command: write_hdl > gates_netlist.v

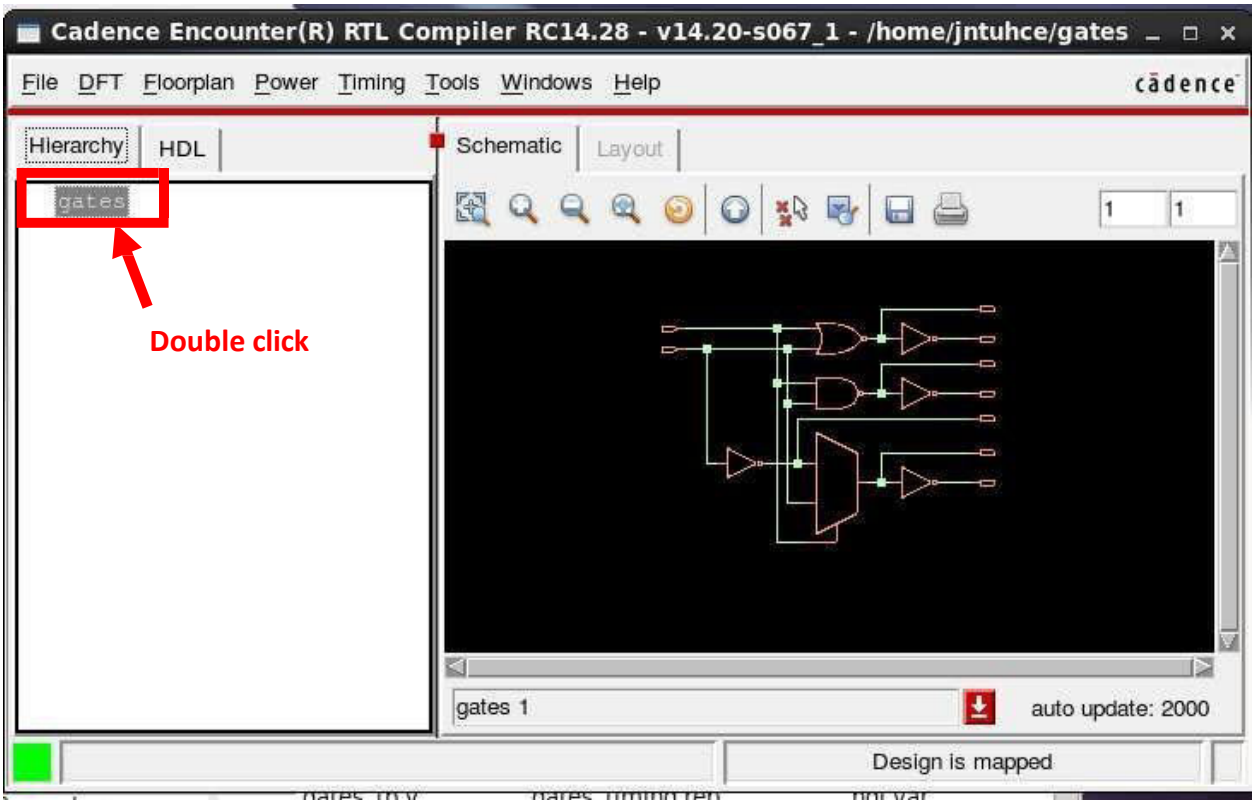
Invoke RTL Compiler by typing below command on your terminal window. The below picture can be seen after typing the above command

rc -f rc_script.tcl -gui



Now open gates_timing.rep file to observe the timing information gates_power.rep file to observe the power

gates_area.rep file to observe the area information.



This will show the RTL schematic of the netlist generated.

EXPERIMENT: 1
LOGIC GATES

AIM: To design, simulate and synthesis the Logic gates using Verilog HDL.

TOOL: Cadence Tool.

THEORY:

Logic Gates are circuits made up of transistors, diodes, and resistors. Logic gates process one or more input signals in a logical fashion. Depending on the input value or voltage, the logic gate will either output a value of '1' for O Nor a value of '0' for OFF.

Logicgates allow simplification of circuit operation. A basic understanding of logic gates will aid technicians in electrical diagnosis.

AND' gates are like two or more switches in series. All the switches haveto be closed ('ON' or a value of '1') in order to make the lamp (output C) turn on.

All input values to the AND gate must be a '1' in order for the output valuetobe'1'. Any other input combinations will result in a 'zero' as the output as shown in the truth table above.

VERILOG CODE:

Module Definition (Gates)

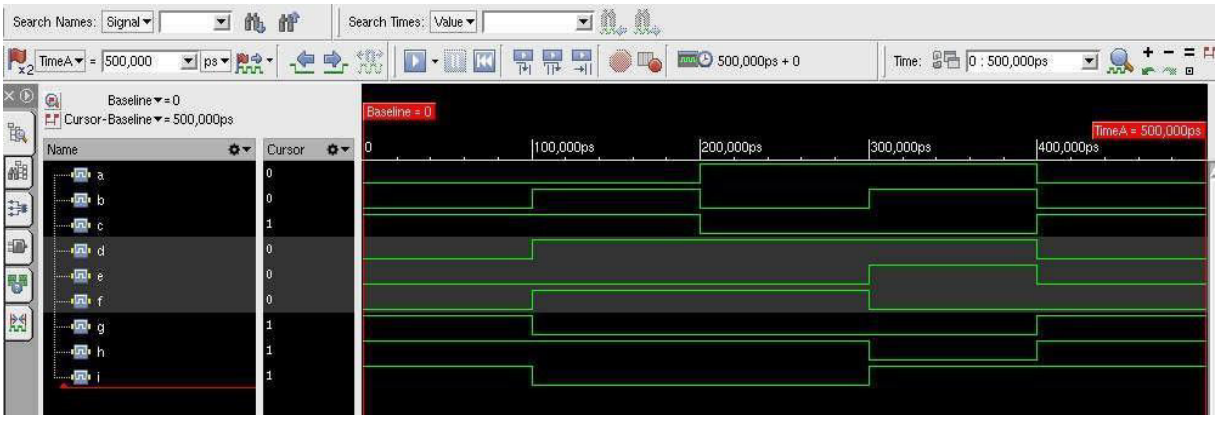
```
module gates (input a, b, output c, d, e, f, g, h, i);
assign c = ~a;      // NOT gate
assign d = a | b;  // OR gate
assign e = a &b;  // AND gate
assign f = a ^ b;  // EX-OR gate
assign g = ~(a | b); // NOR gate
```

```
assign h = ~(a & b); // NAND gate
assign i = ~(a ^ b); // EX-NOR gate
endmodule
```

VERILOG TEST BENCH

```
`timescale 1ns/1ps
module tb();
reg a, b;
wire c, d, e, f, g, h, i;
// Instantiate the gates module
gates uut (.a(a), .b(b), .c(c), .d(d), .e(e), .f(f), .g(g), .h(h), .i(i));
initial begin
// Test cases
                a = 0; b = 0;
               #100; a = 0; b = 1;
               #100; a = 1; b = 0;
               #100; a = 1; b = 1;
               #100; a = 0; b = 0;
#100;
end
endmodule
```

SIMULATION RESULTS:



SYNTHESIS TCL SCRIPT:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib set_attrlibraryslow.lib

set_attr hdl_search_path ./ read_hdlgates.v

elaborate

synthesize -to_mapped -effort medium

write_hdl>gates_netlist.v

gui_show

report timing > gates_timing.repo

report power > gates_power.repo

reportarea>gates_area.repo
```

RESULT:

Hence the Logic Gates has been synthesized and simulated using Cadence Simulator.

VIVA QUESTIONS:

1. Explain what is a combinational circuit?
2. Write two characteristics of combinational circuits.
3. Explain what is a half-adder?
4. Explain what is a full-adder?
5. Explain what is a flip-flop?
6. Explain what is a latch?
7. Explain what are the basic logic elements?
8. Explain what is a truth table?
9. Define positive logic and negative logic.
10. Explain what is a pulse logic system?
11. Explain what is an inverter?
12. Explain what are the universal logic gates?
13. Explain what is the specialty of NAND and NOR gates?
14. Explain why NAND-NAND realization is preferred over AND-OR realization?
15. Explain why is a two-input NAND gate called a universal gate?
16. Explain what is associative law?
17. What are the electronic circuits that operate on one or more input signals to produce standard output?
18. What are the basic building blocks of all circuits in a computer?
19. Which gate gives the output as 1 only if all the input signals are 1?
20. Write the Boolean expression of an OR gate.
21. Which gate is used to reverse the output obtained?
22. Which of the following gates will give a 0 when both of its inputs are 1?
23. When logic gates are connected to form a gating/logic network, it is called a logic circuit.
24. Which are the universal gates that can be used to implement any Boolean expression?
25. Which gate is called an inverter?

EXPERIMENT: 2
ADDER

AIM: To design, simulate, and synthesize 8-Bit Adder using Verilog HDL.

TOOL: Cadence Tool.

THEORY:

Half Adders: A Half Adder is a combinational circuit used to add two single bits. It produces two outputs:

Sum: This output represents the sum of the two input bits.

Carry: This output represents the carry bit, which is passed to the next significant bit.

Full Adders: A Full Adder is a combinational circuit used to add three input bits: two da- ta bits and one carry bit from a previous addition. It produces:

Sum: The sum of the three bits.

Carry: The carry bit, which is passed to the next higher bit.

Full Adder Implementation Using Half Adders: The Full Adder can be implemented using two Half Adders and one OR gate. Here's how:

1. **First Half Adder**: Adds the two data bits and generates a sum and a carry.
2. **Second Half Adder**: Adds the sum from the first Half Adder with the carry-in bit and generates a new sum and carry.
3. **OR Gate**: The final carry is the OR of the two carry outputs from the Half Adders.

The main advantage of using a Full Adder over a Half Adder is that the Full Adder is capable of adding three input bits at a time, which is important for multi-bit addition operations in computers.

VERILOG CODE:

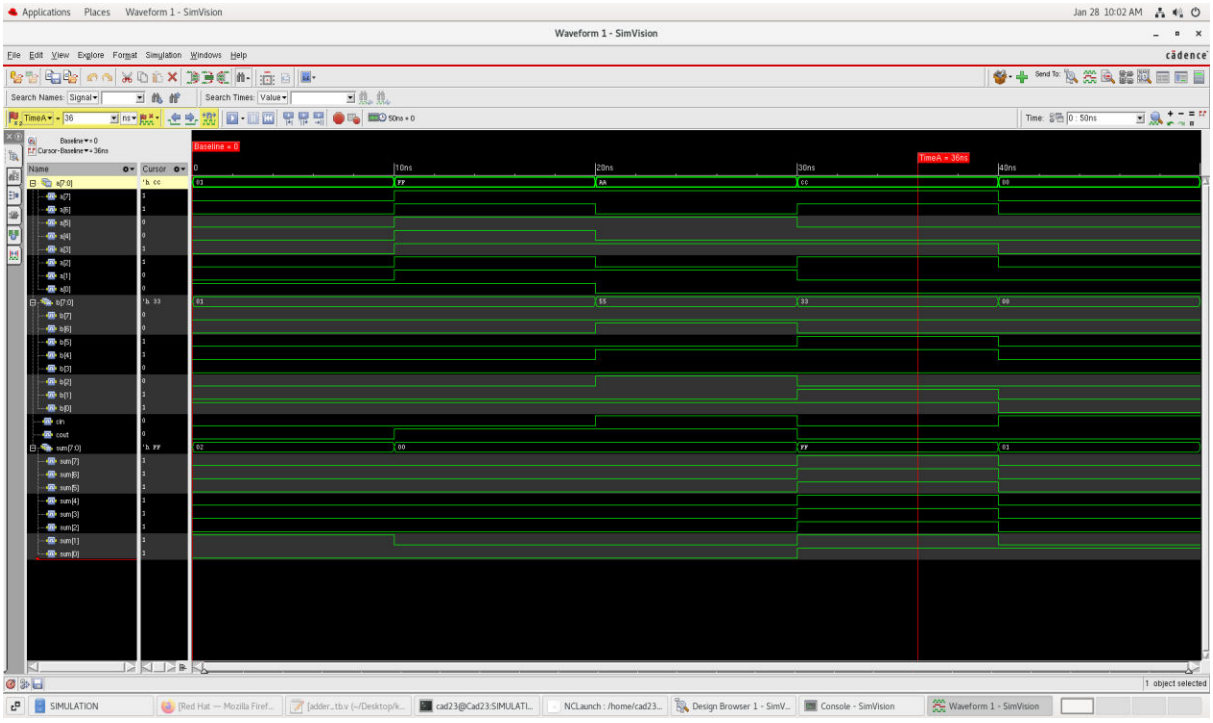
```
module adder(
input [7:0] a,          // 8-bit input A
input [7:0] b,          // 8-bit input B
input cin,              // Carry-in input
output [7:0] sum,       // 8-bit sum output
output cout             // Carry-out
output
);
assign {cout, sum} = a + b + cin; // Perform addition with carry

endmodule
```

VERILOG TEST BENCH:

```
module adder_tb;

reg [7:0] a;            // Test input A

reg [7:0] b;            // Test input B

reg cin;                // Test carry-in input

wire [7:0] sum;         // Test sum output

wire cout;              // Test carry-out output

// Instantiate the 8-bit adder module adder

uut (

.a(a),

.b(b),

.cin(cin),

.sum(sum),
```

```
.cout(cout));

initial begin

 // Apply test cases

 a = 8'b00000001; b = 8'b00000001;

 cin = 0; #10; // Test case 1

 a = 8'b11111111; b = 8'b00000001; cin = 0; #10; // Test case 2

 a = 8'b10101010; b = 8'b01010101; cin = 1; #10; // Test case 3

 a = 8'b11001100; b = 8'b00110011; cin = 0; #10; // Test case 4

 a = 8'b00000000; b = 8'b00000000; cin = 1; #10; // Test case 5

 $stop;    // End the simulation end

 endmodule
```

SIMULATION RESULTS:



SYNTHESIS TCL SCRIPT:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/libset_attrlibraryslow.lib set_attr

hdl_search_path ./read_hdladder.v

elaborate

synthesize -to_mapped -effort mediumwrite_hdl>adder_netlist.v

gui_show

report timing > adder_timing.repo

report power > adder_power.repo

reportarea>adder_area.repo
```

RESULT:

Hence the ADDER has been synthesized and simulated using Cadence Simulator.

VIVA QUESTIONS:

1. What are the universal gates? Why are they called so?
2. Realize the EX-OR gates using minimum number of NAND gates.
3. Give the truth table for EX-NOR and realize using NAND gates.
4. What are the logic low and high levels of TTL IC's and CMOS IC's?
5. Compare TTL logic family with CMOS family.
6. Which logic family is fastest and which has low power dissipation?
7. What are the different methods to obtain minimal expression?
8. What is a Minterm and Maxterm?
9. State the difference between SOP and POS.
10. What is K-map? Why is it used?
11. What do you mean by Logic Gates?
12. What are the applications of Logic Gates?
13. What is a Truth Table?
14. Why do we use basic logic gates?
15. Write down the truth table of all logic gates.
16. What do you mean by universal gate?
17. Write truth table for 2-input NOR, NAND gate.
18. Implement all logic gates by using universal gates.
19. Why are they called universal gates?
20. Give the name of the universal gate.
21. Draw the circuit diagram of Half Adder circuit.
22. Draw the circuit diagram of Full Adder circuit.
23. Draw the full adder circuit by using Half Adder circuit and minimum number of logic gates.
24. Write Boolean function for Half Adder.

EXPERIMENT: 3
MULTIPLIER

AIM: To design and implement a 4-bit Multiplier using Verilog HDL.

TOOL: Cadence Tool.

THEORY:

A **4-bit multiplier** is a combinational digital circuit used to multiply two 4-bit binary numbers. Let the inputs be:

Multiplicand: $A = A_3A_2A_1A_0$

Multiplier: $B = B_3B_2B_1B_0$

The output is an **8-bit product**:

$$P = P_7P_6P_5P_4P_3P_2P_1P_0$$

Working Principle:

The multiplication process is based on the **binary multiplication rule**, similar to decimal multiplication:

Each bit of the multiplier is multiplied with the multiplicand.

This produces **partial products**.

All partial products are then **shifted and added** to obtain the final result.

VERILOG CODE FOR MULTIPLIER

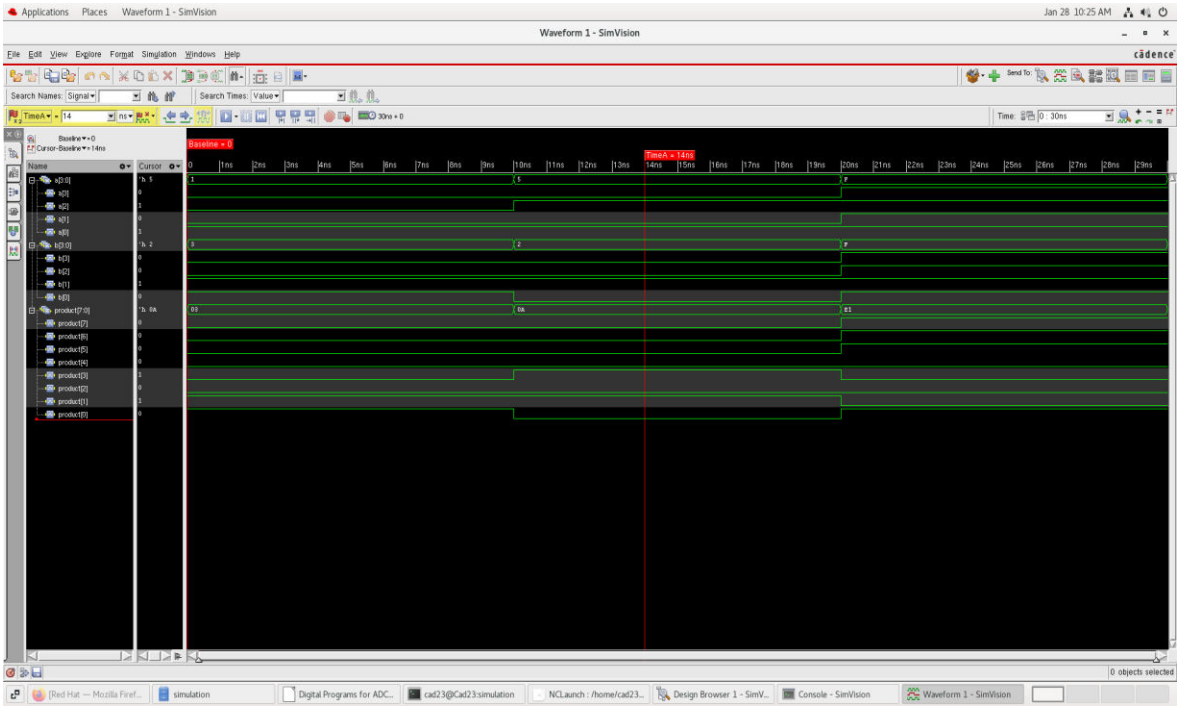
```
module multiplier_4bit ( input [3:0] a, b,
output [7:0] product
);
assign product = a * b;
endmodule
```

TEST BENCH

```
module testbench;
reg [3:0] a, b;
wire [7:0] product;
// Instantiate the multiplier
multiplier_4bit_dataflowuut (.a(a),.b(b),.product(product));
```

```
  initial begin
  $monitor("Time=%0d, a=%b, b=%b, product=%b", $time, a, b, product); a = 4'b0001; b =
  4'b0011; #10; // Test case 1: 1 * 3
  a = 4'b0101; b = 4'b0010; #10; // Test case 2: 5 * 2
  a = 4'b1111; b = 4'b1111; #10; // Test case 3: 15 * 15
$finish;
end
endmodule
```

SIMULATION RESULTS:



SYNTHESIS TCL SCRIPT:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/libset_attrlibraryslow.lib set_attr
hdl_search_path ./read_hdlmux.v
elaborate

synthesize -to_mapped -effort mediumwrite_hdl>mux_netlist.v gui_show
report timing > mux_timing.repo report power > mux_power.repo
reportarea>mux_area.repo
```

RESULT:

Hence the 4 Bit multiplier has been synthesized and simulated using Cadence Simulator.

VIVA QUESTIONS:

1. What is a counter?
2. A counter circuit is usually constructed of what?
3. What is the maximum possible range of bit-count specifically in an n-bit binary counter consisting of 'n' number of flip-flops?
4. How many types of counters are there?
5. Ripple counters are also called....
6. What is the difference between combinational logic and sequential logic?
7. What is an asynchronous counter?
8. How is it different from an asynchronous counter?
9. What are synchronous counters?
10. What is an excitation table?
11. A counter circuit is usually constructed of...
12. What is the maximum possible range of bit-count specifically in an n-bit binary counter consisting of 'n' number of flip-flops?
13. How many types of counters are there?
14. A decimal counter has..... states.
15. Ripple counters are also called.....
16. Synchronous counter is a type of.....
17. A three-decade counter would have.....
18. BCD counter is also known as.....
19. The parallel outputs of a counter circuit represent the....
20. What advantages do synchronous counters have over asynchronous counters?
21. The combinational element of a counter can be made with...
22. What types of flip-flops can be used to implement the memory elements of a counter?
23. What are the advantages of using a microprocessor to implement a counter rather than the conventional method (flip-flops and logic gates)?
24. What is the principal advantage of Gray Code over straight (conventional) binary?

EXPERIMENT: 4
ENCODER AND DECODER

AIM: To design, simulate and synthesis an 8-to-3 Encoder (with and without priority) and a 2-to-4 Decoder using Verilog HDL.

TOOL: Cadence Tool.

THEORY:

1. 8-to-3 Encoder without priority

An 8-to-3 encoder is a combinational circuit that converts 8 input lines into a 3-bit binary output. At any time, only one input is assumed to be HIGH.

Inputs and Outputs

Inputs: $D_0, D_1, D_2, \dots, D_7$

Outputs: Y_2, Y_1, Y_0

Working Principle

The encoder outputs the binary equivalent of the active input line.

Example:

If $D_3 = 1$, output = 011

If $D_5 = 1$, output = 101

2. 8-to-3 Priority Encoder

A priority encoder resolves the ambiguity by assigning priority to higher-order inputs.

Working Principle

The encoder checks inputs from highest priority (D_7) to lowest (D_0).

The output corresponds to the highest-priority active input, ignoring others.

Example:

If $D_7 = 1$and $D_3 = 1$, output = 111(since D_7has higher priority)

Additional Feature

Often includes a valid output (V) to indicate that at least one input is active.

3. 2-to-4 Decoder

A 2-to-4 decoder is a combinational circuit that converts 2-bit input into 4 output lines, where only one output is HIGH at a time.

Inputs and Outputs

Inputs: A_1, A_0

Outputs: Y_0, Y_1, Y_2, Y_3

Working Principle

Based on the binary input, one of the outputs is activated.

VERILOG CODE:

1. 8-to-3 Encoder without priority

```
module priority_encoder8to3 (
input [7:0] D,
output reg [2:0] Y,
output reg valid
);

always @(*) begin
valid = 1;
casex (D)
8'b1xxxxxxx: Y = 3'b111;
8'b01xxxxxx: Y = 3'b110;
8'b001xxxxx: Y = 3'b101;
8'b0001xxxx: Y = 3'b100;
8'b00001xxx: Y = 3'b011;
8'b000001xx: Y = 3'b010;
8'b0000001x: Y = 3'b001;
8'b00000001: Y = 3'b000;
default: begin
Y = 3'b000;
valid = 0;
end
endcase
end
endmodule
```

Verilog Test Bench

```
`module tb_encoder8to3;
reg [7:0] D;
wire [2:0] Y;
// Instantiate DUT
encoder8to3 uut (
.D(D),
.Y(Y)
);
```

```
initial begin
$display("Time\tD\t\tY");
$monitor("%0t\t%b\t%", $time, D, Y);
D = 8'b00000001; #10;
D = 8'b00000010; #10;
D = 8'b00000100; #10;
D = 8'b00001000; #10;
D = 8'b00010000; #10;
D = 8'b00100000; #10;
D = 8'b01000000; #10;
D = 8'b10000000; #10;
// Invalid case (multiple inputs HIGH)
D = 8'b10100000; #10;
$finish;
end
endmodule
```

VERILOG CODE:

2. 8-to-3 Encoder with priority

```
module priority_encoder8to3 (
input [7:0] D,
output reg [2:0] Y,
output reg valid
);

always @(*) begin
valid = 1;
casex (D)
8'b1xxxxxxx: Y = 3'b111;
8'b01xxxxxx: Y = 3'b110;
8'b001xxxxx: Y = 3'b101;
8'b0001xxxx: Y = 3'b100;
8'b00001xxx: Y = 3'b011;
8'b000001xx: Y = 3'b010;
8'b0000001x: Y = 3'b001;
```

```
8'b00000001: Y = 3'b000;
default: begin
Y = 3'b000;
valid = 0;
end
endcase
end

endmodule
```

Testbench for 8-to-3 Priority Encoder

```
module tb_priority_encoder8to3;

reg [7:0] D;
wire [2:0] Y;
wire valid;

// Instantiate DUT
priority_encoder8to3 uut (
   .D(D),
   .Y(Y),
   .valid(valid)
);

initial begin
   $display("Time\tD\t\tY\tValid");
   $monitor("%0t\t%b\t%b\t%b", $time, D, Y, valid);

   D = 8'b00000001; #10;
   D = 8'b00000110; #10; // multiple inputs
   D = 8'b00110000; #10;
   D = 8'b10000000; #10;
   D = 8'b01000000; #10;

   // No input active
   D = 8'b00000000; #10;

   $finish;
end

endmodule
```

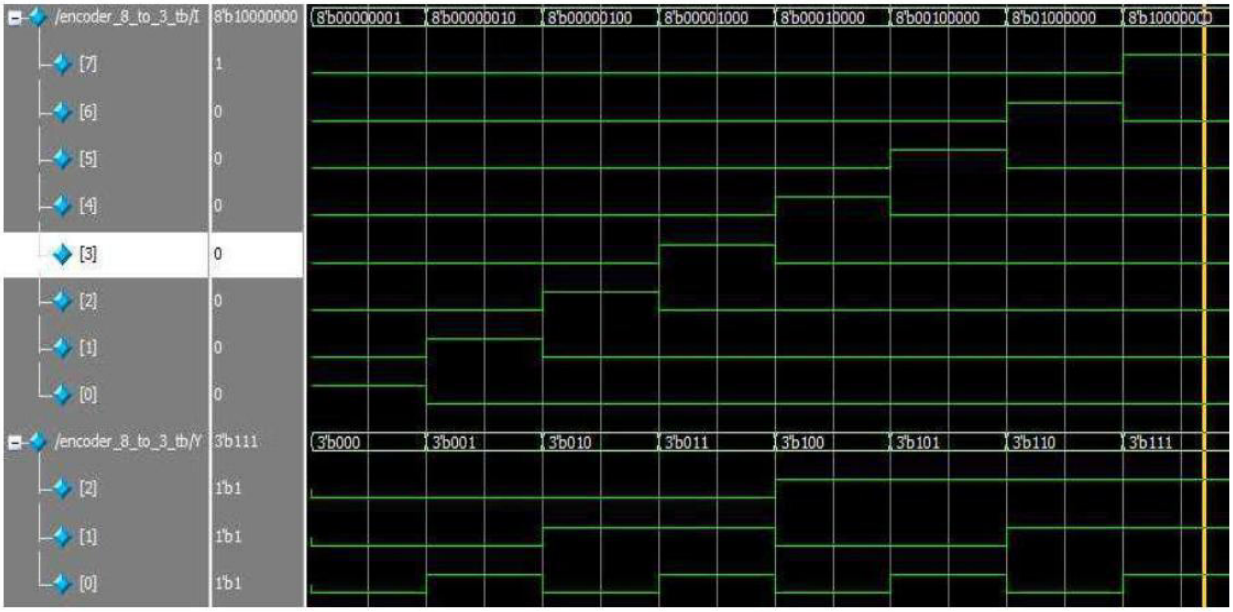
3. 2-to-4 Decoder

```
module decoder2to4 (
input [1:0] A,
output reg [3:0] Y
);
always @(*) begin
case (A)
2'b00: Y = 4'b0001;
2'b01: Y = 4'b0010;
2'b10: Y = 4'b0100;
2'b11: Y = 4'b1000;
endcase
end
endmodule
```

Testbench for 2-to-4 Decoder

```
module tb_decoder2to4;

reg [1:0] A;
wire [3:0] Y;

// Instantiate DUT
decoder2to4 uut (
    .A(A),
    .Y(Y)
);

initial begin
    $display("Time\tA\tY");
    $monitor("%0t\t%b\t%b", $time, A, Y);

    A = 2'b00; #10;
    A = 2'b01; #10;
    A = 2'b10; #10;
    A = 2'b11; #10;

    $finish;
end

endmodule
```

SIMULATION RESULTS:



RESULT:

Hence the 8-to-3 Encoder (with and without priority) and a 2-to-4 Decoder has been synthesized and simulated using Cadence Simulator.

VIVA QUESTIONS:

1. Explain what is a combinational circuit?
2. Write two characteristics of combinational circuits.
3. Explain what is a half-adder?
4. Explain what is a full-adder?
5. Explain what is a flip-flop?
6. Explain what is a latch?
7. Explain what are the basic logic elements?
8. Explain what is a truth table?
9. Define positive logic and negative logic.
10. Explain what is a pulse logic system?
11. Explain what is an inverter?
12. Explain what are the universal logic gates?
13. Explain what is the specialty of NAND and NOR gates?
14. Explain why NAND-NAND realization is preferred over AND-OR realization?
15. Explain why is a two-input NAND gate called a universal gate?
16. Explain what is associative law?
17. What are the electronic circuits that operate on one or more input signals to produce standard output?
18. What are the basic building blocks of all circuits in a computer?
19. Which gate gives the output as 1 only if all the input signals are 1?
20. Write the Boolean expression of an OR gate.
21. Which gate is used to reverse the output obtained?
22. Which of the following gates will give a 0 when both of its inputs are 1?
23. When logic gates are connected to form a gating/logic network, it is called a logic circuit.
24. Which are the universal gates that can be used to implement any Boolean expression?
25. Which gate is called an inverter?

EXPERIMENT: 5
ARITHMETIC LOGIC UNIT

AIM: To design simulate and synthesis the Arithmetic Logic Unit using Verilog HDL.

TOOL: Cadence Tool

THEORY:

In computing, an arithmetic logic unit (ALU) is a combinationaldigital circuit that performs arithmetic and bitwise operations on integerbinary numbers. This is in contrast to a floating-point unit (FPU), which operates on floating point numbers. It is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs).

The inputs to an ALU are the data to be operated on, called operands, and a code indicating the operation to be performed; the ALU's output is the result of the performed operation. In many designs, the ALU also has status inputs or outputs, or both, which convey information about a previous operation or the current operation, respectively, between the ALU and external status registers.

A number of basic arithmetic and bitwise logic functions are commonly supported by ALUs. Basic, general purpose ALUs typically include these operations in their repertoires:

Add with carry: A, B and carry-in are summed and the sum appears at Y and carry-out.

Subtract: B is subtracted from A (or vice versa) and the difference appears at Y and carry-out. For this function, carry-out is effectively a "borrow" indicator. This operation may also be used to compare the magnitudes of A and B; in such cases the Y output may be ignored by the processor, which is only interested in the status bits (particularly zero and negative) that result from the operation.

Subtract with borrow: B is subtracted from A (or vice versa) with borrow (carry- in) and the difference appears at Y and carry-out (borrow out).

Two's complement (negate): A (or B) is subtracted from zero and the difference appears at Y.

Increment: A (or B) is increased by one and the resulting value appears at Y.

Decrement: A (or B) is decreased by one and the resulting value appears at Y.

Pass through: all bits of A (or B) appear unmodified at Y. This operation is typically used to determine the parity of the operand or whether it is zero or negative, or to load the operand into a processor register.

VERILOG CODE:

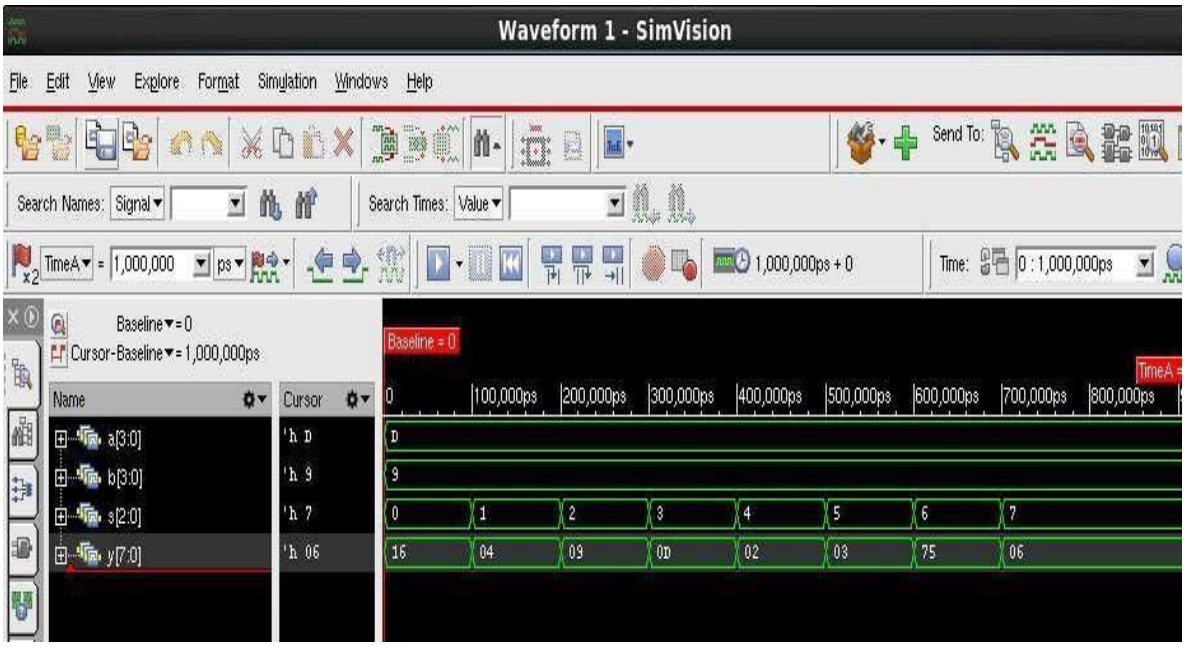
```
modulealu (

input [7:0] a, b,          // 8-bit input operands

input [2:0] op,            // 3-bit operation selector

outputreg [7:0] result // 8-bit result output

);

always @(*) begin case

(op)

3'b000: result = a + b;  // Addition 3'b001: result = a -

b;  // Subtraction 3'b010: result = a & b;  // AND

operation 3'b011: result = a | b;  // OR operation 3'b100:

result = a ^ b;  // XOR operation

3'b101: result = ~a;                    // NOT operation (on 'a')

3'b110: result = a << 1; // Logical left shift 3'b111: result = a

>> 1; // Logical right shift

default: result = 8'b0;  // Default case endcase

end
endmodule
```

VERILOG TEST BENCH:

```
moduletestbench; reg [7:0]

a, b;

reg [2:0] op;

wire [7:0] result;

 // Instantiate the ALU aluuut (

.a(a),

.b(b),
```

```
        .op(op),
        .result(result)
        );
        initial begin
        // Monitor the inputs and output
            $monitor("Time=%0d, a=%b, b=%b, op=%b, result=%b", $time, a, b, op,
        result);
        // Test addition
        a = 8'b00001101; b = 8'b00000011; op = 3'b000; #10; // 13 + 3
        // Test subtraction
        a = 8'b00001101; b = 8'b00000011; op = 3'b001; #10; // 13 - 3
        // Test AND
        a = 8'b10101010; b = 8'b11001100; op = 3'b010; #10; // AND
        // Test OR
        a = 8'b10101010; b = 8'b11001100; op = 3'b011; #10; // OR
          // Test XOR
          a = 8'b10101010; b = 8'b11001100; op = 3'b100; #10; // XOR
          // Test NOT
          a = 8'b10101010; op = 3'b101; #10;                  // NOT a
          // Test left shift
          a = 8'b00001101; op = 3'b110; #10;                  // Shift left
          // Test right shift
          a = 8'b00001101; op = 3'b111; #10;                  // Shift right
          $finish;
    end
endmodule
```

SIMULATION RESULTS:



SYNTHESIS TCL SCRIPT:

set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib set_attrlibraryslow.lib

set_attr hdl_search_path ./read_hdlalu.v elaborate

synthesize-to_mapped-effortmedium

write_hdl > alu_netlist.v

gui_show reporttiming>alu_timing.repo

report power > alu_power.repo

reportarea>alu_area.repo

RESULT:

Hence the Arithmetic Logic Unit has been synthesized and simulated using Cadence Simulator.

VIVA QUESTIONS:

1. What are the three categories of computer architecture?
2. What are some of the components of a microprocessor?
3. What is MESI?
4. What are the different hazards?
5. What is pipelining?
6. What is a cache?
7. What is a snooping protocol?
8. What are the different types of interrupts in a microprocessor system?
9. What is the easiest way to determine cache locations in which to store memory blocks?
10. What is a virtual memory on a computer?
11. Can you state some of the common rules of assembly language?
12. What is a RAID system?
13. What are the two hardware methods to establish a priority? Explain each method.
14. What are flip-flops?
15. What's the difference between interrupt service routine and subroutine?
16. What are the different types of fields that are part of instruction?
17. What are the steps involved in an instruction cycle?
18. What are the five stages in a DLX pipeline?
19. What are the types of micro-operations?
20. What is the write-through method?
21. What is associate mapping?
22. What does wait state mean?
23. What is a DMA?
24. What is a horizontal microcode?
25. What is computer architecture?

EXPERIMENT: 6

4-BIT BINARY-TO GRAY CODE CONVERTER

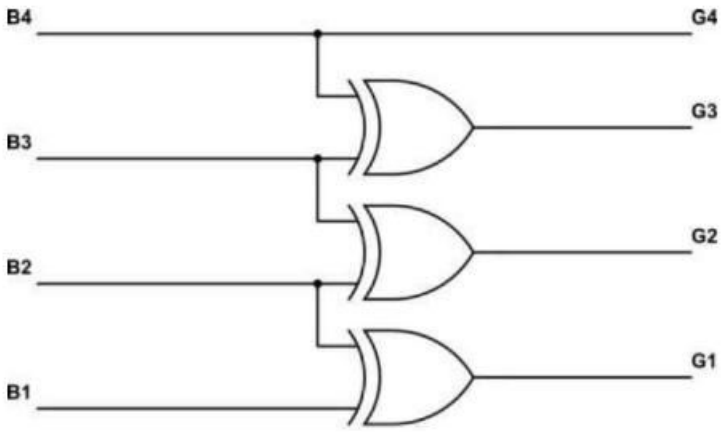
AIM: To design simulate and synthesis 4-bit Binary -to – Gray Code Converter using Verilog HDL.

TOOL: Cadence Tool

THEORY:

The logical circuit which converts binary code to equivalent gray code is known as binary to gray code converter. The gray code is a non weighted code. The successive gray code differs in one bit position only that means it is a unit distance code. It is also referred as cyclic code. It is not suitable for arithmetic operations. It is the most popular of the unit distance codes. It is also a reflective code. An n-bit Gray code can be obtained by reflecting an n-1 bit code about an axis after 2n-1 rows, and putting the MSB of 0 above the axis and the MSB of 1 below the axis. Reflection of Gray codes is shown below. The 4 bits binary to gray code conversion table is
given below

Decimal Number	4 bit Binary Number ABCD	4 bit Gray Code $G_1G_2G_3G_4$
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1
10	1 0 1 0	1 1 1 1
11	1 0 1 1	1 1 1 0
12	1 1 0 0	1 0 1 0
13	1 1 0 1	1 0 1 1
14	1 1 1 0	1 0 0 1
15	1 1 1 1	1 0 0 0



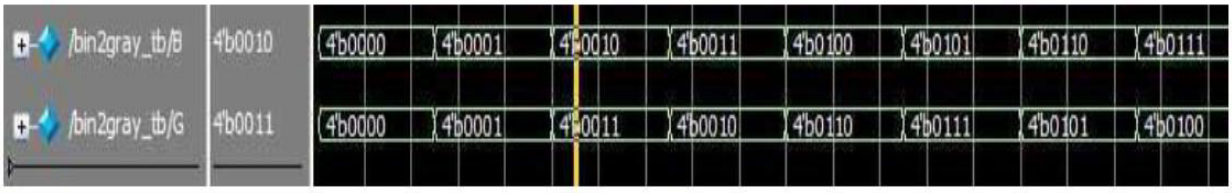
VERILOG CODE:

```
module bin2gray ( B ,G );
input [3:0] B ;
wire [3:0] B ;
output [3:0] G ;
wire [3:0] G ;
assign G[3] = B[3];
assign G[2:0] = B[3:1] ^ B[2:0];
endmodule
```

VERILOG TEST BENCH:

```
module bin2gray_tb;
reg [3:0]B;
wire [3:0]G;
bin2gray B2G (.B(B) ,.G(G));
always #5 B=B+1'b1;
initial
begin
$monitor($time, "\tB=%b\t , G=%b\t", B,G);
B <= 4'b0000;
#80 $finish;
end
endmodule
```

SIMULATION RESULTS:



RESULT:

Hence the 4-bit Binary -to – Gray Code Converter has been synthesized and simulated using Cadence Simulator.

VIVA QUESTIONS:

1. What is Gray code?
2. Why is Gray code called a unit distance code?
3. What is a Binary to Gray code converter?
4. Why do we use Gray code instead of binary code?
5. What is the main advantage of Gray code?
6. Where is Gray code commonly used?
7. How many bits change between consecutive Gray code values?
8. What is the difference between Binary code and Gray code?
9. Is Gray code a weighted code?
10. What is the MSB rule in Binary to Gray conversion?
11. What is the formula for converting binary to Gray code?
12. How is the MSB of Gray code obtained?
13. How are the remaining bits of Gray code calculated?
14. Convert binary 1011 into Gray code.
15. Convert binary 1101 into Gray code.
16. Write the truth table for a 3-bit Binary to Gray converter.
17. Write the truth table for a 4-bit Binary to Gray converter.
18. Which logic operation is used in Binary to Gray conversion?
19. Why is XOR gate used in the conversion?
20. Write Boolean expressions for a 4-bit Binary to Gray converter.
21. Draw the logic circuit for a 3-bit Binary to Gray converter.
22. Draw the logic circuit for a 4-bit Binary to Gray converter.
23. How many XOR gates are required for an n-bit converter?
24. Can Binary to Gray converter be implemented using only XOR gates?
25. What is the hardware complexity of the converter?

EXPERIMENT: 7
UNIVERSAL SHIFT REGISTER

AIM: To design simulate and synthesis the Universal Shift Register using Verilog HDL.

TOOL: Cadence Tool

THEORY:

A Universal Shift Register is a register which can shift its data in both direction i.e. left and right directions. In other Words , a universal shift register is a bidirectional shift register .

A universal shift register is combination of design of bidirectional shift register and a unidirectional shift register with the parallel load provisions.

A universal shift register can perform parallel to serial operation (first loading parallel input and then shifting.

A universal shift register can also perform serial to parallel operation (first shifting and then retrieving parallel output .

The desired operation is then specified by a 2 bit control signal.

Types of Shift Register

There are several types of shift registers, each with its own unique characteristics and applications. Here are the most common types of shift registers:

Serial-In, Serial-Out (SISO) Shift Register: This type of shift register has one data input and one data output, both of which are shifted in a serial manner. This type of register is commonly used for data conversion and data transmission applications.

Serial-In, Parallel-Out (SIPO) Shift Register: This type of shift register has one data input and several data outputs. The input data is shifted in a serial manner, and when the shift operation is complete, the data is available at all output pins simultaneously. This type of register is used for applications that require parallel data transfer.

Parallel-In, Serial-Out (PISO) Shift Register: This type of shift register has several data inputs and one data output. The data is loaded in parallel into the register, and

then shifted out in a serial manner. This type of register is used for applications that require serial data output from a parallel input.

Parallel-In, Parallel-Out (PIPO) Shift Register: This type of shift register has several data inputs and several data outputs. The data is loaded in parallel into the register, and when the shift operation is complete, the data is available at all output pins simultaneously. This type of register is used for applications that require both parallel data input and output.

Bidirectional Shift Register: This type of shift register can shift data in both directions, from left to right and from right to left. This type of register is used for applications that require both shifting and shifting back of the data.

Universal Shift Register: This type of shift register can perform any of the above operations by using control signals. It can shift data in any direction and perform any of the above operations. This type of register is used for applications that require flexibility in data transfer and manipulation.

Each type of shift register has its own unique characteristics and applications, and choosing the right type of register is essential for the success of any digital circuit design.

VERILOG CODE:

```
module shift_register( input clk, //
clock input

input rst, // reset input input d_in, //

data input

input [1:0] sel; //for case statement output [3:0]

d_out // data output

);

always @(posedge clk or posedge rst) begin if (rst)

begin

d_out <= 4'b0000;
```

```
end else
case (sel)
2'b00: d_out <= 4'b0;
2'b01: d_out<={d_out[3:1], din}; //left shift 2'b10:
d_out<={din, d_out[2:0]}; //right shift 2'b11:  d_out <=
4'b0;
default:  d_out <= d_in; endcase
end endmodule
```

OR

```
`timescale 1ns / 1ps
module universal_shift_register (clr,clk,sel,parin,out); input
clr,clk;
input [1:0]sel; input
[3:0]parin; output
reg[3:0]out;
always @(posedge clk) begin
if(clr) out=4'b0000; else
begin case(sel)
2'b00: out=out;
2'b01: out={parin[0],parin[3:1]};
2'b10: out={parin[2:0],parin[3]}; 2'b11:
out=parin;
endcase end
end endmodule
```

VERILOG TEST BENCH:

```
module unsrtb; reg
[3:0]parin; reg clr;
reg clk;
reg [0:1]sel;
wire [3:0]out;
universal_shift_register uut(.parin(parin),.clr(clr),.clk(clk),.sel(sel),.out(out)); initial  begin
clk=0; repeat(100)
#20 clk=~clk; end
initial  begin
parin=4'b1011; sel=2'b01;
clr=1'b0;
#40;
parin=4'b1011; sel=2'b10;
clr=1'b0;
#40;
parin=4'b1011; sel=2'b11;
clr=1'b0;
#40;
end endmodule
```

SIMULATION RESULTS:

SYNTHESIS TCL SCRIPT:

set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib set_attrlibraryslow.lib

set_attr hdl_search_path ./read_hdlalu.v elaborate

synthesize-to_mapped-effortmedium

write_hdl > alu_netlist.v

gui_show reporttiming>alu_timing.repo

report power > alu_power.repo

reportarea>alu_area.repo

RESULT:

Hence the Universal Shift Register has been synthesized and simulated using Cadence Simulator.

VIVA QUESTIONS:

Q.1. What do you mean by serial data?
Q.2. What do you mean by parallel data?
Q.3. What do you mean by serial data transfer?
Q.4. What is a single bit register?
Q.5. What is a register?
Q.6. What do you mean by the storage capacity of a register?
Q.7. What do you mean by loading a register?
Q.8. What are the types of loading the registers?
Q.9. What is serial loading?
Q.10. What is parallel loading?
Q.11. How does a register output data?
Q.12. What is the serial output?
Q.13. What is the parallel output?
Q.14. What are shift registers?
Q.15. What is the basic difference between a shift register and a counter?
Q.16. What are the applications of shift registers?
Q.17. What are buffer registers?
Q.18. How will you use a shift register to multiply or divide a binary number by 2?
Q.19. What are the basic types of shift registers?
Q.20. What is a serial-in, serial-out shift register?
Q.21. What is a serial-in, parallel-out shift register?
Q.22. What is a parallel-in, serial-out shift register?
Q.23. What is a parallel-in, parallel-out shift register?
Q.24. What is a bidirectional shift register?
Q.25. What is a universal shift register?

EXPERIMENT: 8
4-BIT COMPARATOR

AIM: To design simulate and synthesis the 4-Bit Comparator using Verilog HDL.

TOOL: Cadence Tool

THEORY:

A magnitude comparator is a combinational circuit that compares two numbers A & B to determine whether:

A > B, or A = B, or A < B

Inputs: First 4-bit number A

Second 4-bit number B

Outputs:3 output signals (GT, EQ, LT), where:

1. AgtB= 1 IFF A > B
2. AeqB = 1 IFF A = B
3. AltB = 1 IFF A < B

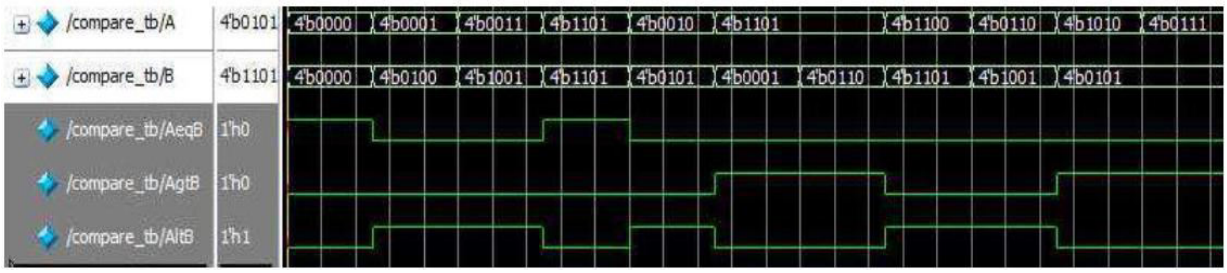
VERILOG CODE:

```
module compare (A, B, AeqB, AgtB, AltB);
input [3:0] A, B;
output AeqB, AgtB, AltB;
reg AeqB, AgtB, AltB;
always @(A or B)
begin
AeqB = 0;
AgtB = 0;
AltB = 0;
if(A == B)
AeqB = 1;
else if (A > B)
AgtB = 1;
else
AltB = 1;
end
endmodule
```

VERILOG TEST BENCH:

```
module compare_tb;
reg [3:0] A, B;
wire AeqB, AgtB, AltB;
compare comp(A, B, AeqB, AgtB, AltB);
always #5 B=$random%16;
always #5 A=$random%16;
initial
begin
$monitor($time, "\tA=%b\t , B=%b\t, AeqB=%b\t, AgtB=%b\t, AltB=%b\t", A,B, AeqB, AgtB, AltB);
A=4'b0000;
B=4'b0000;
#80 $finish;
end
endmodule
```

SIMULATION RESULTS:



RESULT:

Hence the 4-Bit Comparator has been synthesized and simulated using Cadence Simulator.

VIVA QUESTIONS:

1. What is a comparator in digital electronics?
2. What is a 4-bit comparator?
3. What are the inputs and outputs of a 4-bit comparator?
4. What are the three output conditions of a comparator?
5. What does A > B signify in a comparator?
6. What does A < B signify in a comparator?
7. What does A = B signify in a comparator?
8. How many bits are compared in a 4-bit comparator?
9. What is the significance of MSB in comparison?
10. Why is MSB compared first in a comparator?
11. How is equality between two bits checked?
12. Which logic gate is used to check equality?
13. Which logic gates are used to implement greater-than condition?
14. Which logic gates are used to implement less-than condition?
15. Write the Boolean expression for A = B condition.
16. Write the Boolean expression for A > B condition.
17. Write the Boolean expression for A < B condition.
18. What is the role of cascading in comparators?
19. Why do we cascade comparators?
20. What is a magnitude comparator?
21. What is the difference between 1-bit and 4-bit comparator?
22. How many 1-bit comparators are needed to build a 4-bit comparator?
23. What is the working principle of a 4-bit comparator?
24. How do you design a 4-bit comparator using basic gates?
25. What is the truth table of a 1-bit comparator?

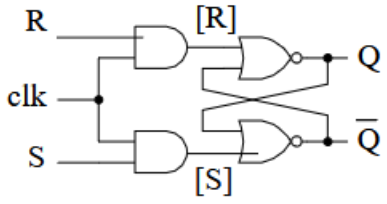
EXPERIMENT: 9
Flip-Flops: SR, D, JK AND T

AIM: To design simulate and synthesis the SR, D, JK and T Flip-Flops using Verilog HDL.

TOOL: Cadence Tool

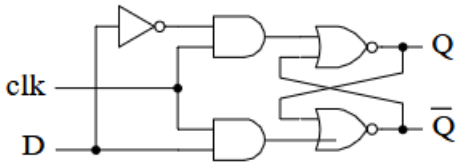
THEORY:

A] SR F/F: The major problem of RS latch is its susceptibility to voltage noise which could change the output states of the FF. With the clocked RS FF, the problem is remedied. With the clock held low, [S] & [R] held low, the output remains unchanged. With the clock held high, the output follows R & S. Thus the output will be latched its states when the clock goes low.



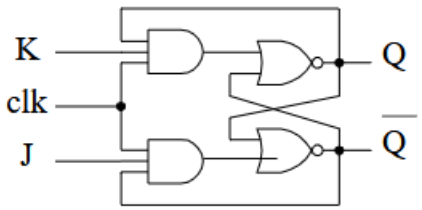
		clk= 1	clk = 0
S	R	Q_{n+1}	Q_{n+1}
0	0	Q_n	Q_n
0	1	0	Q_n
1	0	1	Q_n
1	1	Invalid i/p	Q_n

B] D F/F: The D-type FF remedies the indeterminate state problem that exists when both inputs to a clocked RS FF are high. The schematic is identical to a RS FF except that an inverter is used to produce a pair of complementary input.



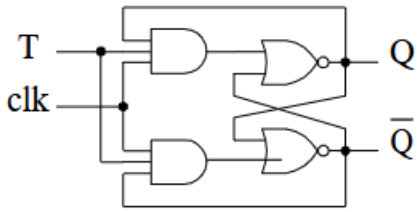
clk	D	Q_{n+1}
0	x	Q_n
1	0	0
1	1	1

C] JK F/F: The JK FF is a refinement of the RS FF in that the undetermined state of the RS type is defined in the JK type. Inputs J and K behave like inputs S and R to set and reset (clear) the FF, respectively. The input marked J is for set and the input marked K is reset



		clk=0	clk=1
J	K	Q_{n+1}	Q_{n+1}
0	0	Q_n	Q_n
0	1	Q_n	0
1	0	Q_n	1
1	1	Q_n	Q_n

D] T- F/F: The toggle (T) FF has a clock input which causes the output state changed for each clock pulse if T is in its active state. It is useful in counter design.



clk	T	Q_{n+1}
0	X	Q_n
1	0	Q_n
1	1	$\overline{Q_n}$

VERILOG CODE:

A] Verilog program for SR-flip flop

```
module srff(s,r,clk,rst,q,qb);
input s,r,clk,rst;
output q,qb;
wire s,r,clk,rst,qb;
reg q;
always @ (posedge clk)
begin
if(rst)
q<=1'b0;
else if (s==1'b0 && r==1'b0) q<=q;
else if (s==1'b0&& r==1'b1) q<=1'b0;
else if (s==1'b1 && r==1'b0) q<=1'b1;
else if (s==1'b1 && r==1'b1) q<=1'bx;
end
assign qb=~q;
endmodule
```

Verilog testbench program for SR-flip flop

```
module srff_tb;
reg s,r,clk,rst;
wire q,qb;
srff srflipflop(.s(s),.r(r),.clk(clk),.rst(rst),.q(q),.qb(qb));
initial
begin
clk=0;
s = 0; r = 0;
#5 rst = 1; #30 rst = 0;
$monitor($time, "\tclk=%b\t ,rst=%b\t, s=%b\t,r=%b\t, q=%b\t, qb=%b",clk,rst,s,r,q,qb);
#100 $finish;
end
always #5 clk = ~clk;
always #30 s = ~s;
always #40 r = ~r;
endmodule
```

B] Verilog program for D-flip flop with sync reset

```
module dff_sync_reset (
data , // Data Input
clk , // Clock Input
reset , // Reset input
q // Q output
);
input data, clk, reset ;
output q;
reg q;
always @ ( posedge clk)
if (reset) begin
q <= 1'b0;
end else begin
q <= data;
end
endmodule
```

Verilog testbench program for D-flip flop with sync reset

```
module dff_sync_reset_tb;
reg data, clk, reset ;
wire q;
dff_sync_reset dffr (.data(data), .clk(clk), .reset(reset) ,.q(q));
initial
begin
clk=0;
data = 0;
reset = 1;
#5 reset = 0;
#80 reset = 1;
$monitor($time, "\tclk=%b\t ,reset=%b\t, data=%b\t, q=%b",clk,reset,data,q);
#100 $finish;
end
always #5 clk = ~clk;
always #30 data = ~data;
endmodule
```

C] Verilog program for JK-flip flop

```
module JKFF ( input J,input K, input clk, input rst, output reg Q);
always @(posedge clk or posedge rst) //asynch reset
begin
if(rst == 1)
begin
Q <= 0;
end
else begin
```

```
case({J, K})
2'b00: Q <= Q; //no change
2'b01: Q <= 1'b0; //Clear
2'b10: Q <= 1'b1; //Set
2'b11: Q <= ~Q; //Complement
endcase
end
end
endmodule
```

Verilog testbench program for JK-flip flop

```
module JKFF_tb;
reg J,K,clk,rst;
wire Q;
JKFF JKflipflop(.J(J),.K(K),.clk(clk),.rst(rst),.Q(Q));
initial
begin
clk=0; J = 0; K = 0;
#5 rst = 1;
#30 rst = 0;
$monitor($time, "\tclk=%b\t ,rst=%b\t, J=%b\t,K=%b\t, Q=%b",clk,rst,J,K,Q);
#100 $finish;
end
always #5 clk = ~clk;
always #30 J = ~J;
always #40 K = ~K;
endmodule
```

D] Verilog program for T-flip flop with sync reset

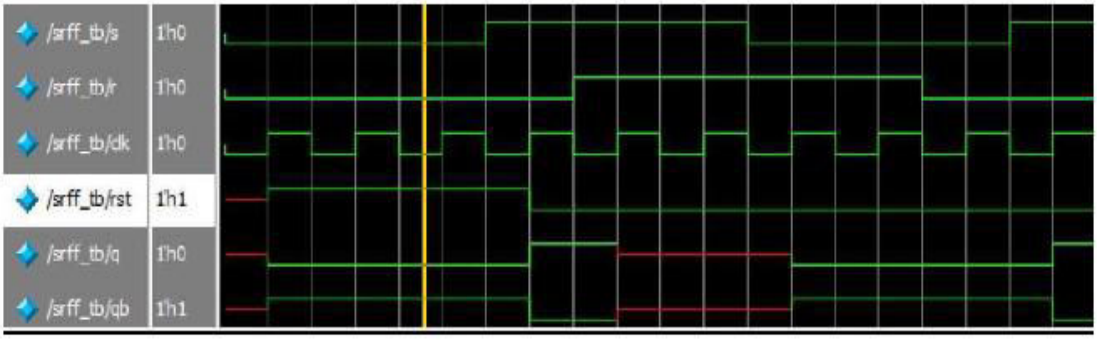
```
module tff_sync_reset (
data , // Data Input
clk , // Clock Input
reset , // Reset input
q // Q output
);
input data, clk, reset ;
output q;
reg q;
always @ ( posedge clk)
if (~reset) begin
q <= 1'b0;
end else if (data) begin
q <= !q;
end
endmodule
```

Verilog testbench program for T-flip flop with sync reset

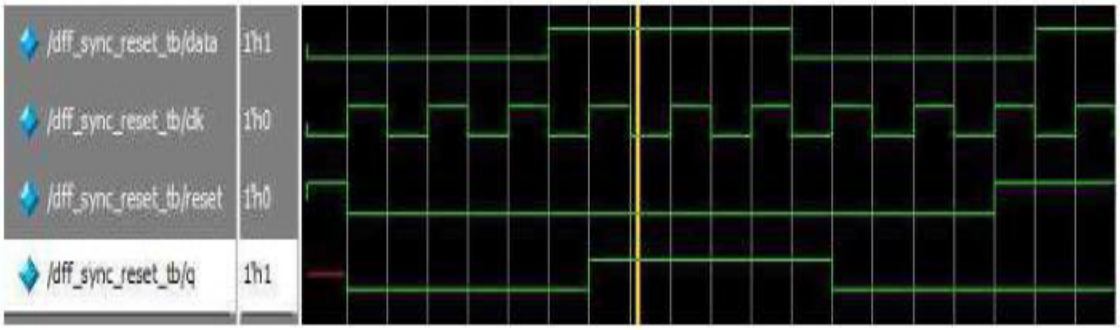
```
module tff_sync_reset_tb;
reg data, clk, reset ;
wire q;
tff_sync_reset tffr (.data(data), .clk(clk), .reset(reset) ,.q(q));
initial
begin
clk=0;
data = 0;
reset = 1;
#5 reset = 0;
#30 reset = 1;
$monitor($time, "\tclk=%b\t ,reset=%b\t, data=%b\t, q=%b",clk,reset,data,q);
#100 $finish;
end
always #5 clk = ~clk;
always #30 data = ~data;
endmodule
```

SIMULATION RESULTS:

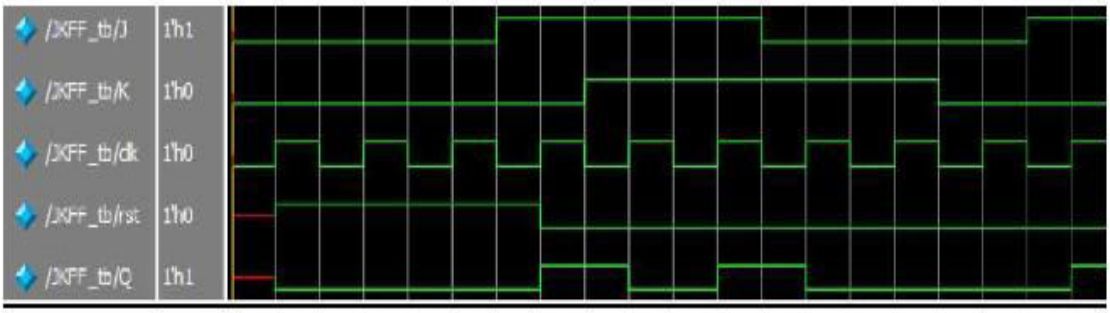
A] SR-flip flop



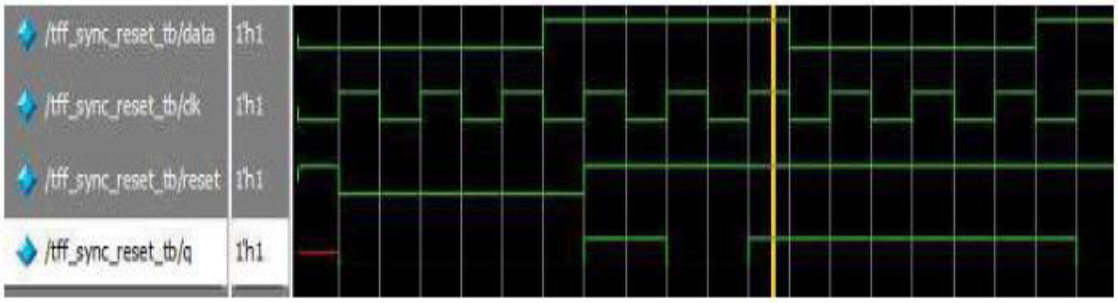
B] D-flip flop



C] JK-flip flop



D] T-flip flop



RESULT:

Hence the SR, D, JK and T flip-flops have been synthesized and simulated using Cadence Simulator.

VIVA QUESTIONS:

1. What is a flip-flop?
2. Why are flip-flops called bistable devices?
3. What are the basic types of flip-flops?
4. What is the difference between latch and flip-flop?
5. What is the function of a clock in flip-flops?
6. What is meant by triggering in flip-flops?
7. What are level-triggered flip-flops?
8. What are edge-triggered flip-flops?
9. What is propagation delay in flip-flops?
10. What is setup time?
11. What is hold time?
12. What is race condition in flip-flops?
13. What is metastability?
14. What is an SR flip-flop?
15. What are the inputs and outputs of an SR flip-flop?

16. What is the truth table of an SR flip-flop?
17. What is the invalid state in SR flip-flop?
18. Why is the SR flip-flop called a Set-Reset flip-flop?
19. What is the characteristic equation of SR flip-flop?
20. What is the excitation table of SR flip-flop?
21. What is a D flip-flop?
22. Why is D flip-flop called a data or delay flip-flop?
23. What is the truth table of a D flip-flop?
24. What is the characteristic equation of D flip-flop?
25. How does D flip-flop eliminate the invalid state of SR flip-flop?

EXPERIMENT: 10
RIPPLE COUNTERS

AIM: To design simulate and synthesis the Mod-10 and Mod-12 Ripple Counters using Verilog HDL.

TOOL: Cadence Tool

THEORY:

In a ripple counter, the flip-flop output transition serves as a source for triggering other flip-flops. In other words, the Clock Pulse inputs of all flip-flops (except the first) are triggered not by the incoming pulses, but rather by the transition that occurs in other flip-flops. A binary ripple counter consists of a series connection of complementing flip-flops (JK or T type), with the output of each flip-flop connected to the Clock Pulse input of the next higher-order flip-flop. The flip-flop holding the LSB receives the incoming count pulses. All J and K inputs are equal to 1.The small circle in the Clock Pulse /Count Pulse indicates that the flip-flop complements duringa negative-going transition or when the output to which it is connected goes from 1 to 0. The flip-flops change one at a time in rapid succession, and the signal propagates through the counter in a ripple fashion. A binary counter with reverse count is called a binary down-counter. In binary down-counter, the binary count is decremented by 1 with every input count pulse.

VERILOG CODE:

A] MOD-10 COUNTER

```
module MOD10(A0,A1,A2,A3,COUNT);
outputA0,A1,A2,A3;
input COUNT;
wire RESET;
//Instantiate Flip-Flop
FF F0(A0,COUNT,RESET);
FFF1(A1,A0,RESET);
FFF2(A2,A1,RESET);
FFF3(A3,A2,RESET);
//InstantiatePrimitivegate nand(RESET,A1,A3);
endmodule
```

```
//Description of Flip-Flop moduleFF(Q,CLK,RESET);

outputQ; inputCLK,RESET;

regQ=1'b0;

always @(negedge CLK or negedge RESET)

if(~RESET)
Q=1'b0;
else
Q=(~Q);
endmodule
```

VERILOG TEST BENCH:
A] MOD-10 COUNTER

```
module simulation;
reg COUNT;
wireA0,A1,A2,A3;
//InstantiateMOD10Counter
MOD10MOD10_TEST(A0,A1,A2,A3,COUNT);
Always#10COUNT=~COUNT;
initial
begin
COUNT=1'b0;
end
endmodule
```

VERILOG CODE:
B] MOD-12 COUNTER

```
Module
MOD12(A0,A1,A2,A3,COUNT);
output A0,A1,A2,A3;
input COUNT;
wire RESET;
//Instantiate Flip-Flop
FF F0(A0,COUNT,RESET);
FF F1(A1,A0,RESET);
FF F2(A2,A1,RESET);
FF F3(A3,A2,RESET);
//Instantiate Primitive gates
nand (RESET,A2,A3);
endmodule
```

```
//Description of Flip-Flop
module FF(Q,CLK,RESET);
output Q;
input CLK,RESET;
reg Q=1'b0;
always @(negedge CLK or negedge RESET) if(~RESET)
Q=1'b0;
else
Q=(~Q);
endmodule
```

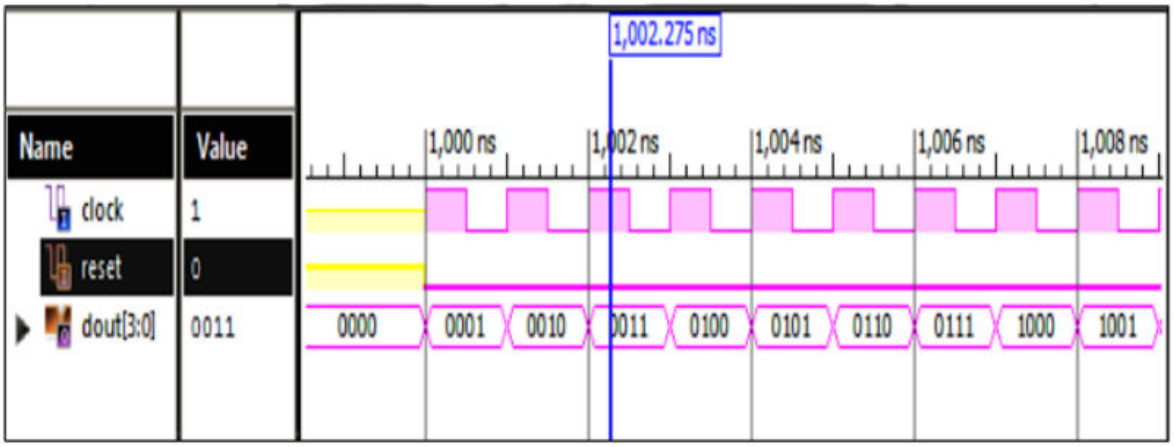
VERILOG TEST BENCH:

B] MOD-12 COUNTER

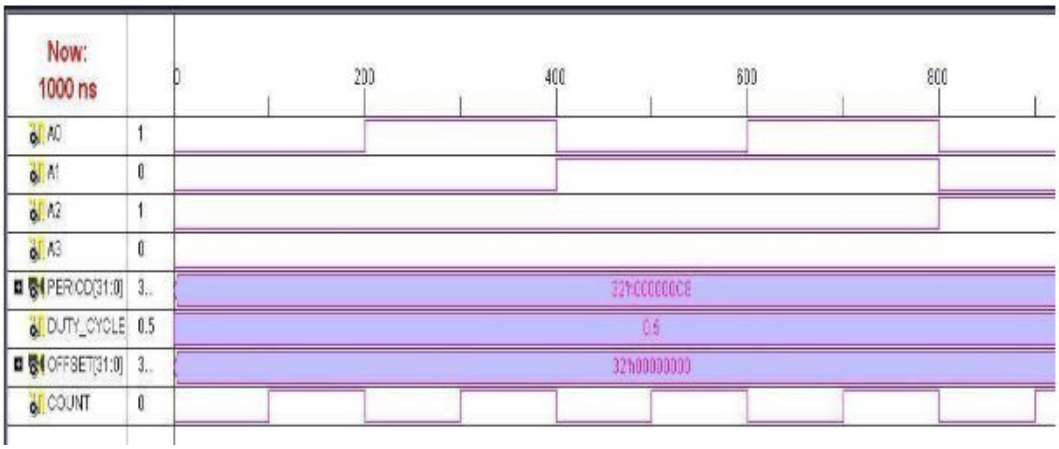
```
module simulation;
reg COUNT;
wire A0,A1,A2,A3;
//Instantiate MOD12 Counter
MOD12 MOD12_TEST(A0,A1,A2,A3,COUNT);
always
#10 COUNT=~COUNT;
initial
begin
COUNT=1'b0;
end
endmodule
```

SIMULATION RESULTS:

A] MOD-10 COUNTER



B] MOD-12 COUNTER



RESULT:

Hence the Mod-10 Counter and Mod-12 have been synthesized and simulated using Cadence Simulator.

VIVA QUESTIONS:

1. What is a Counter?
2. What is a Ripple counter?
3. Advantages of Ripple Counter in Digital Logic
4. Disadvantages of Ripple Counter in Digital Logic
5. How do you overcome the limitations of propagation delay in ripple counters?
6. What are some common alternatives to ripple counters in digital logic design?
7. A sequential circuit design is used to
8. In general, when using a scope to troubleshoot digital systems, the instrument should be triggered by
9. Which counters are often used whenever pulses are to be counted and the results displayed in decimal?
10. The counter in the Altera library has controls that allow it to count up or down, and perform synchronous parallel load and asynchronous cascading.
11. The minimum number of flip-flops that can be used to construct a modulus-5 counter is
12. The duty cycle of the most significant bit from a 4-bit (0–9) BCD counter is
13. MOD-16 counter requires no. of states.
14. Normally, the synchronous counter is designed using.
15. What is a state diagram?
16. High speed counter is
17. Program counter in a digital computer
18. Fundamental mode is another name for ?
19. How many natural states will there be in a 4-bit ripple counter?
20. A ripple counter's speed is limited by the propagation delay of ?
21. One of the major drawbacks to the use of asynchronous counters is that?
22. Internal propagation delay of asynchronous counter is remove

EXPERIMENT: 11
FINITE STATE MACHINE

AIM: To design simulate and synthesis the Finite State Machine usingVerilog HDL
TOOL: Cadence Tool

THEORY:

Finite state machine (FSM) is a term used by programmers, mathematicians, engineers and other professionals to describe a mathematical model for any system that has a limited number of conditional states of being. A practical example of a finite state machine is a set of buttons on a video game controller that are connected to a specific set of actions within the game. When a user inputs hitting certain buttons, the system knows to implement the actions that correspond.

The makeup of a finite state machine consists of the following:

- A set of potential input events.
- A set of probable output events that correspond to the potential input events.
- A set of expected states the system can exhibit.

A finite state machine may be implemented through software or hardware to simplify a complex problem. Within an FSM, all states in consideration exist in a finite list and the abstract machine can only take on one of those states at a time. This approach allows each input and output scenario to be studied and tested.

An FSM may be something very abstract, like a model for a business represented by an illustration, or it may be something concrete, like a vending machine or computer. The list of possible combinations of these elements is limited within a finite state machine. Alternatively, a state machine can be fuzzy. A fuzzy state machine allows the possibility of points of data that are not within discrete, pre-designated categories.

The types of computational models within automata theory include:

- Finite state machines—Models for any system with a limited number of conditional states of being.
- Pushdown automata – More complicated than finite state machines, these use regions of memory called stacks to store information as part of a model.
- Linear-bounded automata (LBA) – Similar to a Turing machine, but the data is limited to a portion of input within a finite group of inputs.
- Turing machines—The most complex mathematical model within automata theory for testing different input combinations to analyze a larger system or problem.

VERILOG CODE:

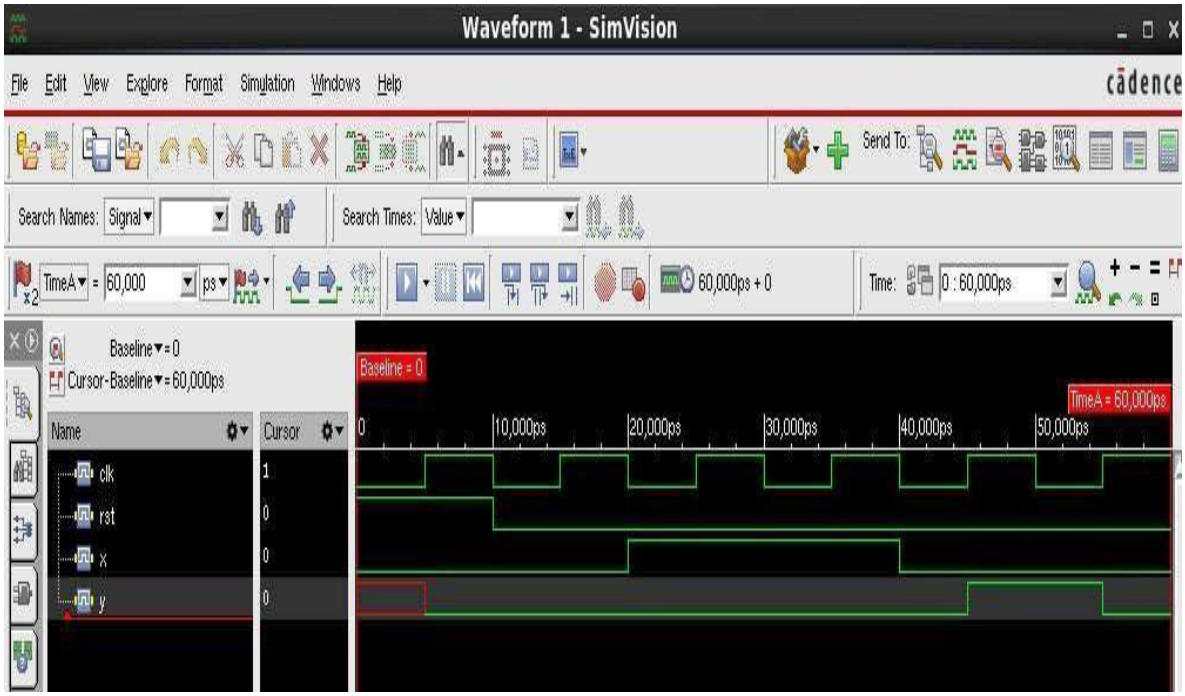
```
`timescale1ns/1ps

module seq_det(x,clk,rst,y);
inputx,clk,rst;
outputy;

reg [2:0] state;
regtemp;
parameter S0 = 3'b000, S1 = 3'b001, S2 = 3'b010, S3 = 3'b011, S4
=3'b100;

always @(posedge clk) begin
if (rst)
state<=S0;
else
case (state)
S0: if(x)
state<=S3;
else
state<=S1;
S1:if(x)
state<=S2;
else
state<=S1;
S2:if(x)
state<=S3;
else
state<=S4;
S3:if(x)
state<=S3;
else
state<=S4;
S4:if(x)
state<=S2;
else
state<=S1;
endcase
end
always @(state)
begin
case (state)
S0:temp<=1'b0;
S1:temp<=1'b0;
S2:temp<=1'b0;
```

```
        S3:temp<=1'b0;
        S4:temp<=1'b1;
        endcase
        end
assign y = temp;
endmodule
```

VERILOG TEST BENCH:

```
        `timescale 1ns/1ps
        moduleseq_det_tb();  reg
        x, clk, rst;wirey; seq_det
        uut(x,clk,rst,y);
        initialbegin
        clk=0;
        forever #5 clk=~clk; end
        initial begin
        rst=1;
        x=0;
        #10; rst=0;
              x=0;
        #10; rst=0;
               x=1;
        #10; rst=0;
               x=1;
        #10; rst=0;
              x=0;
        #10; rst=0;
        x=0;
#10 $finish;
end
endmodule
```

SIMULATION RESULTS:



SYNTHESIS TCL SCRIPT:

```
set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib set_attrlibraryslow.lib
set_attr hdl_search_path ./ read_hdlfsm.v
elaborate
synthesize -to_mapped -effort medium write_hdl>fsm_netlist.v
gui_show
report timing > fsm_timing.repo report
power > fsm_power.repo
reportarea>fsm_area.repo
```

RESULT:

Hence the Finite state machine has been synthesized and simulated using Cadence Simulator.

VIVA QUESTIONS:

1. What are finite state machines?
2. Can you give me an example of a real-world scenario where using a finite state machine would have been useful?
3. How do you define a finite state machine in Python?
4. How can you use the FSM class to implement a simple calculator?

5. What are some examples of applications that could be built using FSMs?
6. Is it possible to represent a Finite State Machine on paper? If yes, then how?
7. What is the difference between a transition and a trigger?
8. What are guard conditions used for in practice?
9. What's the best way to write a unit test for an application that uses finite state machines?
10. What are the differences between Moore and Mealy Machines in practice?
11. What does a DFA stand for? What are its characteristics?
12. Can you explain what an NFA is?
13. What is the advantage of using non-determinism when defining a state machine?
14. What are the advantages of automating your tests with finite state machines?
15. Why are finite state machines considered more efficient than regular expressions in certain situations?
16. Is there a way to convert a finite state machine into a Turing Machine? If yes, then how?
17. What are some of the limitations of finite state machines?
18. Are finite state machines popular in industry today? Which companies are currently making extensive use of them?
19. What is a state diagram?
20. What is a PDA or pushdown automaton?
21. Design a finite state machine FSM for a serial two's complement block
22. and also draw the logic diagram associated with it by using D-flipflop.
23. Explain briefly with the difference between the Mealy model and Moore model.
24. Design an FSM for serial sequence detector with the pattern "1010" with overlapping and with non-overlapping.
25. Design an FSM for serial sequence detector with the pattern "0110" with non-overlapping. use Mealy Machine.

EXPERIMENT: 12
MEMORY ELEMENTS

AIM: To design simulate and synthesis memory elements using Verilog HDL.

TOOL: Cadence Tool

THEORY:

Designing memory using Hardware Description Language (HDL), such as Verilog or VHDL, involves specifying the structure and behavior of memory blocks. This can include different types of memory like RAM (Random Access Memory), ROM (Read-Only Memory), Registers, or even FIFO (First In, First Out) buffers, depending on the application. Below, I'll provide examples for designing basic RAM and ROM modules in both Verilog and VHDL.

Key Considerations:

Synchronous vs Asynchronous: The above designs are **synchronous**, meaning that memory operations (read and write) happen on a clock signal (positive edge).
Initialization: In ROM, data is fixed, whereas in RAM, data can be written and read dy- namically.
Size and Addressing: The memory size and addressing can be modified based on the de- sired configuration (e.g., more bits for data or address, different memory sizes).

These examples can be expanded further for more complex designs such as **dual-port RAM**, **FIFO memory**, **block RAM**, etc. depending on your system's requirements.

VERILOG CODE:

```
module ram (
    input clk,              // Clock signal
    input rst,              // Reset signal
    input we,               // Write enable
    input [3:0] addr,       // 4-bit address (16 locations)
```

```
    input [7:0] data_in,        // 8-bit data input
    output reg [7:0] data_out // 8-bit data output
);
    // Declare memory with 16 locations of 8 bits each reg
    [7:0] memory [0:15];
    // Always block to handle read and write operations
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            // Reset memory to zero on reset integer
            i;
            for (i = 0; i < 16; i = i + 1) begin
                memory[i] <= 8'b0;
            end
            data_out <= 8'b0; // Set output to zero on reset end
        else if (we) begin
            memory[addr] <= data_in; // Write data to memory end
        data_out <= memory[addr]; // Read data from memory end
endmodule
```

VERILOG TEST BENCH:

```
module tb_ram;
// Testbench signals reg
clk;
reg rst;
reg we;
reg [3:0] addr;
reg [7:0] data_in;
wire [7:0] data_out;
```

```
// Instantiate the RAM module
ram uut (
    .clk(clk),
    .rst(rst),
    .we(we),
    .addr(addr),
    .data_in(data_in),
    .data_out(data_out)
);

// Clock generation always
begin
    #5 clk = ~clk; // Toggle the clock every 5 time units end

// Test sequence
initial begin
    // Initialize signals
    clk = 0;
    rst = 0;
    we = 0; addr
    = 4'b0;
    data_in = 8'b0;

    // Apply Reset
    $display("Applying reset..."); rst =
    1;
    #10 rst = 0;  // Release reset after 10 time units

    // Write data to RAM
    $display("Writing data to RAM..."); we =
    1;
    addr = 4'b0001;  // Write to address 1 data_in =
    8'b10101010;  // Data to write #10;  // Wait for
```

```
    the operation to complete

    // Check if the data is correctly written
    $display("Reading from RAM at address 1..."); we =
    0;  // Disable write
    addr = 4'b0001;  // Read from address 1 #10;
    $display("Data at address 1: %b", data_out);  // Display the read data

    // Write more data to another address
    $display("Writing data to address 2..."); we =
    1;
    addr = 4'b0010;
    data_in = 8'b11001100;  // New data #10;

    // Read from address 2
    $display("Reading from RAM at address 2..."); we =
    0;  // Disable write
    addr = 4'b0010;
    #10;
    $display("Data at address 2: %b", data_out);  // Display the read data

    // Perform additional tests...

    // End the simulation
    $finish;
end
endmodule
```

SIMULATION RESULTS:

SYNTHESIS TCL SCRIPT:

set_attr lib_search_path /cad/FOUNDRY/digital/90nm/lib set_attrlibraryslow.lib

set_attr hdl_search_path ./ read_hdlfsm.v

elaborate

synthesize -to_mapped -effort medium write_hdl>fsm_netlist.v

gui_show

report timing > fsm_timing.repo report

power > fsm_power.repo

reportarea>fsm_area.repo

RESULT:

Hence the Memories has been synthesized and simulated using Cadence Simulator.

VIVA QUESTIONS:

1. What is memory in the context of digital systems?
2. What are the different types of memory used in digital systems?
3. Can you explain the difference between RAM and ROM?
4. What is the basic function of the Arithmetic Logic Unit (ALU) in relation to memory?
5. What is the difference between volatile and non-volatile memory?
6. What is static RAM (SRAM) and how does it differ from dynamic RAM (DRAM)?
7. How does a flip-flop relate to memory design?
8. What is the function of a register in a CPU? How is it different from memory?
9. What are the advantages of using SRAM over DRAM?
10. What is read/write memory? Explain with an example.
11. How would you design a single-port RAM in Verilog/VHDL?
12. What is the purpose of the write-enable (WE) signal in memory?
13. Explain how a dual-port RAM differs from a single-port RAM.
14. What is asynchronous memory and how does it differ from synchronous memory?
15. What is the advantage of using a synchronous RAM over an asynchronous RAM?
16. Can you explain how address decoding works in memory modules?
17. What is the purpose of a clock signal in synchronous memory operations?
18. Explain the difference between RAM and ROM in terms of read/write capabili- ties.
19. What are the applications of FIFO memory in digital systems?
20. What is memory-mapped I/O and how is it used in processor design?
21. What is the difference between bit-addressable and byte-addressable memory?
22. Explain how memory banks work and their role in larger memory designs.
23. How would you implement a memory hierarchy (cache, main memory, etc.) in a system?
24. How is cache memory related to main memory in terms of access speed?
25. What is a block RAM (BRAM) and how is it different from distributed RAM?
26. How does word size affect memory architecture?

EXPERIMENT: 13
4-BIT PARITY GENERATOR AND CHECKER

AIM: To design simulate and synthesis the 4-Bit Parity Generator and Checker using Verilog HDL.

TOOL: Cadence Tool

THEORY:

A 4-bit parity generator and checker is a digital circuit used for simple error detection in data transmission. The parity generator adds an extra bit (parity bit) to the 4-bit data to make the total number of 1's either even (even parity) or odd (odd parity), typically using XOR gates. The parity checker verifies the received data along with the parity bit to detect any error during transmission. If the parity condition is not satisfied, an error is indicated. This method is simple and widely used but can only detect single-bit errors.

VERILOG CODE:

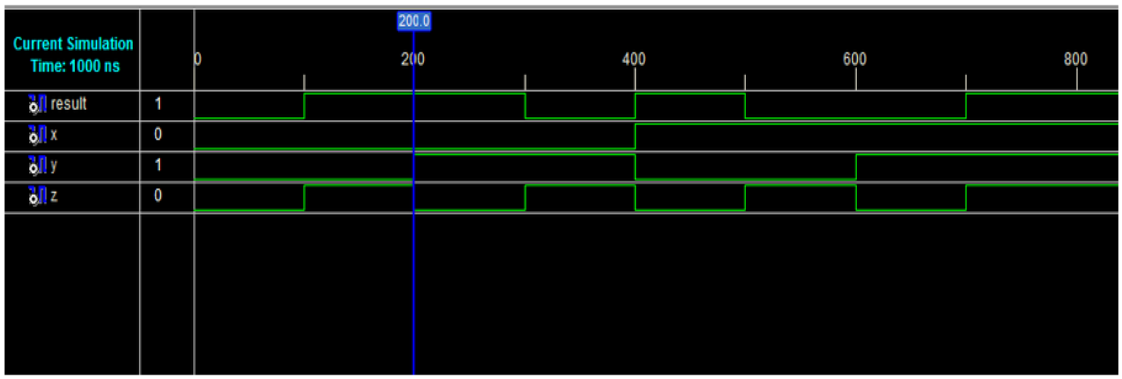
```
module parity_4bit (
input x, y, z, w, // 4-bit inputs
input p_in, // parity input (for checker)
output result_gen, // generated parity
output result_chk // error output (0 = no error, 1 = error)
);
// Parity Generator (Even Parity)
assign result_gen = x ^ y ^ z ^ w;
// Parity Checker
assign result_chk = x ^ y ^ z ^ w ^ p_in;
endmodule
```

VERILOG TEST BENCH:

```
module parity_tb;
reg x, y, z, w;
reg p_in;
wire result_gen, result_chk;
parity_4bit uut (
.x(x),
.y(y),
.z(z),
.w(w),
.p_in(p_in),
.result_gen(result_gen),
.result_chk(result_chk)
);
```

```
initial begin
$display("x y z w | Parity | Error");

// Test case 1
x=0; y=0; z=0; w=0; p_in=0; #10;
$display("%b %b %b %b | %b | %b", x,y,z,w,result_gen,result_chk);

// Test case 2
x=1; y=0; z=1; w=0; p_in=0; #10;
$display("%b %b %b %b | %b | %b", x,y,z,w,result_gen,result_chk);

// Test case 3
x=1; y=1; z=1; w=1; p_in=0; #10;
$display("%b %b %b %b | %b | %b", x,y,z,w,result_gen,result_chk);

// Test case 4 (with correct parity)
x=1; y=0; z=1; w=1; p_in=1; #10;
$display("%b %b %b %b | %b | %b", x,y,z,w,result_gen,result_chk);

// Test case 5 (error case)
x=1; y=0; z=1; w=1; p_in=0; #10;
$display("%b %b %b %b | %b | %b", x,y,z,w,result_gen,result_chk);

$finish;
end
endmodule
```

SIMULATION RESULTS:



RESULT:

Hence the 4-Bit Parity Generator and Checker has been synthesized and simulated using Cadence Simulator.

VIVA QUESTIONS:

1. What is parity in digital systems?
2. What is a parity generator?
3. What is a parity checker?
4. Why is parity used in digital communication?
5. What are the types of parity?
6. What is even parity?
7. What is odd parity?
8. What is a 4-bit parity generator?
9. What is the function of a parity checker?
10. What is the difference between generator and checker?
11. Which logic gate is mainly used in parity circuits?
12. Why is XOR gate used in parity generation?
13. What is the truth table of XOR gate?
14. How is even parity generated for 4 bits?
15. How is odd parity generated for 4 bits?
16. Write the Boolean expression for even parity generator.
17. Write the Boolean expression for odd parity generator.
18. How many XOR gates are required for a 4-bit parity generator?
19. What is the output of parity generator for input 1010 (even parity)?
20. What is the output of parity generator for input 1111 (odd parity)?
21. What is a parity bit?
22. Where is the parity bit placed in data?
23. How does a parity checker detect errors?
24. Can parity detect all types of errors?
25. What type of errors can parity detect?

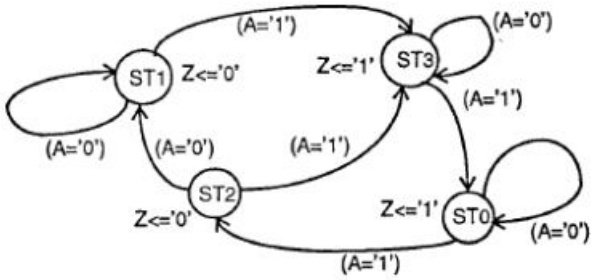
EXPERIMENT: 14
SEQUENCE DETECTOR USING FINITE STATE MACHINE

AIM: To design simulate and synthesis the Sequence Detector Using Finite State Machine using Verilog HDL.

TOOL: Cadence Tool

THEORY:
A sequence detector using Finite State Machine (FSM) is a sequential circuit designed to identify a specific pattern of bits in a serial input stream. It operates by transitioning through a set of defined states based on the input sequence and produces an output when the desired pattern is detected. FSMs can be implemented using either Moore or Mealy models, where the output depends on the current state alone or on both the state and input, respectively. Sequence detectors are widely used in digital systems for pattern recognition, communication, and control applications.

MOORE FSM:



VERILOG CODE:

```
module moorefsm(a,clk,z);
input a;
input clk;
output z;
reg z;
parameter st0=0,st1=1,st2=2,st3=3;
reg[0:1]moore_state;
initial
begin
moore_state=st0;
end
always @ (posedge(clk))
case(moore_state)
st0:
begin
z=1;
if(a)
moore_state=st2;
end
st1:
```

```
begin
z=0;
if(a)
moore_state=st3;
end
st2:
begin
z=0;
if(~a)
moore_state=st1;
else
moore_state=st3;
end
st3:
begin
z=1;
if(a)
moore_state=st0;
end
endcase
endmodule
```

MEALY FSM:

VERILOG CODE:

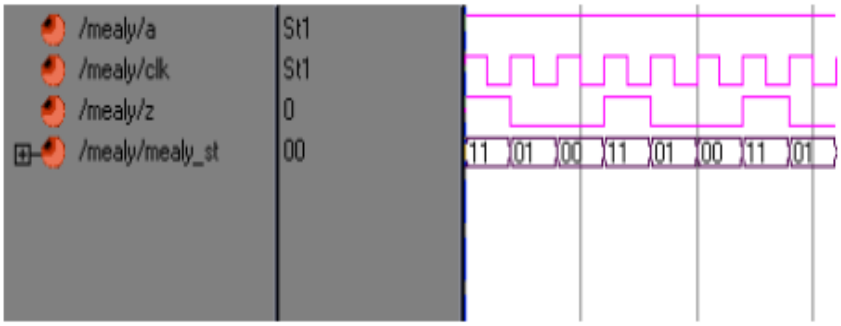
```
module mealayfsm(a, clk, z);
input a;
input clk;
output z;
reg z;
parameter st0=0,st1=1,st2=2,st3=3;
reg[0:1]mealy_state;
initial
begin
mealy_state=st0;
end
always @ (posedge(clk))
case(mealy_state)
st0:
begin
if(a) begin
z=1;
mealy_state=st3; end
else
z=0;
```

```
end
st1:
begin
if(a) begin
z=0;
mealy_state=st0; end
else
z=1;
end
st2:
begin
if(a) begin
z=1;
mealy_state=st1; end
else
z=0;
end
st3:
begin
z=0;
if(a) begin
mealy_state=st1; end
else
mealy_state=st2;
end
endcase
endmodule
```

SIMULATION RESULTS:

MOORE:

MEALY:



RESULT:

Hence the sequence detector has been synthesized and simulated using Cadence Simulator.

VIVA QUESTIONS:

1. What is a sequence detector?
2. What is the purpose of a sequence detector?
3. What is a Finite State Machine (FSM)?
4. Why is FSM used in sequence detection?
5. What are the components of an FSM?
6. What are states in FSM?
7. What is a state transition?
8. What is the role of input in FSM?
9. What is the role of output in FSM?
10. What are the types of FSM?
11. What is a Moore machine?
12. What is a Mealy machine?
13. What is the main difference between Moore and Mealy machines?
14. Which FSM is faster, Moore or Mealy? Why?
15. Which FSM requires fewer states?
16. What is a state diagram?
17. What is a state table?
18. What is state assignment?
19. Why is state assignment important?
20. What is sequence 1011 detector?
21. How many states are required for detecting a sequence?
22. What is overlapping in sequence detection?
23. What is non-overlapping sequence detection?
24. Which is more efficient: overlapping or non-overlapping?
25. What happens after sequence detection in non-overlapping case?

EXPERIMENT: 15
4X1 MULTIPLEXER

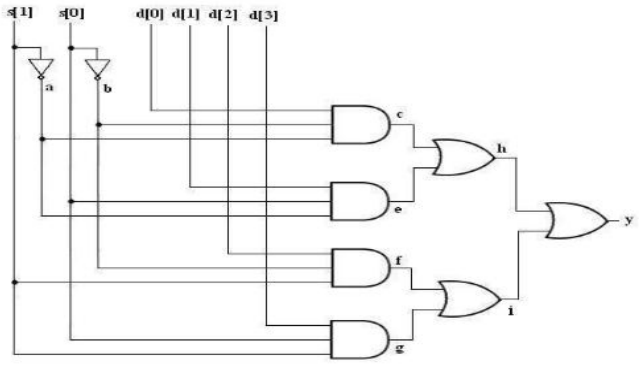
AIM: To design simulate and implement 4X1 Multiplexer using Verilog HDL.

TOOL: Cadence Tool

THEORY:

A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. Multiplexing means transmitting a large number of information units over a smaller number of channels or lines. The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2n input lines and n selection lines whose bit combinations determine which input is selected. A multiplexer is also called a data selector, since it selects one of many inputs and steers the binary information to the output lines. Multiplexer ICs may have an enable input to control the operation of the unit. When the enable input is in a given binary state (the disable state), the outputs are disabled, and when it is in the other state (the enable state), the circuit functions as normal multiplexer. The enable input (sometimes called strobe) can be used to expand two or more multiplexer ICs to digital multiplexers with a larger number of inputs.The size of the multiplexer is specified by the number 2n of its input lines and the single output line. In general, a 2n – to – 1 line multiplexer is constructed from an n – to 2 n decoder by adding to it 2n input lines, one to each AND gate. The outputs of the AND gates are applied to a single OR gate to provide the 1 – line output

INPUT		OUTPUT
s[1]	s[0]	y
0	0	D[0]
0	1	D[1]
1	0	D[2]
1	1	D[3]



VERILOG CODE:

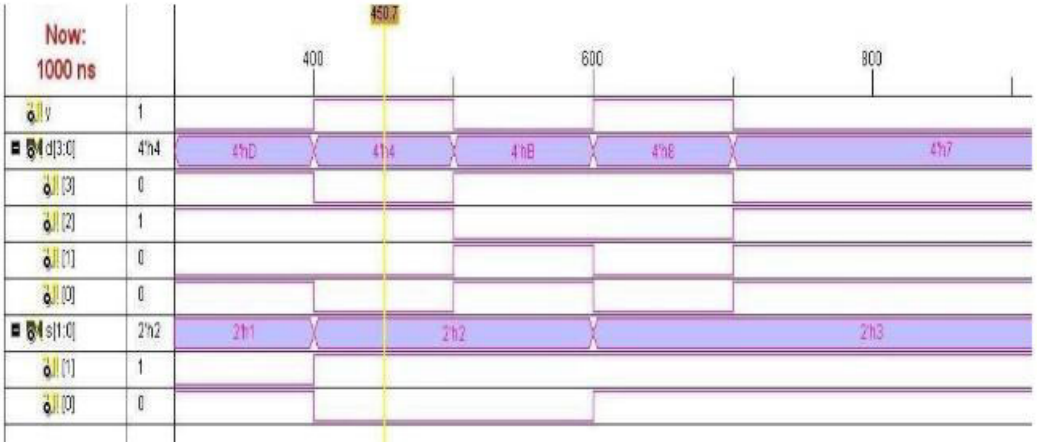
```
module
multiplexer(y,d,s);
output y;
input [3:0] d;
input [1:0] s;
wire a,b,c,e,f,g,h,i;
```

```
//Instantiate Primitive gates
not (a,s[1]);
not (b,s[0]);
and (c,d[0],b,a);
and (e,d[1],s[0],a);
and (f,d[2],b,s[1]);
and (g,d[3],s[0],s[1]);
or (h,c,e);
or (i,f,g);
or (y,h,i);
endmodule
```

VERILOG TEST BENCH:

```
module simulation;
reg [3:0]d;
reg [1:0]s;
wire y;
//Instantiate the 4 to 1 Multiplexer
multiplexer mux_t(y,d,s);
initial begin
s=2'b00;d[0]=1'b1;d[1]= 1'b0;d[2]= 1'b0;d[3]= 1'b0; #100
s=2'b00;d[0]= 1'b0;d[1]= 1'b1;d[2]= 1'b1;d[3]= 1'b1; #100
s=2'b01;d[0]= 1'b0;d[1]= 1'b1;d[2]= 1'b0;d[3]= 1'b0; #100
s=2'b01;d[0]= 1'b1;d[1]= 1'b0;d[2]= 1'b1;d[3]= 1'b1; #100
s=2'b10;d[0]= 1'b0;d[1]= 1'b0;d[2]= 1'b1;d[3]= 1'b0; #100
s=2'b10;d[0]= 1'b1;d[1]= 1'b1;d[2]= 1'b0;d[3]= 1'b1; #100
s=2'b11;d[0]= 1'b0;d[1]= 1'b0;d[2]= 1'b0;d[3]= 1'b1; #100
s=2'b11;d[0]= 1'b1;d[1]= 1'b1;d[2]= 1'b1;d[3]= 1'b0;
end
endmodule
```

SIMULATION RESULTS:



RESULT:

Hence the multiplexer has been desined and simulated using Cadence Simulator.

VIVA QUESTIONS:

1. What is a multiplexer?
2. Why is a multiplexer called a data selector?
3. How many select lines are required for a 4-to-1 multiplexer?
4. Write the general formula relating number of inputs and select lines in a MUX.
5. What is the difference between a multiplexer and a demultiplexer?
6. Draw the block diagram of a 2-to-1 multiplexer and explain its operation.
7. Write the Boolean expression for a 2-to-1 MUX.
8. How does a 4-to-1 multiplexer work?
9. What is the role of select lines in a multiplexer?
10. Can a multiplexer be used to implement Boolean functions? How?
11. How many input lines are there in an 8-to-1 multiplexer?
12. What is the function of an enable (control) input in a MUX?
13. Explain how a multiplexer can be used as a universal logic circuit.
14. What is the difference between combinational circuits and multiplexers?
15. How can you implement a 16-to-1 MUX using smaller multiplexers?
16. What are the advantages of using multiplexers in digital circuits?
17. What are the limitations of multiplexers?
18. How is a multiplexer used in communication systems?
19. Compare multiplexer with encoder.
20. What happens if select lines are not properly defined in a MUX?

EXPERIMENT: 16
DE'MORGANS THEOREM

AIM: To Verify De'morgans Theorem for 2 variables.using Verilog HDL.

TOOL: Cadence Tool

THEORY:

De Morgan's Theorem is a fundamental rule in **Boolean algebra** that explains how the complement of a Boolean expression can be simplified by changing operators and complementing each variable.
It consists of **two laws**:

1. **First Law (AND → OR):**

The complement of a product is equal to the sum of the complements.

$$(A \cdot B)' = A' + B'$$

2. **Second Law (OR → AND):**

The complement of a sum is equal to the product of the complements.

$$(A + B)' = A' \cdot B'$$

VERILOG CODE:

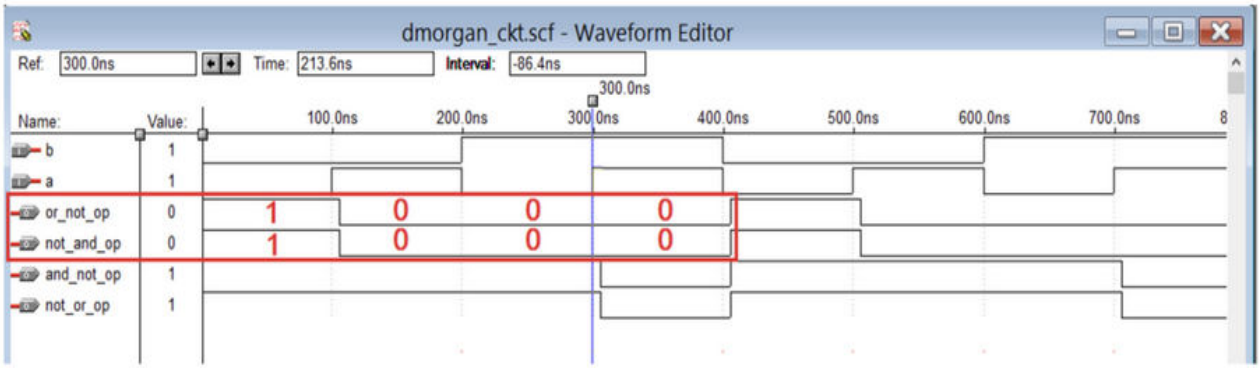
```
module d_morgan(a,b, or_not_op,not_and_op,and_not_op, not_or_op);
  input a,b;
  output or_not_op,not_and_op,and_not_op, not_or_op;
  wire a_bar,b_bar;
    not(a_bar,a);
    not(b_bar,b);
    nand(and_not_op,a,b);
    or(not_or_op,a_bar,b_bar);
    nor(or_not_op,a,b);
    and(not_and_op,a_bar,b_bar);
endmodule
```

VERILOG TEST BENCH:

```
`timescale 1ns/1ps

module d_morgan_tb;

// Inputs
reg a, b;

// Outputs
wire or_not_op, not_and_op, and_not_op, not_or_op;

// Instantiate the DUT (Device Under Test)
d_morgan uut (
.a(a),
.b(b),
```

```
.or_not_op(or_not_op),
.not_and_op(not_and_op),
.and_not_op(and_not_op),
.not_or_op(not_or_op)
);

// Test procedure
initial begin
$display("a b | or_not_op not_and_op and_not_op not_or_op");
$monitor("%b %b | %b %b %b %b",
a, b, or_not_op, not_and_op, and_not_op, not_or_op);

// Apply all input combinations
a = 0; b = 0; #10;
a = 0; b = 1; #10;
a = 1; b = 0; #10;
a = 1; b = 1; #10;

$finish;
end
endmodule
```

SIMULATION RESULTS:



RESULT:

Hence the De'morgans theorem has been desined and simulated using Cadence Simulator.

VIVA QUESTIONS:

1. What is De Morgan's Law?
2. State the two theorems of De Morgan's Law.
3. Write the Boolean expression for the complement of AND operation using De Morgan's Law.
4. Write the Boolean expression for the complement of OR operation using De Morgan's Law.
5. How is De Morgan's Law used in Boolean algebra simplification?

6. Explain De Morgan's Law with a truth table.
7. What is the complement of $A \cdot B$ using De Morgan's Law?
8. What is the complement of $A + B$ using De Morgan's Law?
9. How do you implement De Morgan's Law using basic logic gates?
10. How is De Morgan's Law useful in digital circuit design?
11. Can De Morgan's Law be applied to more than two variables? Give an example.
12. Convert a NAND gate into an OR gate using De Morgan's Law.
13. Convert a NOR gate into an AND gate using De Morgan's Law.
14. What is the relation between NAND, NOR gates and De Morgan's Law?
15. Why is De Morgan's Law important for CMOS logic design?
16. How does De Morgan's Law help in reducing hardware complexity?
17. Apply De Morgan's Law to simplify $(A+B+C)'$.
18. Apply De Morgan's Law to simplify $(A \cdot B \cdot C)'$.
19. What happens if De Morgan's Law is applied incorrectly?
20. How is De Morgan's Law verified using logic gates or simulation?