VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELGAVI-590018, KARNATAK

VIDYA VIKAS INSTITUTE OF ENGINEERING & TECHNOLOGY

#127-128, Mysuru - Bannur Road Post, Alanahalli, Mysuru, 570028

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

LAB MANUAL

Course: Digital Design and Computer Organization

Course Code: BCS302

III-SEMESTER

Prepared by:

Harshitha M

Assistant Professor,

Department of Information Science & Engineering, VVIET, Mysuru

ACADEMIC YEAR: 2024-25

VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI

B.E. in Information Science and Engineering Scheme of Teaching and Examinations 2022

Outcome-Based Education (OBE) and Choice Based Credit System (CBCS)

(Effective from the academic year 2024 - 25)

Integrated Professional Core Course (IPCC)

Refers to Professional Theory Core Course Integrated with Practical's of the same course

Course: Digital Design and Computer Organization

Course Code: BCS302

TABLE OF CONTENTS

VISION OF THE INSTITUTE

“Our vision is to provide learning opportunities, ensuring excellence in education, research and facilitate an inspiring world class environment to encourage creativity. The Institute is committed to disseminating knowledge, and through its ingenuity, bring this knowledge to bear on the world's great challenges. VVIET is dedicated to providing its students with an education that combines academic study and the excitement of discovery kindled by a diverse campus community.”

MISSION OF THE INSTITUTE

- Offer highest professional and academic standards in terms of personal growth and satisfaction, and promote growth and value to our research sponsors.
- Provide students a platform where independent learning and scientific study are encouraged with emphasis on latest engineering techniques.
- Encourage students to implement applications of engineering with a focus on societal needs for the betterment of communities.
- Empower students with vast technical and life skills to raise their stakes of getting placements in top reputed companies.
- Create a benchmark in the areas of research, education and public outreach.

VISION OF THE DEPARTMENT

“Our vision is to strive for excellence in intellectual inquiry and empower students with knowledge to overcome complexities in the operations and strategies of any engineering framework”

MISSION OF THE DEPARTMENT

M1: Imparting quality education by adopting the well designed
Curriculum under VTU, Karnataka in tune with the challenging software needs of the industry.

M2: Providing state of art research facilities to generate knowledge and develop technologies inthe thrust areas of Information science and engineering.

M3: Developing linkages with well recognized organizations to strengthen industry-academia relationships for mutual benefits.

M4: To provide state-of-the-art environment and opportunities to enhance professional skills.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO 1: To demonstrate analytical and technical problem solving abilities.

PEO 2: To conversant in the developments of information Technology, leading towards the employability and higher studies.

PEO 3: To engage in research and development leading to new innovations and products

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO 1: Apply Software Engineering Principles and Practices to provide reliable software solutions

PSO 2: Design and Develop Mobile and Web-based Computational systems for realistic societal challenges and constraints

PSO 3: Empower students with the capability of using modern tools to develop software system

PROGRAM OUTCOMES (POs)

PO1: Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: 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.

PO3: 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.

PO4: 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.

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.

PO6: 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.

PO7: 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.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and Team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: 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.

PO11: 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.

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.

GENERAL LAB GUIDELINES

Do's

1. Maintain discipline in the Laboratory.
2. Before entering the Laboratory, keep the footwear on the shoe rack.
3. Proper dress code has to be maintained while entering the Laboratory.
4. Students should carry a lab observation book, student manual and record book completed in all aspects.
5. Read and understand the logic of the program thoroughly before coming to the laboratory.
6. Enter the login book before switching on the computer.
7. Enter your batch member names and other details in the slips for hardware kits.
8. Students should be at their concerned places; unnecessary movement is restricted.
9. Students should maintain the same computer until the end of the semester.
10. Report any problems in computers/hardware kits to the faculty member in-charge/laboratory technician immediately.
11. The practical result should be noted down into their observation and the result must be shown to the faculty member in-charge for verification.
12. After completing the experiments, students should switch off the computers, enter logout time, return the hardware kits and keep the chairs properly.

Don'ts

1. Do not come late to the Laboratory.
2. Do not enter the laboratory without an ID card, lab dress code, observation book and record.
3. Do not leave the laboratory without the permission of the faculty in-charge.
4. Never eat, drink while working in the laboratory.
5. Do not handle any equipment before reading the instructions/instruction manuals.
6. Do not exchange the computers with others and hardware kits also.
7. Do not misbehave in the laboratory.
8. Do not alter computer settings/software settings.
9. External Disk/drives should not be connected to computers without permission, doing so will attract fines.
10. Do not remove anything from the kits/experimental set up without permission. Doing so will attract fines.
11. Do not mishandle the equipment / Computers.
12. Do not leave the laboratory without verification of hardware kits by the lab instructor.
13. Usage of Mobile phones, tablets and other portable devices are not allowed in restricted places.

INSTRUCTIONS TO STUDENTS

- Students must bring Observation book, record and manual along with pen, pencil, and eraser etc., no borrowing from others.
- Students must follow the installation instructions carefully to avoid any issues.
- Spend time familiarizing yourself with the user interface and features of the simulation software. Explore menus, toolbars, and panels to understand how to navigate through the software.
- Use the simulation software to design the digital or analog circuit as instructed in the experiment.
- If issues arise during the simulation, troubleshoot by checking connections, component values, and simulation settings. Utilize software debugging tools to identify and resolve errors. If needed, seek guidance by the instructor.
- After the completion of the experiment should document your entire simulation process, including circuit design, simulation setup, and results. Follow any specific documentation guidelines provided by your teacher.
- Before leaving the lab, should check whether they have switch off the power supplies and keep their chairs properly.
- Avoid unnecessary talking while doing the experiment
- Handle the system with care. Do not interchange the computer systems while doing the experiment
- Do not panic if you do not get the output.
- Review any feedback provided by the instructor on your simulation results. Use the feedback to improve your understanding and skills for future experiments.
- Keep your work area clean after completing the experiment.
- After completion of the experiment switch off the power and return the components
- Arrange your chairs and tables before leaving.

RULES FOR MAINTAINING LABORATORY RECORD

- Put your name, USN and subject on the outside front cover of the record. Put that same information on the first page inside.
- Update Table of Contents every time you start each new experiment or topic
- Always use pen and write neatly and clearly
- Start each new topic (experiment, notes, calculation, etc.) on a right-side (odd numbered) page
- Obvious care should be taken to make it readable, even if you have bad handwriting
- Date to be written every page on the top right side corner
- On each right-side page
  - ➢ Title of experiment
  - ➢ Aim/Objectives
  - ➢ Components Required
  - ➢ Theory
  - ➢ Procedure described clearly in steps
  - ➢ Result
- On each left side page
  - ➢ Pin diagrams
  - ➢ Circuit diagram
  - ➢ Tables
  - ➢ Graphs
- Use labels and captions for figures and tables
- Attach printouts and plots of data as needed. Stick printouts (A4 Size) on the right side of the lab record
- Strictly observe the instructions given by the Teacher/ Lab Instructor.

SYLLABUS

DIGITAL CIRCUIT AND COMPUTER ORGANIZATION LABORATORY

[As per Choice Based Credit System (CBCS) scheme]

(Effective from the academic year 2022 -2023)

SEMESTER – IV

Course Code	**BCS302**	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	3:0:2:0	SEE Marks	50
Total Hours of Pedagogy	40 hours Theory + 20 Hours of Practicals	Total Marks	100
Credits	**04**	Exam Hours	03

Course Learning Objectives:

CLO 1: To demonstrate the functionalities of binary logic system.

CLO 2: To explain the working of combinational and sequential logic system.

CLO 3: To realize the basic structure of computer system.

CLO 4: To illustrate the working of I/O operations and processing unit.

Teaching-Learning Process (General Instructions)

These are sample Strategies; that teachers can use to accelerate the attainment of the various course outcomes.

1. Chalk and Talk
2. Live Demo with experiments
3. Power point presentation

Module-1

Introduction to Digital Design: Binary Logic, Basic Theorems And Properties Of Boolean Algebra, Boolean Functions, Digital Logic Gates, Introduction, The Map Method, Four-Variable Map, Don't-Care Conditions, NAND and NOR Implementation, Other Hardware Description Language – Verilog Model of a simple circuit.

Text book 1: 1.9, 2.4, 2.5, 2.8, 3.1, 3.2, 3.3, 3.5, 3.6, 3.9

Module-2

Combinational Logic: Introduction, Combinational Circuits, Design Procedure, Binary Adder - Subtractor, Decoders, Encoders, Multiplexers. HDL Models of Combinational Circuits – Adder, Multiplexer, Encoder. **Sequential Logic:** Introduction, Sequential Circuits, Storage Elements: Latches, Flip-Flops.

Text book 1: 4.1, 4.2, 4.4, 4.5, 4.9, 4.10, 4.11, 4.12, 5.1, 5.2, 5.3, 5.4

Module-3

Basic Structure of Computers: Functional Units, Basic Operational Concepts, Bus structure, Performance – Processor Clock, Basic Performance Equation, Clock Rate, Performance Measurement. **Machine Instructions and Programs:** Memory Location and Addresses, Memory Operations, Instruction and Instruction sequencing, Addressing Modes.

Text book 2: 1.2, 1.3, 1.4, 1.6, 2.2, 2.3, 2.4, 2.5

Module-4

Input/output Organization: Accessing I/O Devices, Interrupts – Interrupt Hardware, Enabling and Disabling Interrupts, Handling Multiple Devices,

Direct Memory Access: Bus Arbitration, Speed, size and Cost of memory systems. Cache Memories – Mapping Functions.

Text book 2: 4.1, 4.2.1, 4.2.2, 4.2.3, 4.4, 5.4, 5.5.1

Module-5

Basic Processing Unit: Some Fundamental Concepts: Register Transfers, Performing ALU operations, fetching a word from Memory, Storing a word in memory. Execution of a Complete Instruction. **Pipelining:** Basic concepts, Role of Cache memory, Pipeline Performance.

Text book 2: 7.1, 7.2, 8.1

PRACTICAL COMPONENT OF IPCC

Sl. No	Experiments Simulation packages preferred: Multisim, Modelsim, PSpice or any other relevant
1	Given a 4-variable logic expression, simplify it using appropriate technique and simulate the sameusing basic gates.
2	Design a 4 bit full adder and subtractor and simulate the same using basic gates.
3	Design Verilog HDL to implement simple circuits using structural, Data flow and Behavioural model.
4	Design Verilog HDL to implement Binary Adder-Subtractor – Half and Full Adder, Half and Full Subtractor.
5	Design Verilog HDL to implement Decimal adder.
6	Design Verilog program to implement Different types of multiplexer like 2:1, 4:1 and 8:1.
7	Design Verilog program to implement types of De-Multiplexer.
8	Design Verilog program for implementing various types of Flip-Flops such as SR, JK and D.

Course outcome (Course Skill Set)

At the end of the course, the student will be able to:

CO1: Apply the K–Map techniques to simplify various Boolean expressions.

CO2: Design different types of combinational and sequential circuits along with Verilog programs.

CO3: Describe the fundamentals of machine instructions, addressing modes and Processor performance.

CO4: Explain the approaches involved in achieving communication between processor and I/O devices.

CO5: Analyze internal Organization of Memory and Impact of cache/Pipelining on Processor Performance.

Suggested Learning Resources:

<u>Textbooks</u>

1. M. Morris Mano & Michael D. Ciletti, Digital Design With an Introduction to Verilog Design, 5e, Pearson Education.
2. Carl Hamacher, ZvonkoVranesic, SafwatZaky, Computer Organization, 5th Edition, Tata McGraw Hill.

Weblinks and Video Lectures (e-Resources):

https://cse11-iiith.vlabs.ac.in/

Activity Based Learning (Suggested Activities in Class)/ Practical Based learning

Assign the group task to Design the various types of counters and display the output accordingly
Assessment Methods

- Lab Assessment (25 Marks)
- GATE Based Aptitude Test

COURSE OUTCOMES

At the end of the course the student will be able to:

CO1: Apply the K–Map techniques to simplify various Boolean expressions.

CO2: Design different types of combinational and sequential circuits along with Verilog programs.

CO3: Describe the fundamentals of machine instructions, addressing modes and Processor performance.

CO4: Explain the approaches involved in achieving communication between processor and I/O devices.

CO5: Analyze internal Organization of Memory and Impact of cache/Pipelining on Processor Performance.

Mapping of Course Outcomes with POs & PSOs

Course Outcomes (COs)	**Program Outcomes (POs)**												**Program Specific Outcomes(PSOs)**		
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**1**	**2**	**3**
CO1	2	2	-	-	3	-	-	-	1	1	-	-	1	-	-
CO2	2	2	2	2	3	-	-	-	1	1	-	-	1	-	-
CO3	2	2	-	-	-	-	-	-	-	-	-	-	1	-	-
CO4	2	2	-	-	-	-	-	-	-	-	-	-	-	1	-
CO5	2	2	2	-	-	-	-	-	-	-	-	-	-	1	-
Average	**2**	**2**	**2**	**2**	**3**	-	-	-	**1**	**1**	-	-	**1**	**1**	-

ASSESSMENT DETAILS (BOTH CIE AND SEE)

- The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks).
- A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.
- IPCC means practical portion integrated with the theory of the course.
- CIE marks for the theory component are **25 marks** and that for the practical component is **25 marks**.

CONTINUOUS INTERNAL EVALUATION (CIE) for IPCC

CIE THEORY + CIE PRACTICAL

❖ **CIE for the theory component of the IPCC (Maximum Marks 25)**

- **25 marks for the theory component** are split into
  - **15 marks** for two Internal Assessment Tests (Average of Two Tests each of 25 Marks , scale down the marks scored to 15 marks)
    - The first test at the end of 40-50% coverage of the syllabus and
    - The second test after covering 85-90% of the syllabus
  - **10 marks** for other assessment methods mentioned in 22OB4.2.
- Scaled-down marks of the sum of two tests and other assessment methods will be CIE marks for the theory component of IPCC (that is for 25 marks).
- The student has to secure 40% of 25 marks (10 marks) to qualify in the CIE of the theory component of IPCC.

❖ **CIE for the practical component of the IPCC (Maximum Marks 25)**

- **25 marks for the practical component** are split into
  - **15 marks** for the conduction of the experiment and preparation of laboratory record, and **10 marks** for the test to be conducted after the completion of all the laboratory sessions.
  - On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voce and marks shall be awarded on the same day.

- The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluation of the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to **15 marks.**
- The laboratory test (duration 02/03 hours) after completion of all the experiments shall be conducted for **50 marks** and scaled down to **10 marks.**
- Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IPCC **for 25 marks.**
- The student has to secure 40% of 25 marks to qualify in the CIE of the practical component of the IPCC.

Split-up of Marks used Practical Sessions

Rubrics No.	Practical Sessions- Continuation Evaluation (CE) Methodology / Process Steps per Experiment	Marks
#R1	Observation	10
#R2	Record writing : Write up of Procedure / Algorithm/ Program and Execution/conduction of experiment	30
#R3	Viva – Voce (Questions & Answers on relevant Experiment /Topic)	10
Total Marks (Note: Conduction of experiment's and Preparation of Laboratory records etc. for 50 Marks scale down the marks scored to 15 marks)		**50 (Scale down to 15)**
Rubrics No.	**Practical Sessions-Internal Assessment (IA)**	**Marks**
#R1	Write-up of Procedure/Program/Algorithm	10
#R2	Conduction/Execution	30
#R3	Viva-Voce	10
Total Marks (Note: One test after all experiment's conduction for 50 Marks scale down the marks scored to 10 marks)		**50 (Scale down to 10)**

SEMESTER END EXAMINATION (SEE) for IPCC:

Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the course (**duration 03 hours**)

1. The question paper will have ten questions. Each question is set for 20 marks.
2. There will be 2 questions from each module. Each of the two questions under a module (with a maximum of 3 sub-questions), **should have a mix of topics** under that module.
3. The students have to answer 5 full questions, selecting one full question from each module.
4. Marks scored by the student shall be proportionally scaled down to 50 Marks

The theory portion of the IPCC shall be for both CIE and SEE, whereas the practical portion will have a CIE component only. Questions mentioned in the SEE paper may include questions from the practical component.

To put it simply, evaluation techniques/methods are listed in the table for further understanding

Evaluation Type	Maximum Marks	Minimum Passing Marks	Evaluation Details
CIE-IA Tests	15	06	Average of Two Tests each of 25 Marks , scale down the marks scored to 15 marks
CIE-CCAs	10	04	Any two assignment methods as per clause 22OB4.2 of Regulations (if assessment is project based, then one assessment method may be adopted)
Total CIE Theory	**25**	**10**	**Scale down marks of tests and assignments to 25**
CIE Practical	15	06	Conduction of experiment's and Preparation of Laboratory records etc
CIE Practical Test	10	04	One test after all experiment's conduction for 50 Marks scale down the marks scored to 10 marks
Total CIE Practical	**25**	**10**	**Scale down marks of experiment's record and test to 25**
TOTAL CIE= Total CIE Theory + Total CIE Practical	**50**	**20**	
SEE	**50**	**18**	SEE exam is a theory exam, conducted for 100 marks are scaled down to 50 marks
CIE+SEE	**100**	**40**	

RUBRICS FOR PRACTICAL SESSIONS

Course: **Digital Design and Computer Organization** **Course Code:** **BCS302**

Practical Sessions- Continuous Evaluation (CE)				
Evaluation Parameter	**Level of Achievement**			
#R1:	**Excellent (10-8)**	**Good (7-6)**	**Average (5-4)**	**Poor (3-0)**
Observation (10 Marks)	Observation neatly written. Handwriting is clear. Programs written with no mistakes.	Observation neatly written. Handwriting is clear. Programs written with very few mistakes.	Observation is written in unclear manner. Handwriting is not very clear.	Observation is written in unclear manner. Handwriting is not clear. Programs executed with a large number of errors.
#R2:	**Excellent (30-26)**	**Good (25-20)**	**Average (19-15)**	**Poor (14-0)**
Record writing : Write up of Procedure / Algorithm/ Program and Execution/conduction of experiment **(30 Marks)**	Record is neatly written, handwriting is clear. Programs executed with no errors. Mistakes are covered and corrected properly and neatly. Record submitted on time	Record is neatly written, handwriting is clear. Programs executed with very less errors. Most mistakes are covered and corrected properly and neatly. Record submitted with a delay of 1 - 3 days	Record is written in an unclear manner. Programs executed with few errors Handwriting is not very clear. Mistakes are sometimes corrected properly. Record submitted with a delay of 4 to 5 days	Record is written in an unclear manner. Programs written with lot of mistakes. Handwriting is not very clear. Mistakes are not corrected. Record submitted after a delay of 1 week
#R3:	**Excellent (10-8)**	**Good (7-6)**	**Average (5-4)**	**Poor (3-0)**
Viva (10 Marks)	Answered all questions with elaboration has excellent understanding of the topic.	Answered most of the questions Failed to elaborate some of the concepts	Answered a few questions. Subject knowledge is not adequate	Not able to answer any of the questions. Subject knowledge not adequate
Each week experiment report is evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to **15 marks.**				
Practical Sessions- Internal Assessment (IA)				
#R1:	**Excellent (10-8)**	**Good (7-6)**	**Average (5-4)**	**Poor (3-0)**
Write-Up (10 Marks)	Program neatly written. Handwriting is clear. Programs	Program neatly written. Handwriting is clear.	Program is written in unclear manner. Handwriting is	Program is written in unclear manner. Handwriting is

	written with no mistakes.	Programs written with very few mistakes.	not very clear. Programs written with fewer mistakes.	not clear, Programs written with lot of mistakes.
#R2: Conduction/Execution (30 Marks)	**Excellent (30-26)**	**Good (25-20)**	**Average (19-15)**	**Poor (14-0)**
	Execution of the program done as per the procedure. Programs had less than 10 errors. The errors were debugged without any help. The Result was tabulated for all the cases.	Execution of the program done as per the procedure. Programs had less than 20 errors. The errors were debugged with a little help. The Result was tabulated for almost all the cases	Execution of the program done as per the procedure. Programs had more than 20 errors. The errors were debugged with the help of instructor. The Result was tabulated for few of the cases	Execution of the program was not done as per the procedure. Programs was full of syntax and logical error. The errors were resolved by the instructor. The Result was tabulated only for 1 or 2 Cases
#R3: Viva (10 Marks)	**Excellent (10-8)**	**Good (7-6)**	**Average (5-4)**	**Poor (3-0)**
	Answered all questions with elaboration has excellent understanding of the topic.	Answered most of the questions Failed to elaborate some of the concepts	Answered a few questions. Subject knowledge is not adequate	Not able to answer any of the questions. Subject knowledge not adequate
The laboratory test **(duration 02/03 hours)** at the end of the 15th week of the semester /after completion of all the experiments (whichever is early) shall be conducted for 50 marks and scaled down to **10 marks.**				

LIST OF MAJOR EQUIPMENT

Sl. No.	Name of the Equipment	Specialization	Quantity
1	COMPUTER	CORE I5 8GB RAM 160GB HDD 17inch MONITOR	30
2	SOFTWARE	MODEL SIM (WINDOWS 10)	30
3	White board	-	1

Room Number	**:**	**B-201A**
Name of the Faculty	**:**	Prof. Harshitha M
Name of the lab instructor	**:**	Mr. Mukund

LIST OF EXPERIMENT/PROGRAMS

Additional Experiment/Programs- Content beyond Syllabus

1. Verification of the truth tables of gates using Digital Trainer Kit.
2. Verify the NAND and NOR gates as universal logic gates Digital Trainer Kit.
3. Design and verification of the truth tables of Half and Full adder circuits Digital Trainer Kit.

INTRODUCTION

Introduction to the Modelsim Interface

This tutorial is designed to familiarize you with Verilog coding/syntax and simulation in the ModelSim environment. Verilog HDL is a hardware description language used to design digital systems. Along with VHDL, Verilog is the primary industry tool for programming digital systems. ModelSim is the industry standard simulation tool for verifying digital designs.

Installation steps are given below: You can access ModelSim either through the PCs in the lab or an Athena Sun Workstation.

1. **Creating files in ModelSim**
   There are two ways to start creating your designs in ModelSim: 1.Creating a project or 2.Opening or creating a Verilog file without a project.

   1. Creating a project Create a new project:
      File > New > Project

   2. Creating without a project:
      File > New > Source > Verilog or open an existing Verilog file: File > Open > File

2. **Compiling in ModelSim Compile source files:**
   Compile > Compile

3. **Simulation**:
   You are now ready to simulate the counter module using the testbench.

4. **Viewing Simulations – The Wave Window**

Useful Buttons and Command Lines

Buttons	Description*	Command Line

	Compile this file
	open the Compile Source File dialog; in a project, compile the file
		To compile:
		`vlog -work [working library] [filename.v]`
		`ex) vlog -work work tb_tutorial.v`
		
		To recompile:
		`vlog -work [working library] -refresh`

	Compile All compile all files in the open project	Compile > Compile All

	Simulate
	load the selected design unit or simulation configuration object
		`vsim -c [working library].[file name]`
		
		`Ex) vsim -c work.tb_tutorial`

	Restart reload the design elements and reset the simulation time to zero, with the option of using current formatting, breakpoints, WLF file, virtual definitions, and assertion settings	`restart -f`
	Run run the current simulation for the specified run length	`run`
	Open Wave Viewer	Window > Wave
	Zoom Mode set mouse to Zoom Mode – drag left mouse button to zoom, click middle mouse button to select	N/A
	Insert Cursor add a cursor to the waveform pane	Insert > Cursor

EXPERIMENT NO – 01

Given a 4-variable logic expression, simplify it using appropriate technique and simulate the same using basic gates.

Aim: Given a 4-variable logic expression, simplify it using appropriate technique and simulate thesame using basic gates.

Objectives:

1. Simplify 4-variable logic expression for efficient design.
2. Simulate using basic gates for practical verification and optimization.

E=A'B'C'D'+A'B'C'D+A'B'CD+A'BC'D+A'BCD+AB'C'D'+AB'C'D+AB'CD+ABC'D+A B CD

K-Map Simplification

AB/CD	C'D'	C'D	CD	CD'
A'B'	1	1	1	0
A'B	0	1	1	0
AB	0	1	1	0
A'B'	1	1	1	0

E=D+B'C'

Verilog Code for above Expression:

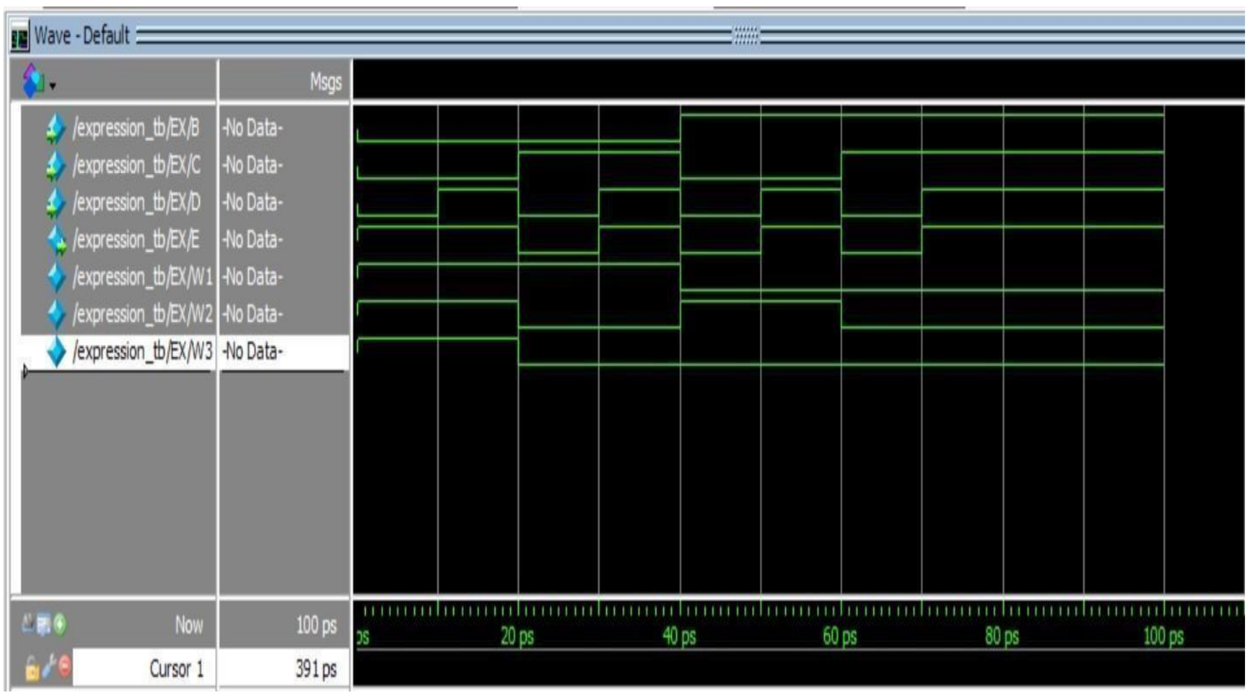
```
module expression(B,C,D,E);
input B,C,D;
output E;
wire W1,W2,W3;
not(W1,B);
not(W2,C);
and(W3,W1,W2);
or(E,D,W3);
endmodule

module expression_tb();
reg B,C,D;
wire E;
expression  EX(.B(B),.C(C),.D(D),.E(E));
```

```
initialbegin

B=0;    C=0;    D=0;
#10;    B=0;    C=0;
D=1;    #10;    B=0;
C=1;    D=0;    #10;
B=0;    C=1;    D=1;
#10;    B=1;    C=0;
D=0;    #10;    B=1;
C=0;    D=1;    #10;
B=1;    C=1;    D=0;
#10; B=1; C=1; D=1;
#10;
end
endmodule
```

Output


Wave - Default
Msgs
/expression_tb/EX/B -No Data-
/expression_tb/EX/C -No Data-
/expression_tb/EX/D -No Data-
/expression_tb/EX/E -No Data-
/expression_tb/EX/W1 -No Data-
/expression_tb/EX/W2 -No Data-
/expression_tb/EX/W3 -No Data-
Now 100 ps
Cursor 1 391 ps
20 ps
40 ps
60 ps
80 ps
100 ps

Possible viva questions:

1. **What is the purpose of simplifying a logic expression?**

   Answer: Simplifying a logic expression reduces the number of gates needed, leading to more efficient and faster circuits.

2. **How many cells does a 4-variable K-map have?**

   Answer: A 4-variable K-map has 16 cells (2^4).

3. **Explain the steps involved in simplifying a Boolean expression using K-maps.**

   Answer: Identify minterms, group adjacent 1s in the K-map, find the minimal sum-of-products (SOP) expression, and then apply any additional simplifications.

4. **Given a 4-variable expression ABCD + A'C'D + A'BD', simplify it using a K-map.**

   Answer: The simplified expression is AD' + A'C'.

5. **Given the expression (A + B)(C + D), simplify it using Boolean algebra.**

   Answer: AC + AD + BC + BD

6. **What the significance is of don't care conditions in logic simplification?**

Answer: Don't care conditions represent input combinations for which the output value is not critical. Utilizing don't care conditions allows for further optimization during simplification.

EXPERIMENT NO – 02

Design a 4 bit full adder and subtractor and simulate the same using basic gates.

Aim: To design a 4 bit full adder and subtractor and simulate the same using basic gates.

Objectives:

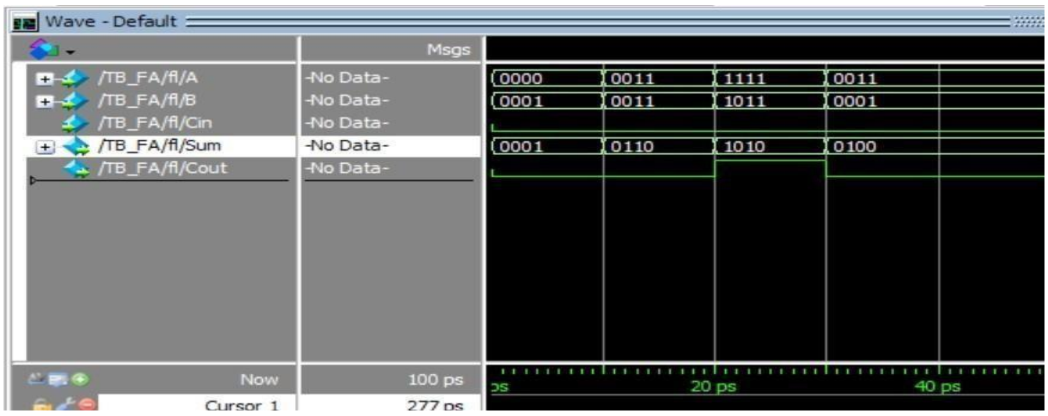
1. Design a 4-bit full adder and subtractor circuit in Modelsim using basic gates.
2. Simulate the circuit to analyze and verify correct functionality, considering various inputscenarios.

//4-bit fulladder

```
module
fulladder_4bit(A,B,Cin,Sum,Cout);
input [3:0]A,B;
input Cin;
output
[3:0]Sum;
output
Cout;
assign
{Cout,Sum}=A+B+Cin;
endmodule

moduleTB_FA();
reg [3:0]a,b;
reg cin;
wire
[3:0]sum;
wire cout;
fulladder_4bit
fl(.A(a),.B(b),.Cin(cin),.Sum(sum),.Cout(cout));
initial begin
a=4'b0000;b=4'b0001;cin=1'b0;#10;
a=4'b0011;b=4'b0011;cin=1'b0;#10;
a=4'b1111;b=4'b1011;cin=1'b0;#10;
a=4'b0011;b=4'b0001;cin=1'b0;#10;
end
endmodule
```

Output

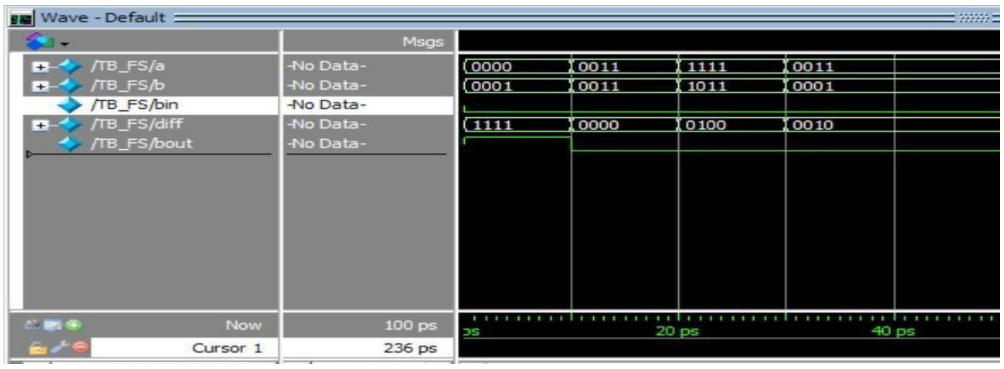


//4-bit fullsubtractor

```
module fullsub_4bit(A,B,Bin,Diff,Bout);
input [3:0]A,B;
input  Bin;
output [3:0]Diff;
output Bout;
assign {Bout,Diff}=A-B-Bin;
endmodule

module TB_FS();reg
[3:0]a,b;
reg bin;
wire [3:0]diff;
wire bout;
fullsub_4bit fs(.A(a),.B(b),.Bin(bin),.Diff(diff),.Bout(bout));initial
begin
a=4'b0000;b=4'b0001;bin=1'b0;#10;
a=4'b0011;b=4'b0011;bin=1'b0;#10;
a=4'b1111;b=4'b1011;bin=1'b0;#10;
a=4'b0011;b=4'b0001;bin=1'b0;#10;
end
endmodule
```

Output



Possible viva questions:

1. **What is the primary function of a 4-bit full adder?**
   A 4-bit full adder performs addition on two 4-bit binary numbers, considering inputs and generating a sum along with a carry-out.

2. **Can a 4-bit full adder handle subtraction as well?**
   Yes, a 4-bit full adder can be used for subtraction by incorporating additional logic to manage borrow inputs and adjustments.

3. **How many control inputs are there in a 4-bit full subtractor for each XOR gate used?**

4. **What is the purpose of the AND gates in a full subtractor?**
   A) Generate the sum output
   B) Generate the borrow-out
   C) Perform bitwise AND operation
   D) Implement overflow detection

   Answer: C) Performbitwise AND operation

5. **List the basic gates required to design a 4-bit full adder.**
   Answer: Basic gates include AND gates, XOR gates, and OR gates.

6. **In a 4-bit full subtractor, what is the role of the borrow-out?**
   A) It indicates a borrow to the next stage.
   B) It determines the most significant bit of the difference.
   C) It controls the XOR gates.
   D) It is not used in subtraction.

   Answer: A) It indicates a borrow to the next stage.

7. **What is the purpose of the AND gates in a full subtractor?**
   A) Generate the sum output
   B) Generate the borrow-out
   C) Perform bitwise AND operation
   D) Implement overflow detection Answer: C) Perform bitwise AND operation

8. **How would you simulate the 4-bit full adder and subtractor using basicgates in asoftware tool like Multisim or similar?**
   Answer: Create a schematic in the simulation tool, instantiate the required basic gates (AND,XOR, OR), and interconnect them according to the design of the 4-bitfull adder and subtractor. Apply appropriate inputs and observe the outputs.

9. **How many XOR gates are required for the sum output in a 4-bit full adder?**
   Answer: Four XOR gates are required for generating the sum outputs in a 4-bit full adder.

10. **Can you identify any challenges or considerations when designing a 4-bit fullsubtractor?**
    Answer: One consideration is managing borrow inputs for each bit subtraction, and carefulhandling of borrow propagation to ensure accurate subtraction results

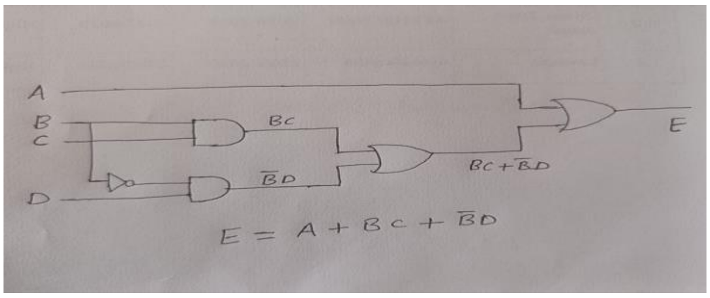
EXPERIMENT NO – 03

Design Verilog HDL to implement simple circuits using structural, Data flow and Behavioural model.

Aim: To design Verilog HDL to implement simple circuits using structural, Data flow andBehavioural model.

Objectives:

1. Implement Verilog for simple circuits using structural, data flow, and behavioral models.
2. Verify designs through simulation for functionality and performance assessment.



E=A+BC+B'D

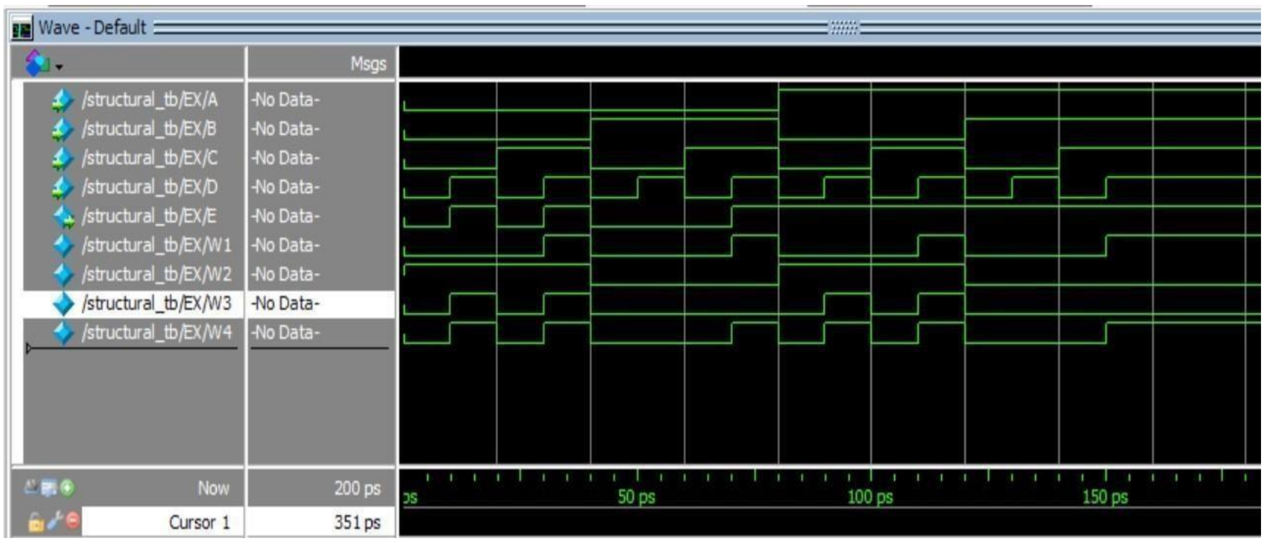
Structural Modelling

```
module structural(A,B,C,D,E);
input A,B,C,D;
output E;
wire W1,W2,W3,W4;
and(W1,D,C);
not(W2,B);
and(W3,W2,D);
or(W4,W1,W3);
or(E,A,W4);
endmodule

module structural_tb();
reg A,B,C,D;
wire E;
structural EX(.A(A),.B(B),.C(C),.D(D),.E(E));
initilbegin
A=0; B=0; C=0; D=0; #10;
B=1; C=0; D=1;  #10;
A=0; B=1; C=1; D=0; #10;
A=0; B=1; C=1; D=1; #10;
A=1; B=0; C=0; D=0; #10;
```

```
A=1; B=0; C=0; D=1; #10;
A=1; B=0; C=1; D=0; #10;
A=1; B=0; C=1; D=1; #10;
A=1; B=1; C=0; D=0; #10;
A=1; B=1; C=0; D=1; #10;
A=1; B=1; C=1; D=0; #10;
A=1; B=1; C=1; D=1; #10;
end
endmodule
```

Output


Wave - Default
Msgs
/structural_tb/EX/A -No Data-
/structural_tb/EX/B -No Data-
/structural_tb/EX/C -No Data-
/structural_tb/EX/D -No Data-
/structural_tb/EX/E -No Data-
/structural_tb/EX/W1 -No Data-
/structural_tb/EX/W2 -No Data-
/structural_tb/EX/W3 -No Data-
/structural_tb/EX/W4 -No Data-
Now 200 ps
Cursor 1 351 ps
50 ps
100 ps
150 ps

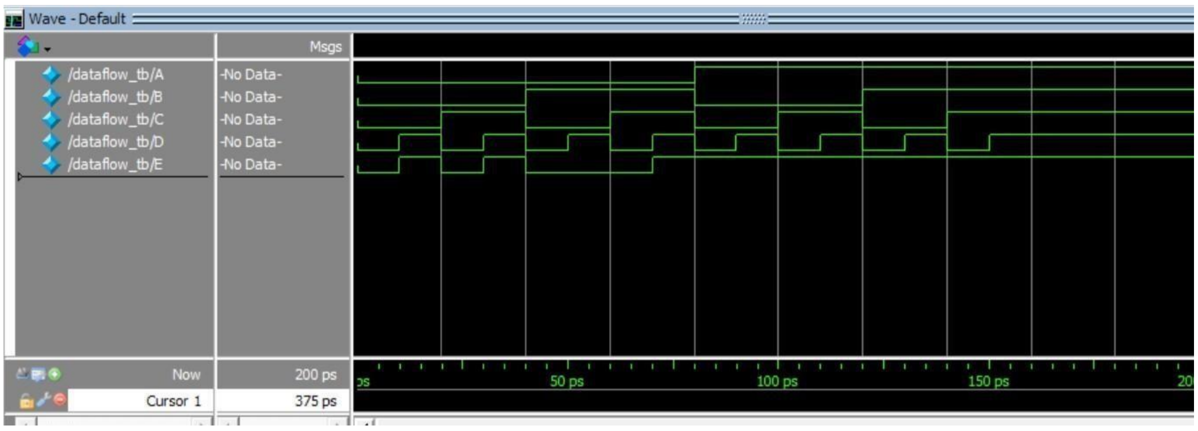
<u>DataFlow Modelling</u>

```
module dataflow(A,B,C,D,E);input
A,B,C,D;
output E;
assign E = A|(D&C)|(~B&D);
endmodule

module dataflow_tb();reg
A,B,C,D;
wire E;
dataflow EX(.A(A),.B(B),.C(C),.D(D),.E(E));
initial
begin
A=0; B=0; C=0; D=0; #10;
A=0; B=0; C=0; D=1; #10;
A=0; B=0; C=1; D=0; #10;
A=0; B=0; C=1; D=1; #10;
A=0; B=1; C=0; D=0; #10;
A=0; B=1; C=0; D=1; #10;
A=0; B=1; C=1; D=0; #10;
```

```
A=0; B=1; C=1; D=1; #10;
A=1; B=0; C=0; D=0; #10;
A=1; B=0; C=0; D=1; #10;
A=1; B=0; C=1; D=0; #10;
A=1; B=0; C=1; D=1; #10;
A=1; B=1; C=0; D=0; #10;
A=1; B=1; C=0; D=1; #10;
A=1; B=1; C=1; D=0; #10;
A=1; B=1; C=1; D=1; #10;
end
endmodule
```

Output:



Possible viva questions:

1. **What is Verilog HDL?**
   Verilog Hardware Description Language (HDL) is a language used for modeling electronic systems at the behavioral and structural levels, commonly employed in digital design and simulation.
2. **Explain the structural modeling in Verilog.**
   Structural modeling involves building circuits using predefined modules. Instances of modules are interconnected to create a complete design.

3. **Explain the concept of continuous assignment in data flow modeling.**
   Continuous assignments use the assign keyword to express the relationship between signals. These assignments operate continuously, updating the output whenever input signals change.

4. **In behavioral modeling, what does the always block signify?**
   The always block in behavioral modeling defines a set of conditions and statements that are executed whenever the conditions specified in the sensitivity list change.

5. **Can Verilog be used for both simulation and synthesis?**
   Yes, Verilog is versatile and can be used for both simulation, to verify designs, and synthesis, to generate hardware based on the design description

6. **How are signals represented in behavioral modeling in Verilog?**
   In behavioral modeling, signals are represented using variables and expressions. Changes in variables trigger the execution of blocks, simulating the algorithmic behavior.

7. **How is time accounted for in behavioral modeling in Verilog?**
   Behavioral modeling can include delays using constructs like # to represent time delays between statements. This allows simulation to account for time-dependent behavior.

8. **How does structural modeling differ from behavioral modeling in Verilog?**
   Structural modeling focuses on the interconnection of modules, representing the physical structure of the circuit. Behavioral modeling emphasizes the algorithmic functionality, abstracting the physical implementation details.

9. **In which scenarios would you prefer to use structural modeling in Verilog?**
   Structural modeling is beneficial when designing circuits with well-defined modules and known physical implementations, making it suitable for hierarchical designs.

10. **When is behavioral modeling more appropriate in Verilog?**
    Behavioral modeling is preferred for abstractly describing the functionality of a circuit without concern for the specific physical structure. It is useful for algorithmic or higher-level design descriptions.

EXPERIMENT NO – 04

Design Verilog HDL to implement Binary Adder-Subtractor – Half and Full Adder, Half and Full Subtractor.

Aim: To Design Verilog HDL to implement Binary Adder-Subtractor – Half and Full Adder, Half and Full Subtractor.

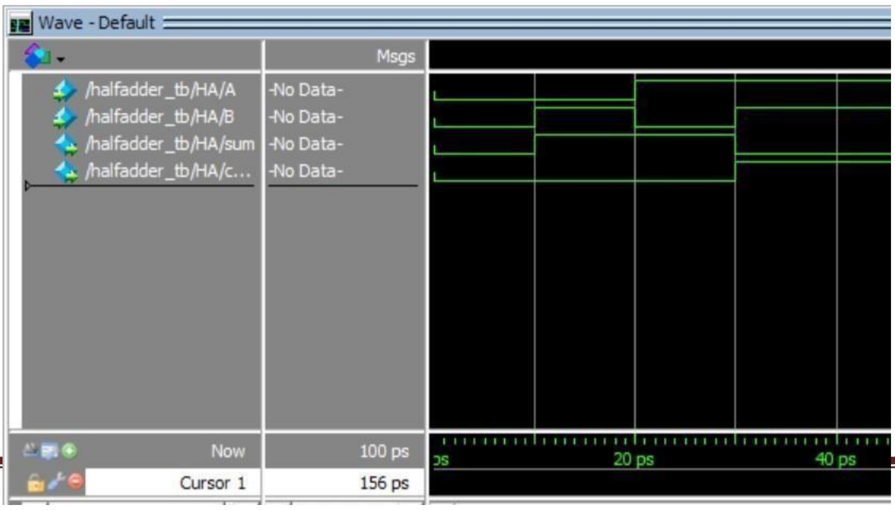
Objectives:

1. Create Verilog HDL for Binary Adder-Subtractor, including Half and Full Adder, Half and Full Subtractor.
2. Validate functionality through simulation for accurate arithmetic operations in digital systems.

//Halfadder

```
module halfadder(A,B,sum,carry);
input A,B;
output sum, carry;
assign sum = A^B;
assign carry = A&B;
endmodule

module halfadder_tb();
reg A,B;
wire sum, carry;
halfadder HA(.A(A),.B(B),.sum(sum),.carry(carry));
initial begin
A=0; B=0; #10;
A=0; B=1; #10;
A=1; B=0; #10;
A=1; B=1; #10;
end endmodule
```

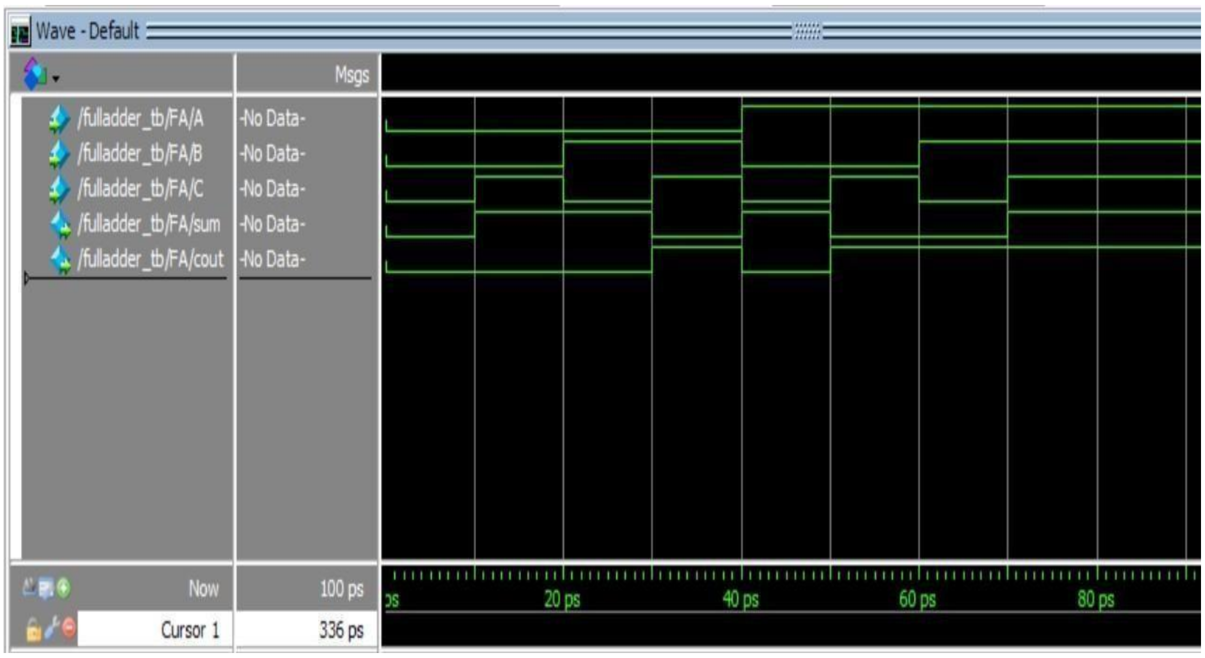

Wave - Default
Msgs
/halfadder_tb/HA/A -No Data-
/halfadder_tb/HA/B -No Data-
/halfadder_tb/HA/sum -No Data-
/halfadder_tb/HA/c... -No Data-
Now 100 ps
Cursor 1 156 ps
20 ps
40 ps

//Fulladder

```
module fulladder(A,B,C,sum,cout);
input A,B,C;
output sum, cout;
assign sum = A^B^C;
assign cout = (A&B)|(B&C)|(C&A);
endmodule

module fulladder_tb();
reg A,B,C;
wire sum, cout;
fulladder FA(.A(A),.B(B),.C(C),.sum(sum),.cout(cout));
initial begin
A=0; B=0; C=0; #10;
A=0; B=0; C=1; #10;
A=0; B=1; C=0; #10;
A=0; B=1; C=1; #10;
A=1; B=0; C=0; #10;
A=1; B=0; C=1; #10;
A=1; B=1; C=0; #10;
A=1; B=1; C=1; #10;
end endmodule
```

Output

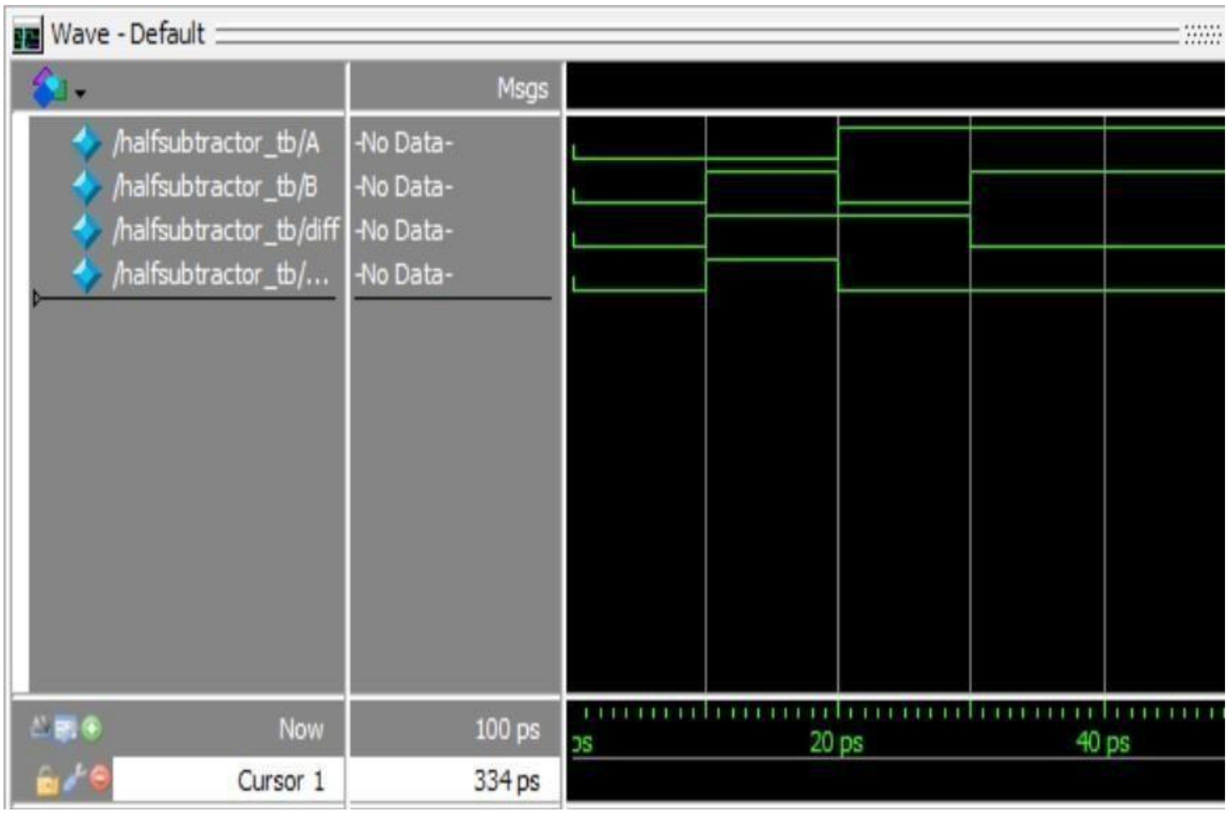

Wave - Default
Msgs
/fulladder_tb/FA/A -No Data-
/fulladder_tb/FA/B -No Data-
/fulladder_tb/FA/C -No Data-
/fulladder_tb/FA/sum -No Data-
/fulladder_tb/FA/cout -No Data-
Now 100 ps
Cursor 1 336 ps
20 ps
40 ps
60 ps
80 ps

//Halfsubtractor

```
module halfsubtractor(A,B,diff,borrow);
input A,B;
output diff, borrow;
assign diff = A^B;
assign borrow = ~A&B;
endmodule

module halfsubtractor_tb();
reg A,B;
wire diff, borrow;
halfsubtractor HS(.A(A),.B(B),.diff(diff),.borrow(borrow));
initial begin
A=0; B=0; #10;
A=0; B=1; #10;
A=1; B=0; #10;
A=1; B=1; #10;
end
endmodule
```

Output

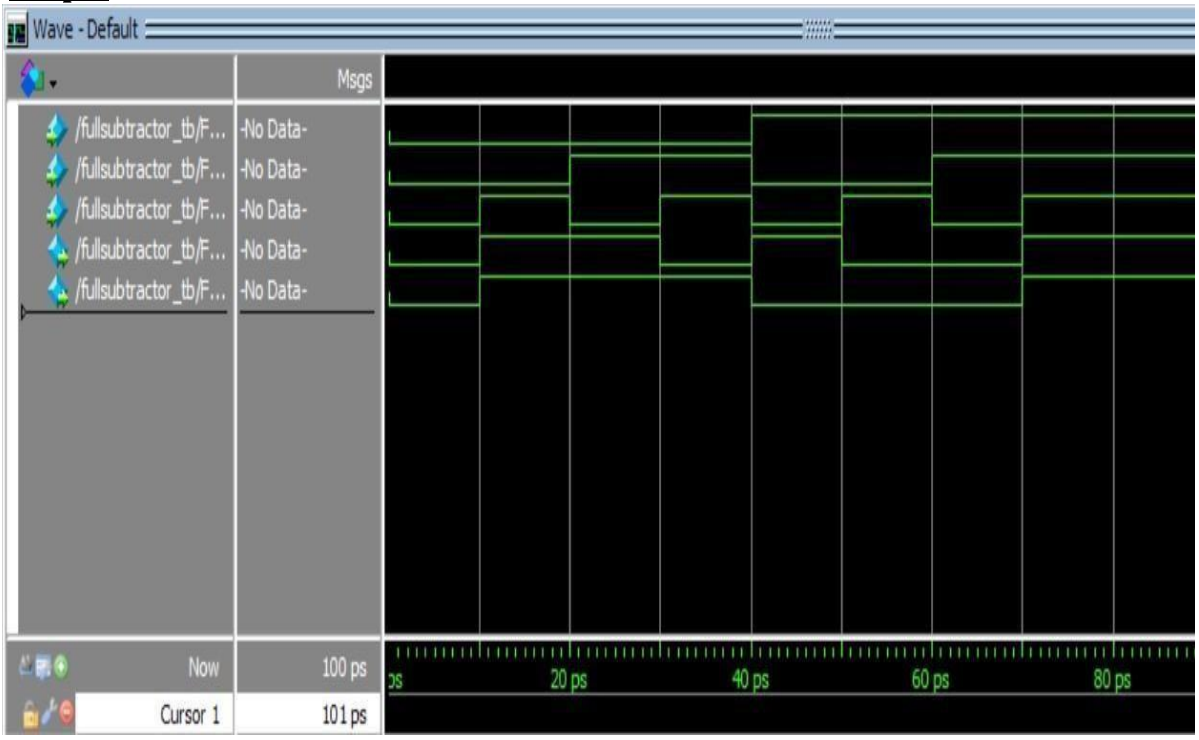

Wave - Default
Msgs
/halfsubtractor_tb/A -No Data-
/halfsubtractor_tb/B -No Data-
/halfsubtractor_tb/diff -No Data-
/halfsubtractor_tb/... -No Data-
Now 100 ps
Cursor 1 334 ps
ps
20 ps
40 ps

//Fullsubtractor

```
module fullsubtractor(A,B,C,diff,bout);
input A,B,C;
output diff, bout;
assign diff = A^B^C;
assign bout = (~A&B)|(B&C)|(~A&C);
endmodule

module fullsubtractor_tb();
reg A,B,C;
wire diff, bout;
fullsubtractor FS(.A(A),.B(B),.C(C),.diff(diff),.bout(bout));
initial begin
A=0; B=0; C=0; #10;
A=0; B=0; C=1; #10;
A=0; B=1; C=0; #10;
A=0; B=1; C=1; #10;
A=1; B=0; C=0; #10;
A=1; B=0; C=1; #10;
A=1; B=1; C=0; #10;
A=1; B=1; C=1; #10;
end
endmodule
```

Output



.

Possible viva questions:

1. **What is the basic function of a binary adder?**

Ans: A binary adder combines two binary numbers, producing their sum and a possible carry output.

2. **What is a Half Adder in binary arithmetic?**

Ans: A Half Adder adds two binary digits (bits) and produces a sum and a carry-out. It lacks the ability to handle a carry-in from a previous stage.

3. **How does a Full Adder differ from a Half Adder?**

Ans: A Full Adder adds three binary digits (two inputs and a carry-in) to produce a sum and a carry-out, enabling it to handle carry inputs from previous stages.

1. **What is a Half Subtractor in binary arithmetic?**

Ans: A Half Subtractor subtracts two binary digits and produces a difference and a borrow-out. It does not consider borrow inputs from previous stages.

2. **How does a Full Subtractor differ from a Half Subtractor?**

Ans: A Full Subtractor subtracts three binary digits (two inputs and a borrow-in) to produce a difference and a borrow-out. It can handle borrow inputs from previous stages.

3. **How does a Half Subtractor handle the case when the minuend is smaller than the subtrahend?**

Ans: In such cases, the Half Subtractor produces a difference without a borrow-out, signifying that no borrowing is required for the subtraction.

4. **Explain the significance of the carry-in and borrow-in signals in Full Adders and Full Subtractors.**

Ans: The carry-in (Cin) in Full Adders and the borrow-in in Full Subtractors represent inputs from the previous stage, allowing these circuits to operate on multiple bits.

5. **Can a Full Subtractor be constructed using Half Subtractors?**

Ans: Yes, a Full Subtractor can be built using two Half Subtractors and additional logic to manage borrow inputs.

6. **How does the carry propagation differ in Adders and Subtractors?**

Ans: In Adders, carries propagate from lower bits to higher bits during addition, while in Subtractors, borrows propagate from higher bits to lower bits during subtraction.

7. **Why is it called a "Half" Adder or Subtractor?**

Ans: It is called "Half" because it handles only two input bits, not considering any carry or borrow inputs from previous stages, making it a simpler building block compared to "Full" Adders and Subtractors.

EXPERIMENT NO – 05

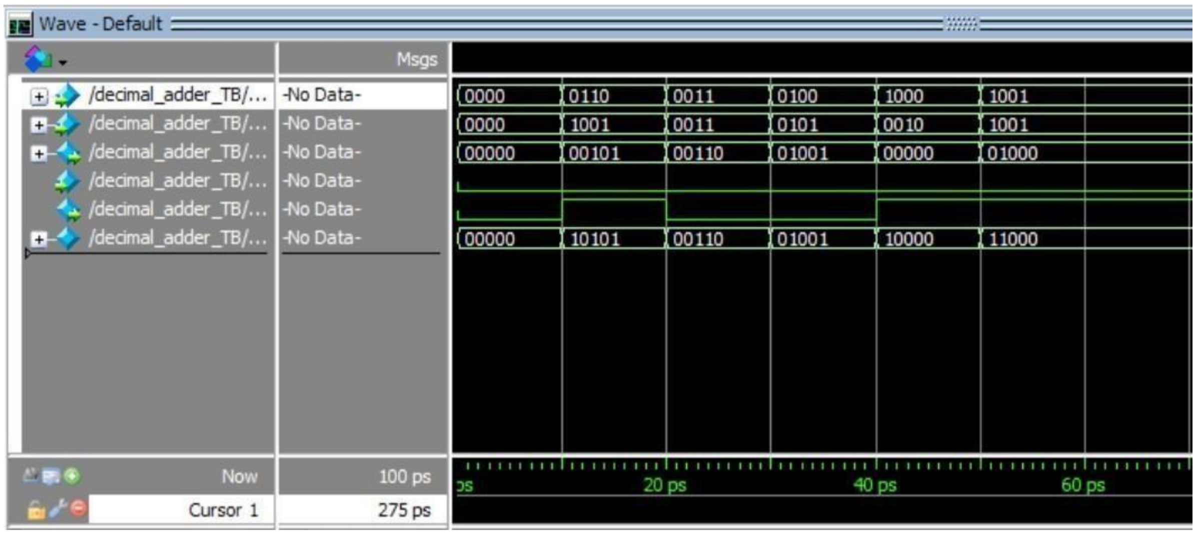
Design Verilog HDL to implement Decimal adder

Aim: To Design Verilog HDL to implement Decimal adder.

```
module decimal_adder(a,b,cin,sum,carry);
input [3:0]a,b;
output [3:0]sum;
input cin;
output carry;
 reg[4:0]sum_temp;
reg[4:0]sum;
reg carry;
always @(a,b,cin) begin
sum_temp=a+b+cin;
if(sum_temp>9) begin
sum_temp=sum_temp+6;
carry=1; sum=sum_temp[3:0];
end
else begin
carry=0;
sum=sum_temp[3:0];
end
end endmodule

module decimal_adder_TB();
reg [3:0]a;
reg [3:0]b;
reg cin;
wire [3:0]sum;
wire carry;
decimal_adder DA(.a(a),.b(b),.cin(cin),.sum(sum),.carry(carry));
initial begin
a=0; b=0; cin=0; #10;  a=6;
b=9;  cin=0;  #10;  a=3; b=3;
cin=0; #10; a=4; b=5; cin=0;
#10;  a=8;  b=2;  cin=0;  #10;
a=9; b=9; cin=0; #10;
end endmodule
```

Output:


Wave - Default
Msgs
/decimal_adder_TB/... -No Data- 0000 0110 0011 0100 1000 1001
/decimal_adder_TB/... -No Data- 0000 1001 0011 0101 0010 1001
/decimal_adder_TB/... -No Data- 00000 00101 00110 01001 00000 01000
/decimal_adder_TB/... -No Data-
/decimal_adder_TB/... -No Data-
/decimal_adder_TB/... -No Data- 00000 10101 00110 01001 10000 11000
Now 100 ps
Cursor 1 275 ps
ps 20 ps 40 ps 60 ps

<u>Possible viva questions:</u>

1. **What is a decimal adder?**
Ans: A decimal adder is a digital circuit designed to perform addition operations specifically for decimal numbers.

2. **How does a decimal adder differ from a binary adder?**
Ans: While binary adders operate on binary numbers, decimal adders process numbers in base-10, considering the decimal digits 0 through 9.

3. **Explain the significance of a carry in a decimal adder.**
Ans: In a decimal adder, a carry occurs when the sum of two digits exceeds 9, requiring an additional unit to be carried to the next higher decimal place.

4. **How is overflow handled in a decimal adder?**
Ans: Overflow occurs when the sum in a decimal place exceeds 9. In such cases, the carry is propagated to the next higher decimal place to prevent data loss.

5. **What are the applications of a decimal adder in digital systems?**
Ans: Decimal adders are essential components in applications like financial calculations, where decimal numbers are prevalent, ensuring accurate arithmetic operations in electronic systems.

6. **How does the addition process differ between binary and decimal adders?**
Ans: Decimal addition involves carrying over to the next decimal place when the sum exceeds 9, while binary addition carries over when the sum exceeds 1.

7. **Can a binary adder be adapted for decimal addition?**
Ans: Yes, but it may not efficiently handle decimal-specific carry propagation. Decimal adders are optimized for base-10 arithmetic, considering the unique carry requirements of decimal digits.

8. **Explain the role of the carry-out in a decimal adder.**
Ans: The carry-out signal indicates whether there is a carry from the most significant digit (MSD), signaling potential overflow in multi-digit decimal additions.

9. **How is subtraction implemented in a decimal adder circuit?**
Subtraction in a decimal adder is achieved by using additional logic to complement and adjust the inputs based on the borrow generated during the subtraction process.

10. **Can a decimal adder handle negative decimal numbers?**
Yes, by incorporating logic to manage sign bits and handling subtraction appropriately, a decimal adder can process negative decimal numbers.

EXPERIMENT NO – 06

Design Verilog program to implement Different types of multiplexer like 2:1, 4:1 and 8:1.

Aim: Design Verilog program to implement Different types of multiplexers like 2:1, 4:1 and 8:1.

Objectives:

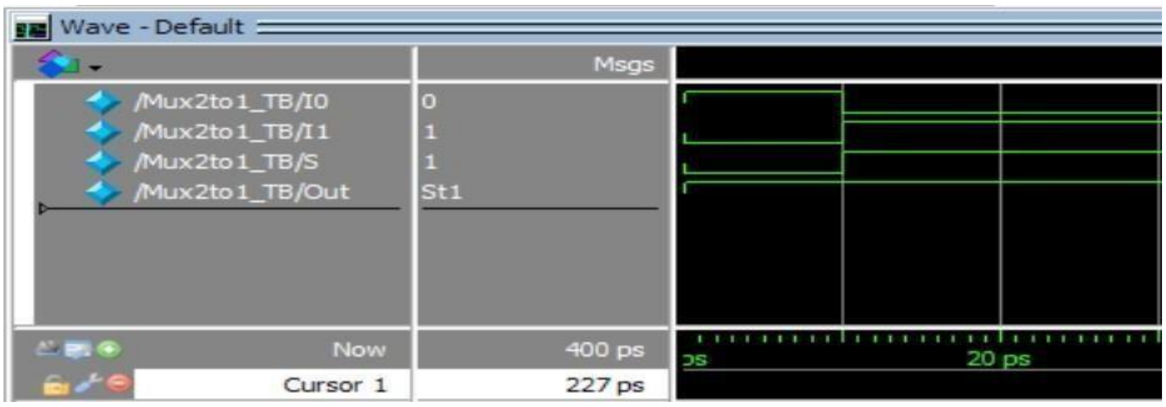
1. Develop Verilog program for 2:1, 4:1, 8:1 multiplexer design.
2. Validate through simulation for accuracy and efficiency in circuit implementation.

//2:1 muultiplexer

```
module Mux2to1(I0,I1,S,Out);
input I0,I1,S;
output reg Out;
always @(*)begincase(S)
1'b0:Out=I0;
1'b1:Out=I1;
endcase
end
endmodule

module Mux2to1_TB();
reg I0,I1,S;
wire Out;
Mux2to1 MX(.I0(I0),.I1(I1),.S(S),.Out(Out));
initial begin
S=0; I0=1; I1=0; #10;
S=1; I0=0; I1=1; #10;
end endmodule
```

Output

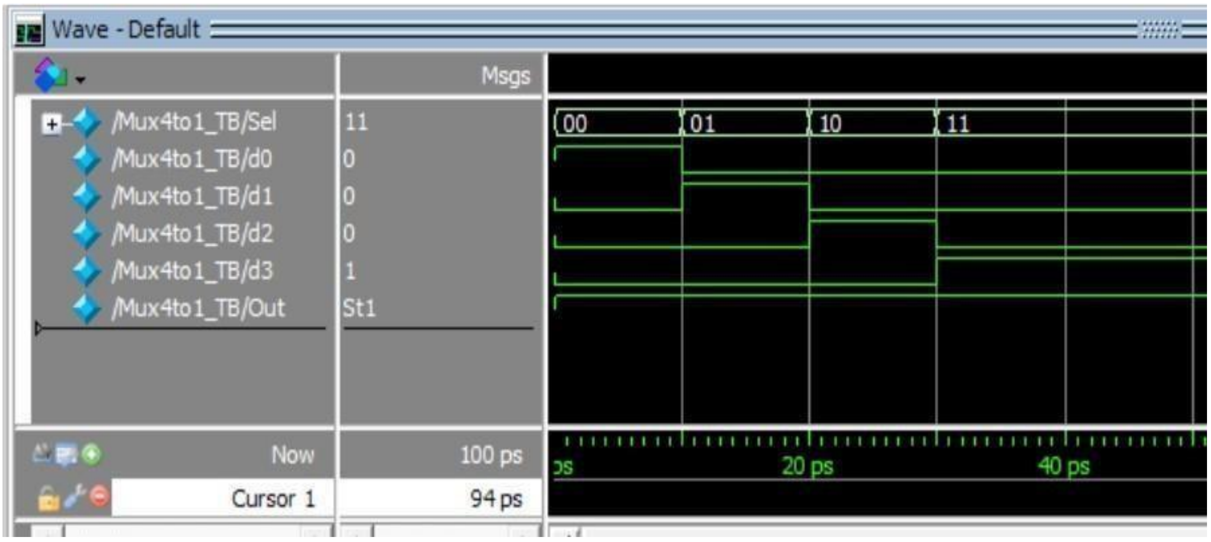


//4:1 muultiplexer

```
module Mux4to1(d0,d1,d2,d3,Sel,Out);
output reg Out;
input [1:0]Sel;
 input d0,d1,d2,d3;
always @(Sel) begin
case(Sel) 2'b00:Out=d0;
2'b01:Out=d1;
2'b10:Out=d2;
2'b11:Out=d3;
endcase
end
endmodule

module Mux4to1_TB();
reg [1:0]Sel;
reg d0; reg d1;
reg d2; reg d3;
wire Out;
Mux4to1 MUX(.d0(d0),.d1(d1),.d2(d2),.d3(d3),.Sel(Sel),.Out(Out));
 initial begin
d0=1; d1=0; d2=0; d3=0; Sel=0; #10;
d0=0; d1=1; d2=0; d3=0; Sel=1; #10;
d0=0; d1=0; d2=1; d3=0; Sel=2; #10;
d0=0; d1=0; d2=0; d3=1; Sel=3; #10;
end endmodule
```

Output



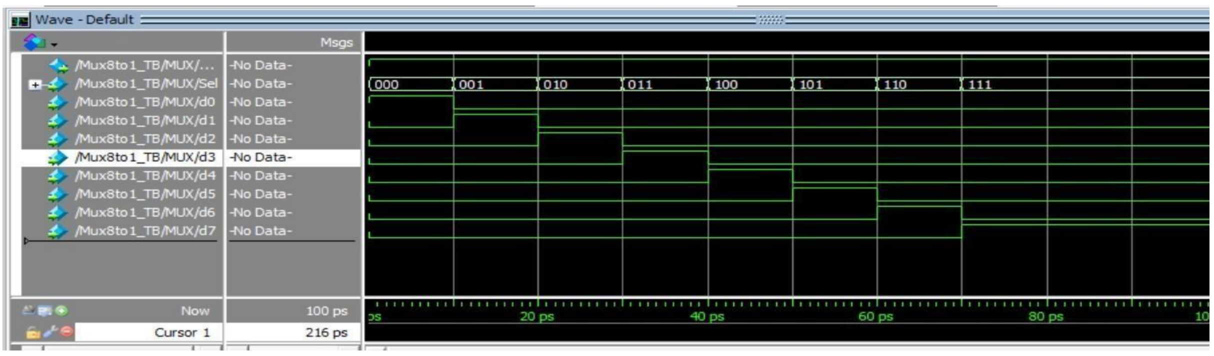
```
Wave - Default
                              Msgs
/Mux4to1_TB/Sel               11
/Mux4to1_TB/d0                0
/Mux4to1_TB/d1                0
/Mux4to1_TB/d2                0
/Mux4to1_TB/d3                1
/Mux4to1_TB/Out               St1
Now                           100 ps
Cursor 1                      94 ps
```

//8:1 Multiplexer

```
module Mux8to1(d0,d1,d2,d3,d4,d5,d6,d7,Sel,Out);
output reg Out;
input [2:0]Sel;
input d0,d1,d2,d3,d4,d5,d6,d7;
always @(Sel)
begin case(Sel)
3'b000:Out=d0;
3'b001:Out=d1;
3'b010:Out=d2;
3'b011:Out=d3;
3'b100:Out=d4;
3'b101:Out=d5;
3'b110:Out=d6;
3'b111:Out=d7;
endcase
end endmodule

module Mux8to1_TB();reg
[2:0]Sel;
reg d0; reg d1;
reg d2; reg d3;
reg d4; reg d5;
reg d6; reg d7;
wire Out;
Mux8to1
MUX(.d0(d0),.d1(d1),.d2(d2),.d3(d3),.d4(d4),.d5(d5),.d6(d6),.d7(d7),.Sel(Sel),.Out(Out));
initial begin
d0=1; d1=0; d2=0; d3=0; d4=0; d5=0; d6=0; d7=0; Sel=0; #10; d0=0;
d1=1; d2=0; d3=0; d4=0; d5=0; d6=0; d7=0; Sel=1; #10; d0=0; d1=0;
d2=1; d3=0; d4=0; d5=0; d6=0; d7=0; Sel=2; #10; d0=0; d1=0; d2=0;
d3=1; d4=0; d5=0; d6=0; d7=0; Sel=3; #10; d0=0; d1=0; d2=0; d3=0;
d4=1; d5=0; d6=0; d7=0; Sel=4; #10; d0=0; d1=0; d2=0; d3=0; d4=0;
d5=1; d6=0; d7=0; Sel=5; #10; d0=0; d1=0; d2=0; d3=0; d4=0; d5=0;
d6=1; d7=0; Sel=6; #10; d0=0; d1=0; d2=0; d3=0; d4=0; d5=0; d6=0;
d7=1; Sel=7; #10;
end endmodule
```

Output



Possible viva questions:

1. What is the purpose of a multiplexer?
Ans: A multiplexer is a digital circuit that selects one of several input signals and directs it to a single output.

2. How does a 2:1 multiplexer differ from a 4:1 multiplexer?
Ans: A 2:1 multiplexer has two inputs, and a 4:1 multiplexer has four inputs.

3. Explain the role of the select input in a multiplexer.
Ans: A 2:1 multiplexer has two inputs, and a 4:1 multiplexer has four inputs.

4. What happens when the select input is 0 in a 2:1 multiplexer?
Ans: When the select input is 0 in a 2:1 multiplexer, the output is connected to the first input.

5. How is the output determined in an 8:1 multiplexer?
Ans: The select input in an 8:1 multiplexer specifies which of the eight inputs is routed to the output.

6. Can a multiplexer be used to implement logic gates? How?
Ans: Yes, multiplexers can be configured to function as logic gates by appropriately selecting inputs.

7. Can Verilog code for a 2:1 multiplexer be reused for a 4:1 multiplexer? Why or why not?
Ans: Some code can be reused, but adjustments are needed for the increased number of inputs.

8. How does the complexity of a multiplexer circuit increase with the number of inputs?
Ans: The complexity of a multiplexer circuit increases linearly with the number of inputs, leading to more logic gates and connections.

9. Explain the concept of data routing in multiplexers.
Ans: Data routing refers to selecting and transmitting a specific input to the output based on the select input.

10. What is the difference between a multiplexer and a demultiplexer?
Ans: A multiplexer selects one input among many for output, while a demultiplexer routes one input to multiple outputs.

EXPERIMENT NO – 07

Design Verilog program to implement types of De-Multiplexer.

Aim: To Design Verilog program to implement Different types of De-multiplexer like 1:2, 1:4 and 1:8.

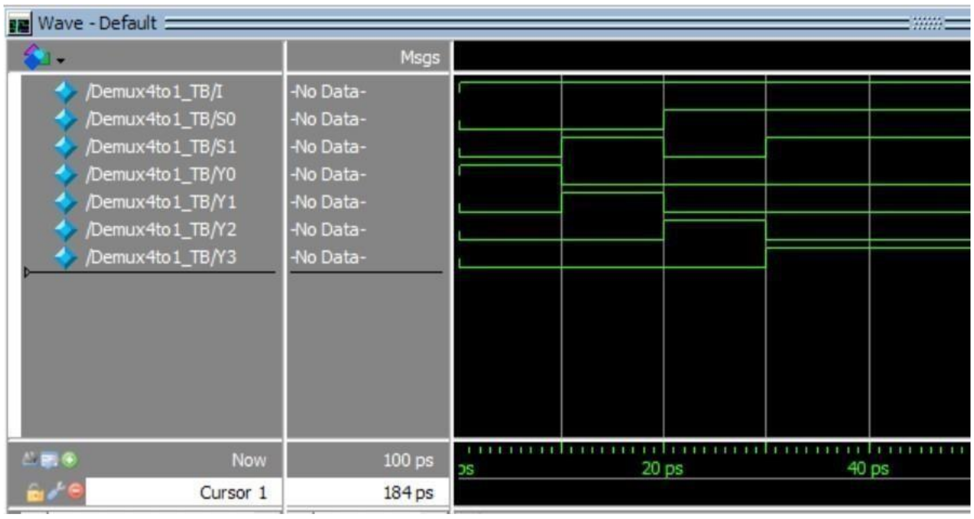
Objectives:
1. Implement Verilog program for 1:2, 1:4, 1:8 Demultiplexer designs.
2. Verify functionality through simulation for accuracy and performance assessment.

//4:1 Demultiplexer

```
module Demux4to1(I,S0,S1,Y0,Y1,Y2,Y3);
input I,S0,S1;
output Y0,Y1,Y2,Y3; assign
Y0=(~S0)&(~S1)&I;assign
Y1=(~S0)&(S1)&I; assign
Y2=(S0)&(~S1)&I; assign
Y3=(S0)&(S1)&I;
endmodule

module Demux4to1_TB();
reg I,S0,S1;
wire Y0,Y1,Y2,Y3;
Demux4to1 DEMUX(I,S0,S1,Y0,Y1,Y2,Y3);
initial begin
S0=0;S1=0;I=1;#10;
S0=0;S1=1;I=1;#10;
S0=1;S1=0;I=1;#10;
S0=1;S1=1;I=1;#10;
end
endmodule
```

Output

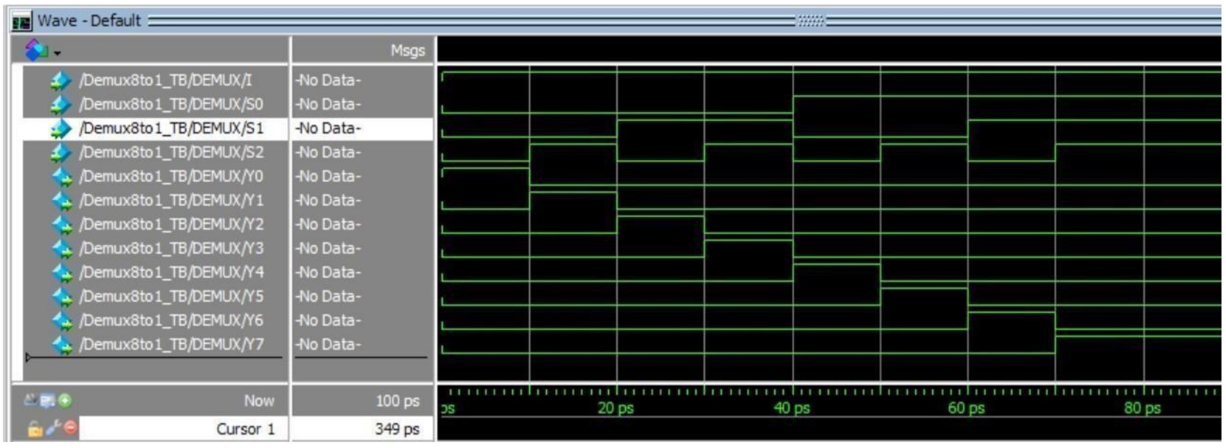

Wave - Default
Msgs
/Demux4to1_TB/I -No Data-
/Demux4to1_TB/S0 -No Data-
/Demux4to1_TB/S1 -No Data-
/Demux4to1_TB/Y0 -No Data-
/Demux4to1_TB/Y1 -No Data-
/Demux4to1_TB/Y2 -No Data-
/Demux4to1_TB/Y3 -No Data-
Now 100 ps
Cursor 1 184 ps
20 ps
40 ps

//8:1 Demultiplexer

```
module   Demux8to1(I,S0,S1,S2,Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7);
input I,S0,S1,S2;
output  Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7;
assign  Y0=(~S0)&(~S1)&(~S2)&I;assign
Y1=(~S0)&(~S1)&(S2)&I; assign
Y2=(~S0)&(S1)&(~S2)&I; assign
Y3=(~S0)&(S1)&(S2)&I; assign
Y4=(S0)&(~S1)&(~S2)&I; assign
Y5=(S0)&(~S1)&(S2)&I; assign
Y6=(S0)&(S1)&(~S2)&I; assign
Y7=(S0)&(S1)&(S2)&I; endmodule

module  Demux8to1_TB();reg
I,S0,S1,S2;
wire  Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7;
Demux8to1   DEMUX(I,S0,S1,S2,Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7);
initial begin S0=0;S1=0;S2=0;I=1;#10;
S0=0;S1=0;S2=1;I=1;#10;
S0=0;S1=1;S2=0;I=1;#10;
S0=0;S1=1;S2=1;I=1;#10;
S0=1;S1=0;S2=0;I=1;#10;
S0=1;S1=0;S2=1;I=1;#10;
S0=1;S1=1;S2=0;I=1;#10;
S0=1;S1=1;S2=1;I=1;#10;
end endmodule
```

Output


Wave - Default
Msgs
/Demux8to1_TB/DEMUX/I -No Data-
/Demux8to1_TB/DEMUX/S0 -No Data-
/Demux8to1_TB/DEMUX/S1 -No Data-
/Demux8to1_TB/DEMUX/S2 -No Data-
/Demux8to1_TB/DEMUX/Y0 -No Data-
/Demux8to1_TB/DEMUX/Y1 -No Data-
/Demux8to1_TB/DEMUX/Y2 -No Data-
/Demux8to1_TB/DEMUX/Y3 -No Data-
/Demux8to1_TB/DEMUX/Y4 -No Data-
/Demux8to1_TB/DEMUX/Y5 -No Data-
/Demux8to1_TB/DEMUX/Y6 -No Data-
/Demux8to1_TB/DEMUX/Y7 -No Data-
Now 100 ps
Cursor 1 349 ps
ps
20 ps
40 ps
60 ps
80 ps

Possible viva questions:

1. **What is the primary function of a De-Multiplexer (De-Mux)?**

Ans: A De-Mux takes a single input and directs it to one of several possible outputs.

2. **Explain the operation of a 1-to-2 De-Multiplexer.**

Ans: A 1-to-2 De-Mux selects between two output lines based on the control input.

3. **How does a 1-to-4 De-Multiplexer differ from a 1-to-2 De-Multiplexer?**

Ans: A 1-to-4 De-Mux directs one input to one of four outputs based on the control input.

4. **What is the significance of the Enable input in a De-Multiplexer?**

Ans: The Enable input allows or inhibits the De-Mux operation, controlling the output based on its state.

5. **Discuss the concept of a Binary De-Multiplexer.**

Ans: A Binary De-Mux with n control lines can select one of 2^n output lines for the input.

6. **How does a De-Multiplexer with multiple outputs, like 1-to-8, function?**

Ans: A 1-to-8 De-Mux routes a single input to one of eight possible output lines determined by the control inputs.

7. **What is the purpose of a Priority De-Multiplexer?**

Ans: A Priority De-Mux ensures that a specific input is directed to its corresponding output in the presence of multiple inputs.

8. **In what scenarios is a Decoder considered a specialized De-Multiplexer?**

Ans: A Decoder acts as a De-Mux by converting a binary code into an equivalent decimal or other code.

9. **How do De-Multiplexers contribute to data routing in communication systems?**

Ans: De-Muxes route data to specific channels, aiding in the distribution of information in communication networks.

10. **Explain how De-Multiplexers enhance flexibility in digital circuits.**

Ans: De-Muxes provide flexibility by enabling the selection of different output lines for a single input based on control signals.

EXPERIMENT NO – 08

Design Verilog program for implementing various types of Flip-Flops such as SR,JK and D

Aim: To design and simulate Verilog modules for three different types of flip-flops (SR, JK, and D)using behavioral modeling. The program should demonstrate the functionality and behavior of these flip-flops under various input conditions.

Objectives:

- Implement Verilog modules for SR, JK, and D flip-flops using behavioral modeling. Developa testbench to simulate the behavior of the implemented flip-flops.
- Apply different input conditions to the flip-flops in the testbench to observe their responses.
- Verify that the flip-flops exhibit the expected sequential logic behavior, such as state changesand toggling.

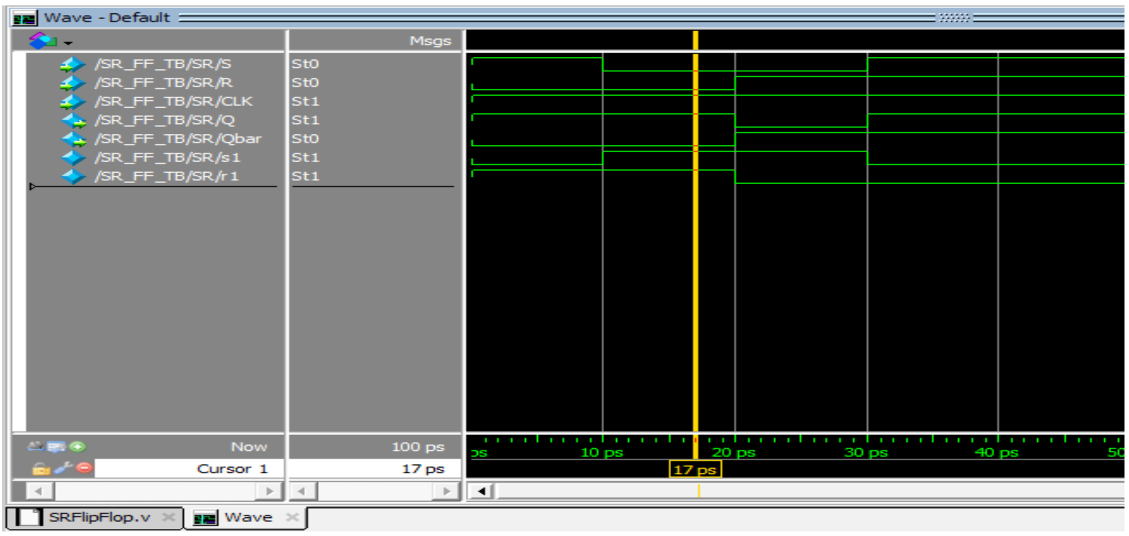
//SR FLIP FLOP

```
module
SR_FF(S,R,CLK,Q,Qbar);
input S,R,CLK;
output  Q,Qbar;
wire s1,r1;
nand n1(s1,S,CLK);
nand n2(r1,R,CLK);
nand n3(Q,s1,Qbar);
nand n4(Qbar,r1,Q);
endmodule

module
SR_FF_TB();
reg S,R,CLK;
wire Q;
wire Qbar;
SR_FF SR(S,R,CLK,Q,Qbar);
initial begin
$monitor("S=%b,R=%b,CLK=%b,Q=%b,Qbar=%b",S,R,CLK,Q
,Qbar);  S=1;R=0;CLK=1;#10;
```

```
 S=0;R=0;CLK=1; #10;
S=0;R=1;CLK=1; #10;
 S=1;R=1;CLK=1;
end
endmodule
```

Output



S=1,R=0,CLK=1,Q=1,Qbar=0#

S=0,R=0,CLK=1,Q=1,Qbar=0 #

S=0,R=1,CLK=1,Q=0,Qbar=1 #

S=1,R=1,CLK=1,Q=1,Qbar=1

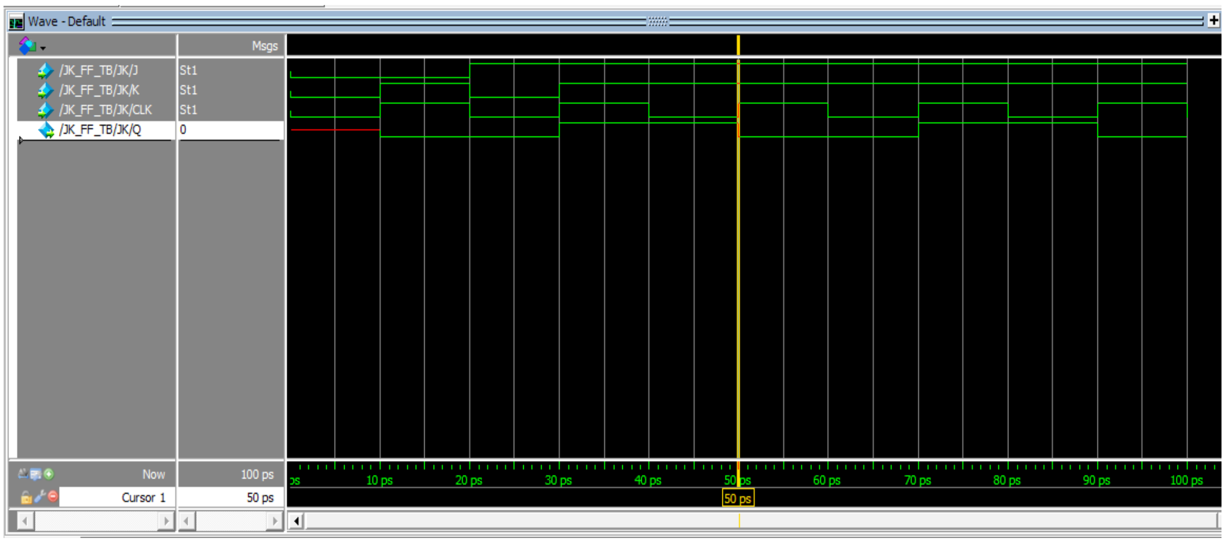
//JK FLIP FLOP

```
module
JK_FF(J,K,Q,CLK);
input J,K,CLK;
output reg Q;
always @(posedge CLK)case({J,K}) 2'b00:Q<=Q;
2'b01:Q<=0;
2'b10:Q<=1;
2'b11:Q<=~Q;
endcase
endmodule
```

```
        module JK_FF_TB();
        reg J,K,CLK;
        wire Q;
        JK_FF  JK(J,K,Q,CLK);
        initial beginCLK=0;
        forever #10 CLK=~CLK;
        end
        initial begin

        J=0;
        K=0;
        #10;
        J=0;
        K=1;
        #10;
        J=1;
        K=0;
        #10;
        J=1;
        K=1;
        #10;
end
endmodule
```

Output



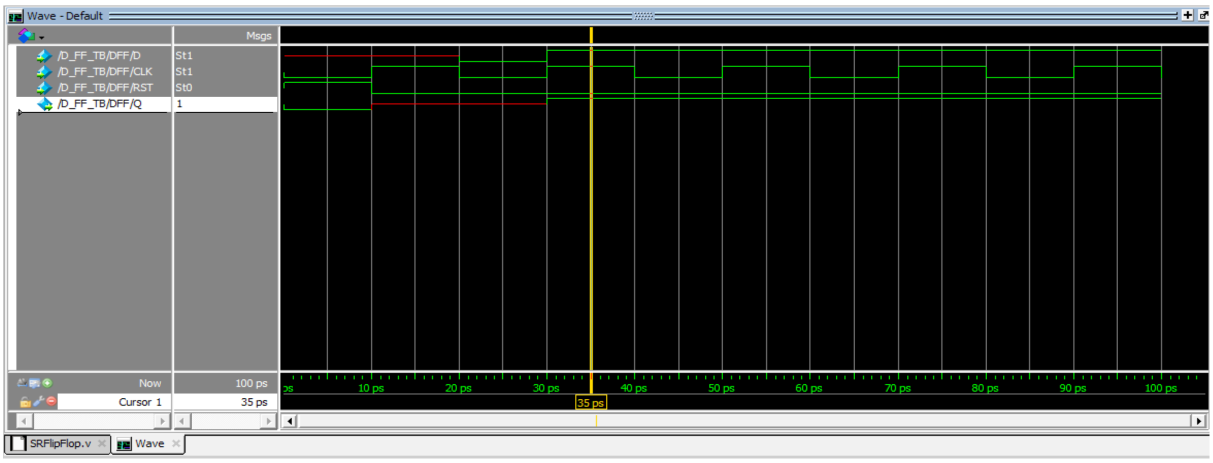
D Flip Flop

```
module  D_FF(Q,D,CLK,RST);
input D,CLK,RST;
output reg Q;
always @(posedge CLK or posedge
RST)begin
if(RST==1)
Q<=0;
else
Q<=D;
end
endmodule

module D_FF_TB();reg D,CLK,RST;
wire Q;
D_FF  DFF(Q,D,CLK,RST);

Initial begin
CLK=0;
forever
#10
CLK=~CLK;
end
initial beginRST=1; #10; RST=0; #10;
D=0;
#10;
D=1;
#10;
end
endmodule
```

Output



Possible viva questions:

1. **What is an SR flip-flop, and what are its primary characteristics?**
   An SR flip-flop is a digital circuit with Set (S) and Reset (R) inputs, storing binary information.

2. **What are the possible states of an SR flip-flop?**
   Possible SR flip-flop states: Set (Q=1), Reset (Q=0), Hold (Q unchanged).

3. **How does the presence of an illegal state in an SR flip-flop affect its operation?**
   Illegal states in SR flip-flop may lead to undefined behavior or undesired output transitions.

4. **Describe a real-world application where an SR flip-flop is useful.**
   SR flip-flops are used in memory circuits, latch circuits, and control systems for state storage.

5. **How can you use an SR flip-flop to create a simple toggle circuit?**
   Connect S and R inputs of an SR flip-flop, toggling between Set and Reset states.

6. **Explain the concept of setup time and hold time in the context of an SR flip-flop.**
   Setup time ensures stable inputs before a clock edge, while hold time maintains stability after the edge.

7. **What are the features that distinguish a JK flip-flop from an SR flip-flop?**
   JK flip-flop features include a toggle functionality, preventing illegal states, and aversatile behavior distinguishing it from an SR flip-flop.

8. **How is a JK flip-flop used to create a toggle circuit?**
   Connecting J and K inputs allows a JK flip-flop to function as a toggle circuit, changing state on each clock edge.
9. **In what applications is a JK flip-flop preferred over other types of flip-flops?**
   JK flip-flops are preferred in applications requiring toggling behavior, memory storage, and sequential logic due to their versatility and prevention of illegal states.

10. **In what applications is a D flip-flop typically used for data storage and transfer?**
    D flip-flops are used in applications like registers, memory units, and sequential logic circuits for reliable data storage and transfer