VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELGAVI-590018, KARNATAK

A J INSTITUTE OF ENGINEERING & TECHNOLOGY

(A unit of Laxmi Memorial Education Trust. (R))

NH - 66, Kottara Chowki, Kodical Cross - 575 006

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(Accredited by NBA)

MASTER MANUAL

Course: Digital Design and Computer Organization

Course Code: BCS302

III-SEMESTER

Prepared by:

Dr. Laxmi Gulappagol

Associate Professor,

Department of Computer Science & Engineering, AJIET, Mangaluru

ACADEMIC YEAR: 2023-24

VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI

B.E. in Computer Science and Engineering

Scheme of Teaching and Examinations 2022

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

(Effective from the academic year 2022 - 23)

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

"To produce top-quality engineers who are groomed for attaining excellence in their profession and competitive enough to help in the growth of nation and global society."

MISSION OF THE INSTITUTE

M1: To offer affordable high-quality graduate program in engineering with value education and make the students socially responsible.

M2: To support and enhance the institutional environment to attain research excellence in both faculty and students and to inspire them to push the boundaries of knowledge base.

M3: To identify the common areas of interest amongst the individuals for the effective industry-institute partnership in a sustainable way by systematically working together.

M4: To promote the entrepreneurial attitude and inculcate innovative ideas among the engineering professionals.

VISION OF THE DEPARTMENT

"To adapt the evolutionary changes in computer science and expose the students to the cutting-edge technologies to produce globally competent professionals."

MISSION OF THE DEPARTMENT

M1: To mould the students by providing quality computer education and by strengthening the Industry -Academic interface.

M2: To impart the professional skills, innovative research activities and entrepreneurial capabilities in students.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

After 4 years of graduation, graduates will be able to

PEO1: To develop in students, the ability to solve real life problems by applying fundamental science and elementary strengths of computer science courses.

PEO2: To mould students, to have a successful career in the IT industry where graduates will be able to design and implement the needs of society and nation.

PEO3: To transform students, to excel in a competitive world through higher education and indulge in research through continuous learning process.

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.

PROGRAM SPECIFIC OUTCOMES (PSOs)

At the end of the program, graduates will be able to

PSO1: Apply engineering principles, professional ethics and fundamental science in designing systems and communication models (protocols).

PSO2: Design and develop smart and intelligent based applications in computational environment.

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, anderaser 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 thatsame 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 rightside 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:

Textbooks

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**
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: Observation (10 Marks)	**Excellent (10-8)**	**Good (7-6)**	**Average (5-4)**	**Poor (3-0)**
	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: Record writing : Write up of Procedure / Algorithm/ Program and Execution/conduction of experiment **(30 Marks)**	**Excellent (30-26)**	**Good (25-20)**	**Average (19-15)**	**Poor (14-0)**
	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: 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
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: Write-Up (10 Marks)	**Excellent (10-8)**	**Good (7-6)**	**Average (5-4)**	**Poor (3-0)**
	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

Name of the Laboratory: COMPUTING LABORATORY-1

Sl. No.	Name of the Equipment	Specialization	Quantity
1	Desktop	Intel(R) Core (TM) i3-6100 CPU @ 3.70GHz, 4.00 GB RAM, 64-bit operating system, x64-based processor, ACER-Monitor, Keyboard and Mouse.	35
2	GSAS LPC 2148 ARM 7 Evaluation Board with Stepper Motor and DC Motor	LPC2148 ARM7 TDMI micro controller with 512k on-chip memory	15
3	UPS	20KVA	1
4	Switches	24 Port Gigabytes	1
5	Internet	100mbps	1
6	Projector	EPSON Projector with HDMI Port	1

Room Number	:	**A-324**
Total Area of the laboratory	:	130 Sq. Meters
Total Amount Spent	:	Rs. 13, 96, 513 /-
Name of the HOD	:	Dr. Antony P. J
Name of the lab in charge	:	Mr. Vijaykumar Dudhanikar
Name of the lab instructor	:	Ms. Meghna Kotiyan

LIST OF EXPERIMENT/PROGRAMS

Additional Experiment/Programs- Content beyond Syllabus

1. Verification of the truth tables of TTL 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.

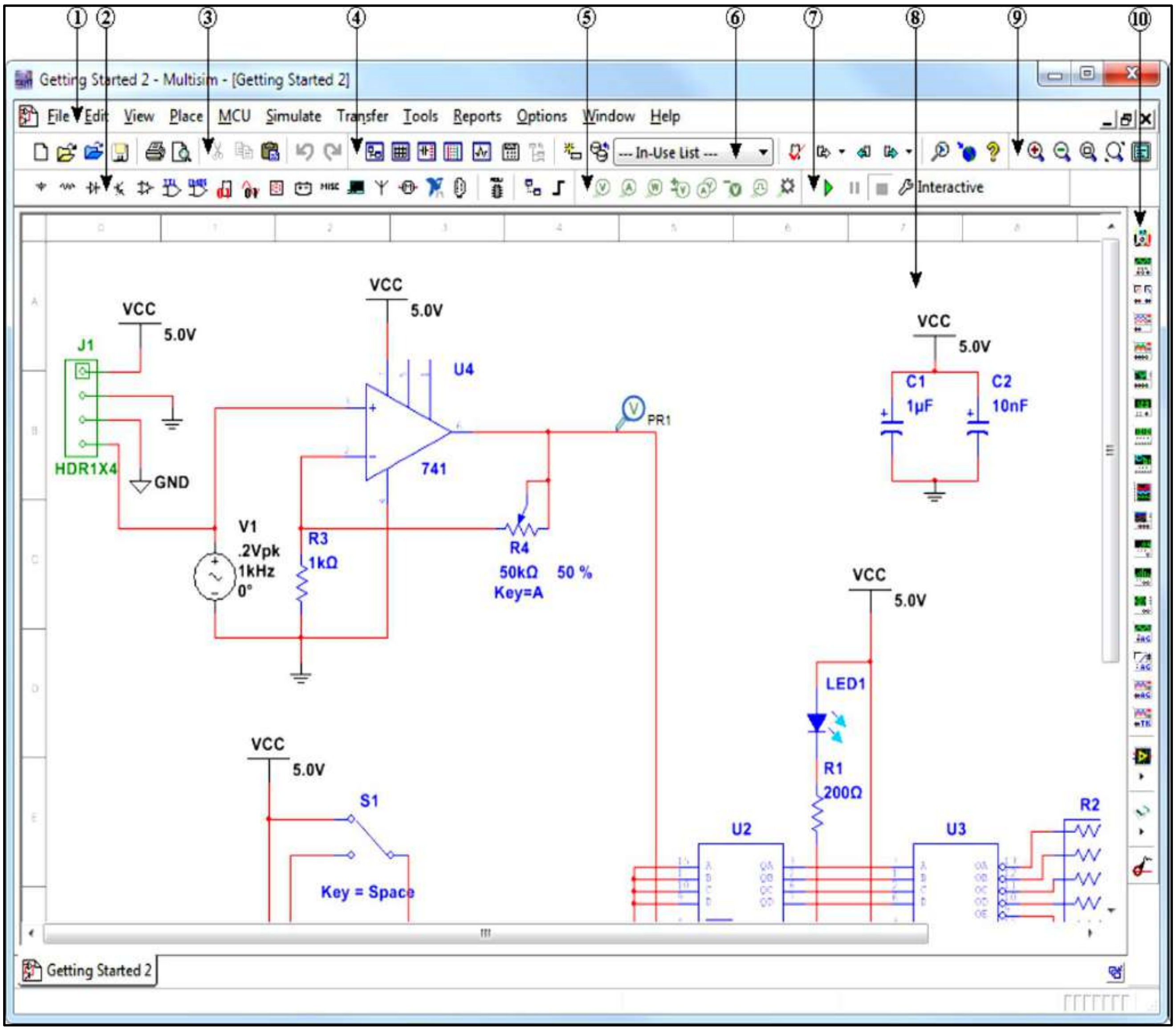
INTRODUCTION

❖ Introduction to the Multisim Interface

Multisim is the schematic capture and simulation application of National Instruments Circuit Design Suite, a suite of EDA (Electronics Design Automation) tools that assists you in carrying out the major steps in the circuit design flow. It provides a virtual environment for designing, testing, and simulating electronic circuits. Multisim allows users to create schematic diagrams, simulate circuit behavior, and analyze circuit performance. Multisim is designed for schematic entry, simulation, and feeding to downstage steps, such as PCB layout.

Multisim User Interface

The Multisim user interface includes the following elements:


①
②
③
④
⑤
⑥
⑦
⑧
⑨
⑩
Getting Started 2 - Multisim - [Getting Started 2]
File Edit View Place MCU Simulate Transfer Tools Reports Options Window Help
--- In-Use List ---
Interactive
Getting Started 2

Refer to the table below as needed:

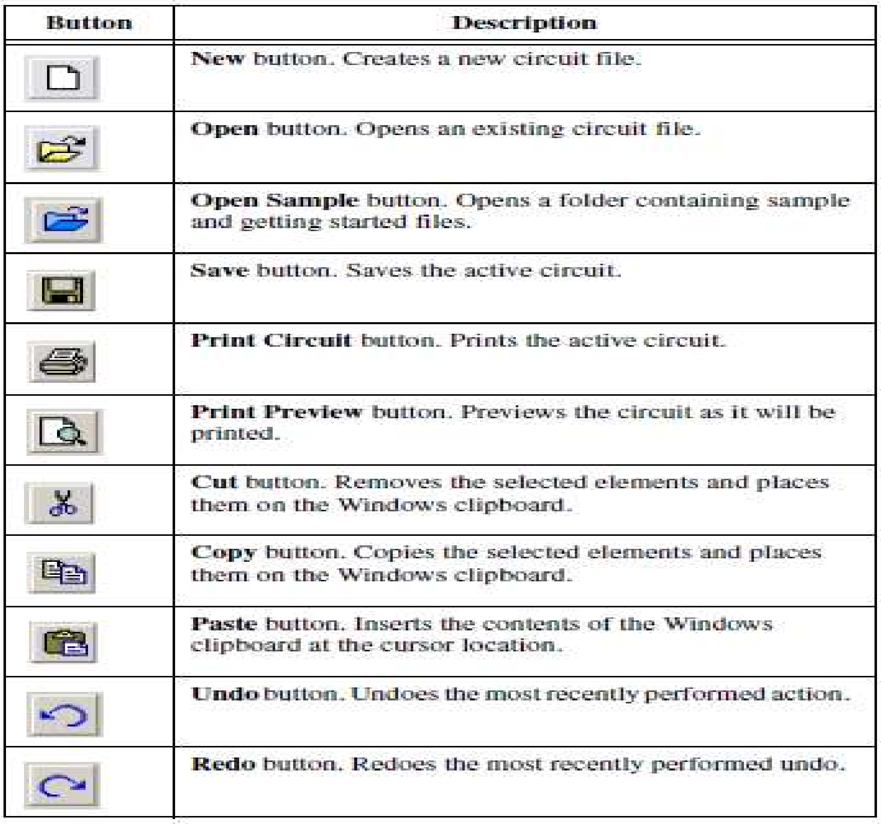
	Element	**Description**
1	Menu Bar	Contains the commands for all functions.
2	Component Toolbar	Contains buttons that you use to select components from the Multisim database for placement in your schematic.
3	Standard Toolbar	Contains buttons for commonly-performed functions such as Save, Print, Cut, and Paste.
4	Main Toolbar	Contains buttons for common Multisim functions.
5	Place Probe Toolbar	Contains buttons that you use to place various types of probes on the design. You can also access **Probe Settings** from here.
6	In-Use Toolbar	Contains a list of all components used in the design.
7	Simulation Toolbar	Contains buttons for starting, stopping and pausing simulation.
8	Workspace	This is where you build your designs.
9	View Toolbar	Contains buttons for modifying the way the screen is displayed.
10	Instruments Toolbar	Contains buttons for each instrument.

Installation steps are given below: Install Multisim by obtaining the software from National Instruments and following the provided installation instructions.

1. Launch Multisim and choose to **create a new project** or open an existing one from the start page.
2. Create a **new project** by giving it a name and choosing a save location.
3. Within the project, create a schematic by clicking **"New"** and selecting **"Schematic."**
4. Build your circuit on the schematic canvas by dragging components from the library and connecting them with wiring tools.
5. Optionally, add virtual instruments like oscilloscopes or multimeters to visualize and analyze your circuit during simulation.
6. Simulate your circuit by clicking on the **"Simulate" tab**, choosing the simulation type, adjusting settings, and **clicking "Run."**
7. Analyze simulation results using built-in instruments, graphs, and charts to understand circuit behavior.
8. Optionally, transfer your schematic to NI Ultiboard for PCB layout by clicking "Transfer to Ultiboard."
9. Save your project periodically, export simulation results or other data as needed, and close Multisim when finished.

Standard Toolbar

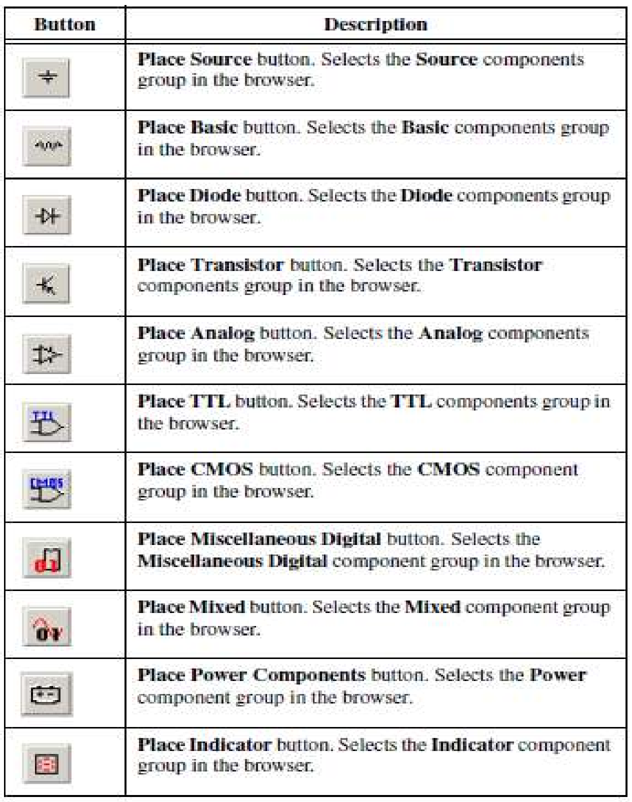
The **Standard** toolbar contains buttons for commonly performed functions. Its buttons are described below:



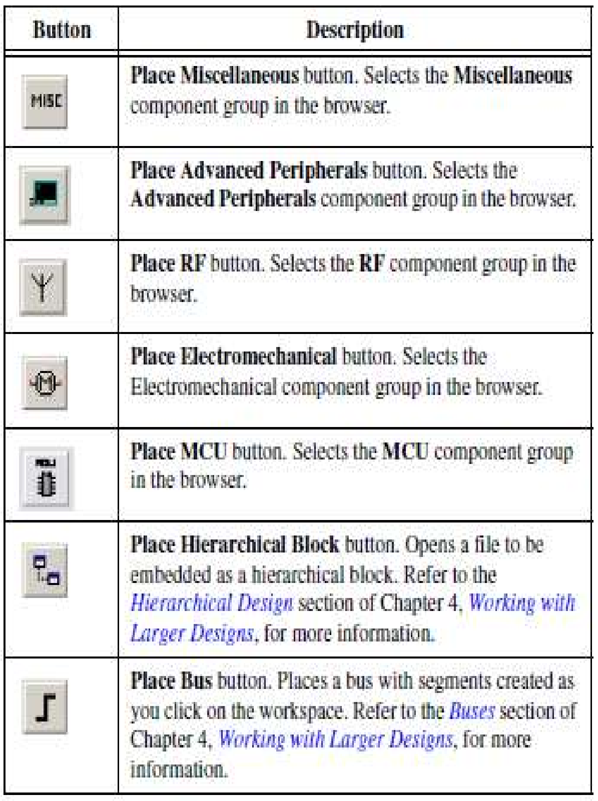
Button	Description
	New button. Creates a new circuit file.
	Open button. Opens an existing circuit file.
	Open Sample button. Opens a folder containing sample and getting started files.
	Save button. Saves the active circuit.
	Print Circuit button. Prints the active circuit.
	Print Preview button. Previews the circuit as it will be printed.
	Cut button. Removes the selected elements and places them on the Windows clipboard.
	Copy button. Copies the selected elements and places them on the Windows clipboard.
	Paste button. Inserts the contents of the Windows clipboard at the cursor location.
	Undo button. Undoes the most recently performed action.
	Redo button. Redoes the most recently performed undo.

Components Toolbar

The buttons in the Components toolbar are described below. Each button will launch the place component browser (Select a Component browser) with the group specified on the button pre-selected.



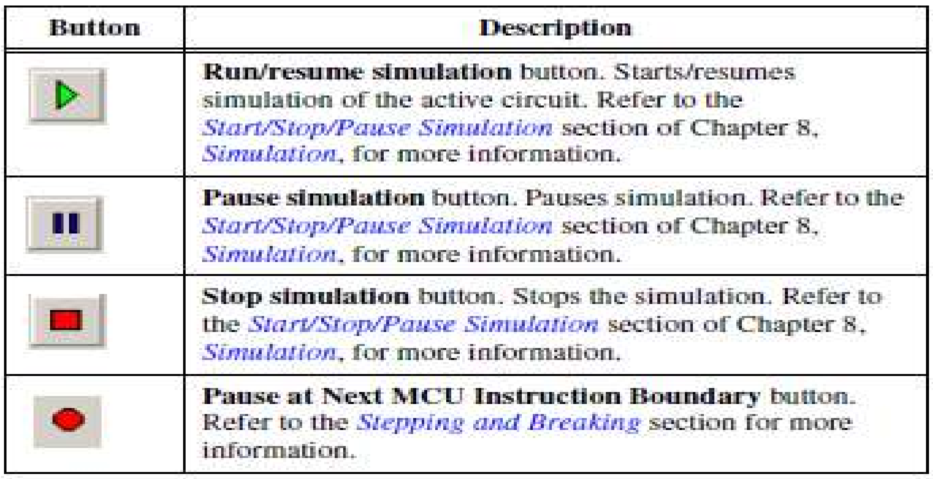
Button	Description
	Place Source button. Selects the **Source** components group in the browser.
	Place Basic button. Selects the **Basic** components group in the browser.
	Place Diode button. Selects the **Diode** components group in the browser.
	Place Transistor button. Selects the **Transistor** components group in the browser.
	Place Analog button. Selects the **Analog** components group in the browser.
TTL	**Place TTL** button. Selects the **TTL** components group in the browser.
CMOS	**Place CMOS** button. Selects the **CMOS** component group in the browser.
	Place Miscellaneous Digital button. Selects the **Miscellaneous Digital** component group in the browser.
	Place Mixed button. Selects the **Mixed** component group in the browser.
	Place Power Components button. Selects the **Power** component group in the browser.
	Place Indicator button. Selects the **Indicator** component group in the browser.



Button	Description
MISC	**Place Miscellaneous** button. Selects the **Miscellaneous** component group in the browser.
	Place Advanced Peripherals button. Selects the **Advanced Peripherals** component group in the browser.
	Place RF button. Selects the **RF** component group in the browser.
	Place Electromechanical button. Selects the Electromechanical component group in the browser.
	Place MCU button. Selects the **MCU** component group in the browser.
	Place Hierarchical Block button. Opens a file to be embedded as a hierarchical block. Refer to the *Hierarchical Design* section of Chapter 4, *Working with Larger Designs*, for more information.
	Place Bus button. Places a bus with segments created as you click on the workspace. Refer to the *Buses* section of Chapter 4, *Working with Larger Designs*, for more information.

Simulation Toolbar

The **Simulation** toolbar contains buttons used during simulation.



Button	Description
▶	**Run/resume simulation** button. Starts/resumes simulation of the active circuit. Refer to the *Start/Stop/Pause Simulation* section of Chapter 8, *Simulation*, for more information.
❚❚	**Pause simulation** button. Pauses simulation. Refer to the *Start/Stop/Pause Simulation* section of Chapter 8, *Simulation*, for more information.
■	**Stop simulation** button. Stops the simulation. Refer to the *Start/Stop/Pause Simulation* section of Chapter 8, *Simulation*, for more information.
●	**Pause at Next MCU Instruction Boundary** button. Refer to the *Stepping and Breaking* section for more information.

<u>OVERVIEW OF HDL</u>

Hardware Description Languages (HDLs) are specialized programming languages used for the design, modeling, and simulation of digital circuits and systems. HDLs allow engineers to describe the behavior and structure of electronic circuits at various levels of abstraction. Here is an overview of HDLs:

Purpose of HDLs:

1. **Digital Circuit Design:** HDLs are primarily used for designing digital circuits, including processors, memory units, controllers, and other digital components.
2. **Simulation:** HDLs facilitate the simulation of digital circuits before actual implementation, helping designers to verify functionality and detect potential issues.
3. **Synthesis:** HDLs support synthesis tools that convert high-level hardware descriptions into low-level representations suitable for implementation on programmable devices like FPGAs or ASICs.

Types of HDLs:

1. **Verilog:** Verilog is one of the most widely used HDLs. It supports both behavioral and structural modeling and is used for simulating and synthesizing digital circuits.
2. **VHDL (VHSIC Hardware Description Language):** VHDL is another popular HDL used for modeling digital circuits. It was developed by the U.S. Department of Defense and is known for its strong type-checking and suitability for complex systems.

Levels of Abstraction:

1. **Behavioral Modeling:** Describes the functionality of a circuit without specifying its structure. It includes high-level constructs like processes, tasks, and functions.
2. **Structural Modeling:** Describes a circuit in terms of its components and their interconnections.

It uses primitive logic gates, flip-flops, and instances of other modules.

3. **RTL (Register-Transfer Level):** Describes the flow of data between registers. It is often used for synthesis and represents a mid-level abstraction.
4. **Gate-Level Modeling:** Describes the circuit using logic gates and flip-flops. This level is close to the actual hardware implementation.

Applications:

1. **ASIC (Application-Specific Integrated Circuit) Design**: HDLs are commonly used in ASIC design to create custom integrated circuits tailored for specific applications.
2. **FPGA (Field-Programmable Gate Array) Design**: FPGAs can be programmed using HDLs to implement digital circuits on reconfigurable hardware.
3. **Digital Signal Processing:** HDLs are used in the design of digital signal processing circuits for applications such as communication systems and audio processing.

Verilog is a hardware description language (HDL) used for the modeling and design of digital systems. It is widely used in the field of electronic design automation (EDA) for both simulation and synthesis of digital circuits. Here are key aspects of Verilog:

1. Modules: Verilog designs are organized into modules, representing logical units of a digital circuit.

2. Ports: Modules have input and output ports, which define the signals entering and leaving the module.

3. Data Types: Verilog supports various data types including wire for connections, reg for registers, and integer for integers.

4. Behavioral Modeling: Describes the functionality of a circuit without specifying its structure using constructs like always blocks.

5. Structural Modeling: Describes a circuit in terms of its components and how they are interconnected using gate-level primitives and module instances.

6. Data Flow Modeling: Describes how data moves through a circuit based on the relationships between signals using continuous assignment statements.

7. Procedural Blocks: Verilog uses procedural blocks like always, initial, and assign for describing the behavior of a circuit at different levels of abstraction.

8. Sequential Logic: Supports modeling of flip-flops, latches, and other sequential logic elements for building state machines and memory elements.

9. Testbenches: Testbenches are written in Verilog to verify the correctness of a design through simulation.

10. Simulation: Verilog is commonly used for simulation using tools like ModelSim or VCS to validate and debug designs before implementation.

11. Synthesis: Verilog can be synthesized into netlists suitable for implementation on programmable devices like FPGAs or ASICs.

12. Programming Constructs: Verilog includes programming constructs like if, case, for, and while for control flow and conditional operations.

13. Tasks and Functions: Tasks and functions allow for modularization and code reuse within Verilog designs.

14. File Types: Verilog uses different file types, including. v for Verilog source files, .sv for System Verilog files, and others.

15. Hierarchy: Verilog supports hierarchical design where modules can be instantiated within other modules to create a modular and scalable design structure.

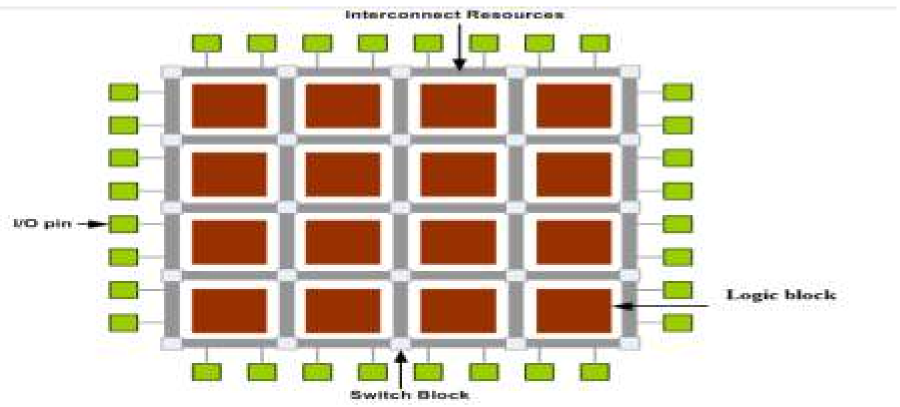
HDL Programming using Xilinx ISE design suite

Xilinx ISE means Xilinx® Integrated Software Environment (ISE), i.e programmable logic design tool in electronics industry. This Xilinx ® design software suite allows taking design from design entry through Xilinx device programming. The ISE Project Navigator manages and processes design through several steps in the ISE design flow. These steps are Design Entry, Synthesis, Implementation, Simulation/Verification, and Device Configuration. Xilinx is one of most popular software tool used to synthesize VHDL/Verilog code.

INTRODUCTION TO FPGA (FIELD PROGRAMMABLE GATE ARRAY)

FPGA contains a two-dimensional arrays of logic blocks and interconnections between logic blocks. Both the logic blocks and interconnects are programmable. Logic blocks are programmed to implement a desired function and the interconnects are programmed using the switch boxes to connect the logic blocks. To implement a complex design (CPU for instance), the design is divided into small sub functions and each sub function is implemented using one logic block. All the sub functions implemented in logic blocks must be connected and this is done by programming the interconnects.

INTERNAL STRUCTURE OF AN FPGA



FPGAs, alternative to the custom ICs, can be used to implement an entire System On one Chip (SOC). The main advantage of FPGA is ability to reprogram. User can reprogram an FPGA to implement a design and this is done after the FPGA is manufactured. This brings the name “Field Programmable.” Custom ICs are expensive and takes long time to design so they are useful when produced in bulk amounts. But FPGAs are easy to implement within a short time with the help of Computer Aided Designing (CAD) tools.

XILINX FPGA

Xilinx logic block consists of one Look Up Table (LUT) and one Flip-flop. An LUT is used to implement number of different functionality. The input lines to the logic block go into the LUT and enable it. The output of the LUT gives the result of the logic function that it implements and the output of logic block is registered or unregistered output from the LUT.

PROCEDURE:

The Procedure to be followed for Software and Hardware Programs are as follows:

Step 1: Go to Start Menu All Programs Xilinx ISE 13.1i and Select Project Navigator.

Step 2: Go to File Menu and select Close project to close previously opened project if any, and then Select New Project.

Step 3: Enter the Project name and location and Select the Top level module type as HDL.

Step 4: Select the Device family and Device name as Spartan3, pin density TQ144, -5 for FPGA.

Step 5: Right click on the source file and select new source followed by Verilog module and Give the file name same as the name of the entity.

Step 6: Define the ports used and their respective directions in the next window that opens.

Step 7: Write the architecture body and the generics etc. in the incomplete Verilog code that opens and save the file after completion of editing.

Step 8: Go to the Process view window and right click on the Synthesize - XST and Select Run. Correct the errors if any.

Step 9: Select and Right click the source file and click on the New Source tab and then select the Test Bench Waveform and give the appropriate file name for the same.

Step 10: Make the alterations in the Clock information and initial length of the test bench if needed.

Step 11: Set or Reset the inputs as required and save the test bench waveform file.

Step 12: Go to Process view and under Xilinx ISE Simulator Right click on the Simulate Behavioral model to see the output for the input conditions.

Step 13: Make the appropriate connections between the PC and the FPGA kit for the observation of outputs in the FPGA kit and for other Hardware Programming.

Step 14: Select and Right click the source file and click on the New Source tab and then select the Implementation Constraints file and give the appropriate file name for the same.

Step 15: Go to Process view and under User Constraints, double click on the Edit Constraints (Text).

Step 16: Write the code for the user constraints file as required and save the same.

Step 17: Select the main source file and right click on the Implement design in the process view window and select run.

Step 18: Right click on the Generate Programming file in the process view window and select run.

Step 19: Under the Generate Programming file tab, right click on the Configure device (Impact) and click on the Run option.

Step 20: Select the appropriate mode and make changes in the jumper settings of the FPGA Kit as required, select the appropriate. BIT extension file in the pop up window.

Step 21: Right click on the Chip picture in the pop up window and Select “Program”. Debug the errors if it is there. Set the conditions for the inputs using Dip switch and observe the outputs.

EXPERIMENT NO – 01

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

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

Objectives:

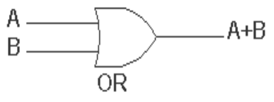
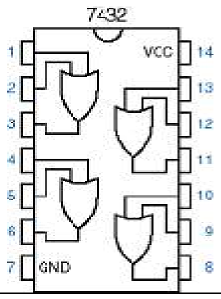
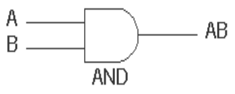
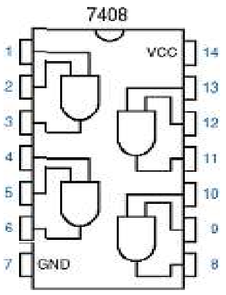
1. Simplify 4-variable logic expression for efficient design.
2. Simulate using basic gates for practical verification and optimization.

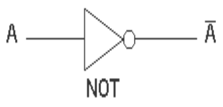
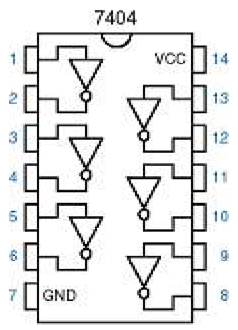
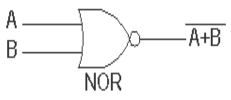
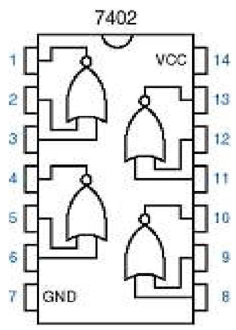
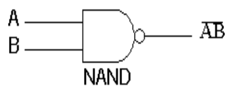
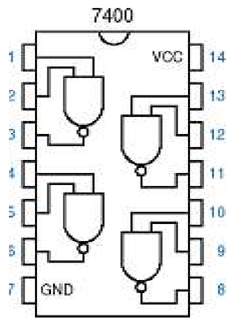
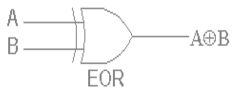
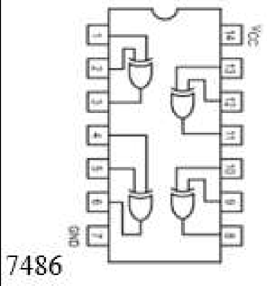
Theory:

SUM OF PRODUCT:

The short form of the sum of the product is SOP, and it is one kind of **Boolean algebra** expression. In this, the different product inputs are being added together. The product of inputs is Boolean logical AND whereas the sum or addition is Boolean logical OR. Before going to understand the concept of the sum of products

Expression of SOP F=AB+C(D+E)

Gate	Description	Truth Table			Logic Symbol	Pin Diagram
		A	B	Output		
OR	The output is active high if any one of the input is in active high state, Mathematically, Q = A+B	0 0 1 1	0 1 0 1	0 1 1 1		
AND	The output is active high only if both the inputs are in active high state, Mathematically,	0 0 1 1	0 1 0 1	0 0 0 1		

NOT	In this gate the output is opposite to the input state, Mathematically, Q = A	A 0 1		Output 1 0		
NOR	The output is active high only if both the inputs are in active low state, Mathematically, Q = (A+B)’	A 0 0 1 1	B 0 1 0 1	Output 1 0 0 0		
NAND	The output is active high only if any one of the input is in active low state, Mathematically, Q = (A.B)’	A 0 0 1 1	B 0 1 0 1	Output Q 1 1 1 0		
XOR	The output is active high only if any one of the input is in active high state, Mathematically, Q = A.B’ + B.A’	A 0 0 1 1	B 0 1 0 1	Output 0 1 1 0		

Equipment / components required:

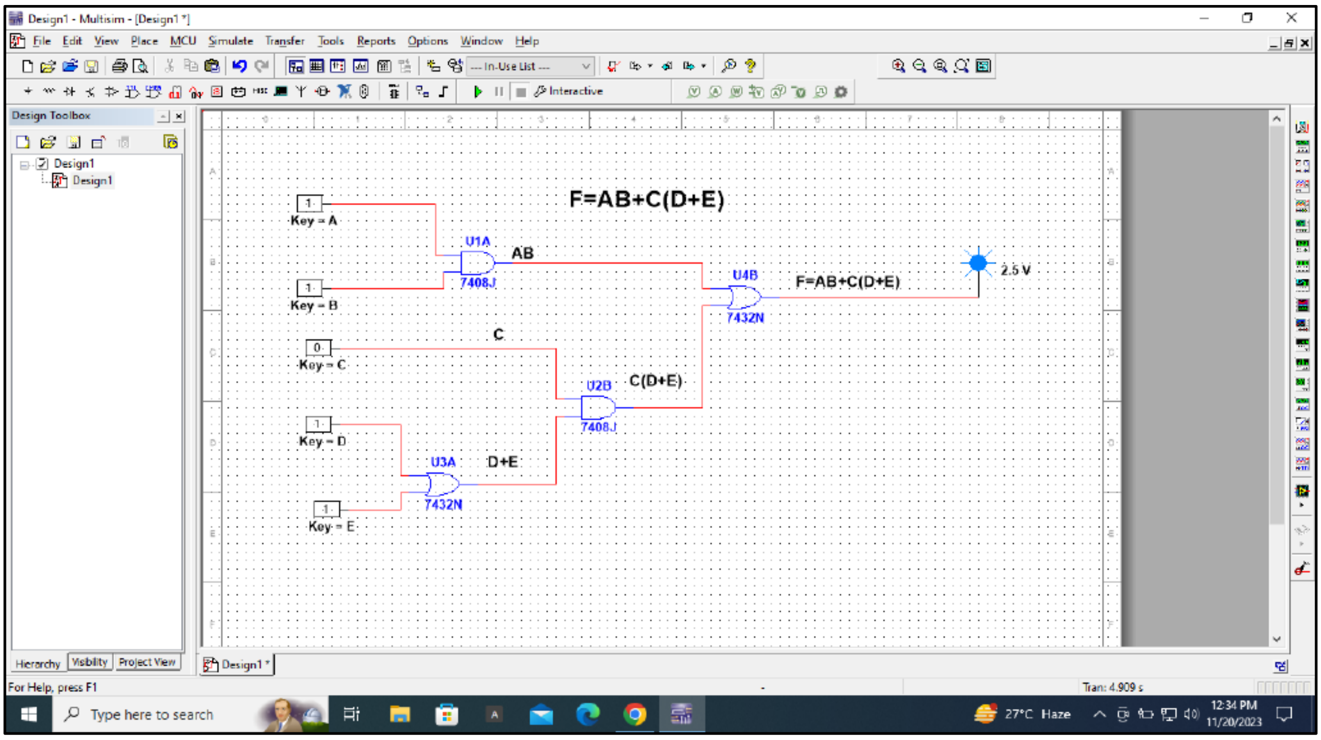
- Computer with Modelsim
- HP Computer i4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk
- Softwares: Modelsim - 5.7c

Procedure:

1. Open Multisim software on your computer.
2. Click on "File" and select "New" to create a new schematic.
3. Drag and drop components from the component toolbar onto the workspace.
4. Connect the components using wires.Specify values and properties for components.

5. Add a power supply to the circuit. Connect it appropriately (Vcc and GND).
6. Double-click on components to open their properties.
7. Set parameters, such as resistor values, capacitor values, etc.
8. Click on "Simulate" in the toolbar.
9. Set simulation parameters like start time, stop time, and time step.
10. Add instruments to measure and observe circuit behavior.
11. Click on the "Run" button (play icon) to start the simulation.
12. Observe and analyze waveforms and measurements from the instruments. Check for correct circuit behavior.
13. If unexpected results occur, review the circuit, component values, and connections. Make necessary adjustments and rerun the simulation.
14. Save your schematic and simulation results.
15. Document the circuit design, simulation parameters, and observed behavior.
16. Export simulation results or circuit diagrams for sharing or further analysis.

Simulation Output:



Results & conclusions:

The Simulate for Given a 4-variable logic expression, simplify it using appropriate technique using basic gates are verified.

Possible viva questions:

1. **What is Multisim?**

Answer: Multisim is a simulation software developed by National Instruments that allows users to design, simulate, and analyze electronic circuits. It provides a virtual environment for testing circuits before physical implementation
.

2. **How does Multisim aid in circuit design?**

   Answer: Multisim allows users to create and simulate electronic circuits virtually, enabling them to test the functionality and performance of the circuits without the need for physical components. This aids in the design and validation process.

3. **What types of components can you simulate in Multisim?**

   Answer: Multisim supports a wide range of electronic components, including resistors, capacitors, inductors, transistors, op-amps, microcontrollers, and more. Users can simulate both analog and digital components in their circuits.

4. **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.

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

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

6. **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.

7. **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'.

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

   Answer: AC + AD + BC + BD

9. **What is the significance 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.

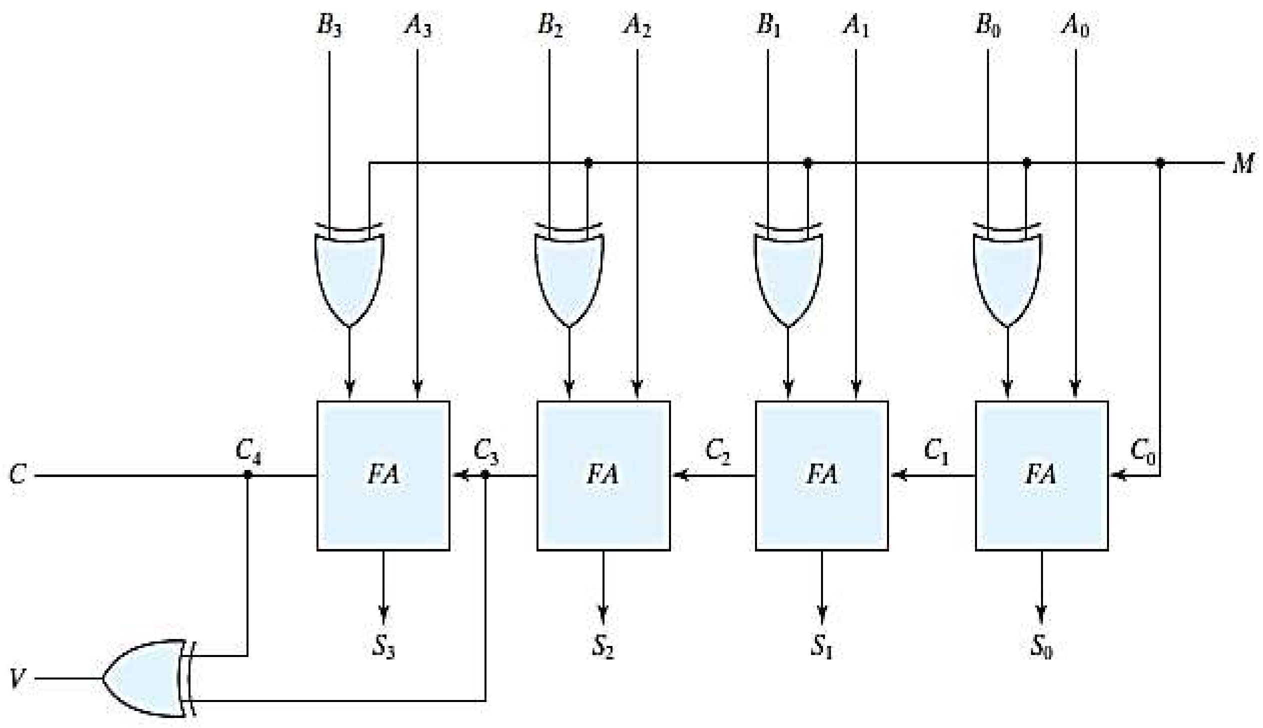
Aim: To design a 4 bit full adder and subtractor and simulate the same using basic gates.

Objectives:

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

Theory:

The addition and subtraction operations can be combined into one circuit with one common binary adder by including an exclusive-OR gate with each full-adder.



The mode input M controls the operation as the following:

If M=0 → the circuit is an adder
M=1 → the circuit becomes a subtractor

Each exclusive-OR gate receives input M and one of the inputs of B.

- When M = 0, we have B⊕ 0 = B. The full adders receive the value of B, the input carry is C_0 = 0, and the circuit performs A + B.

- When M = 1, we have B ⊕ 1 = $\bar{B}$ and C_0 = 1. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A + (the 2's complement of B)= A –B
- The exclusive-OR with output V is for detecting an overflow.

The binary adder–subtractor circuit with outputs C and V is shown in Fig. 4.13. If the two binary numbers are considered to be unsigned, then the C bit detects a carry after addition or a borrow after subtraction. If the numbers are considered to be signed, then the V bit detects an overflow. If V = 0 after an addition or subtraction, then no overflow occurred and the n -bit result is correct. If V = 1, then the result of the operation contains n + 1 bits, but only the rightmost n bits of the number fit in the space available, so an overflow has occurred. The 1n + 12 th bit is the actual sign and has been shifted out of position.

Equipment / components required:

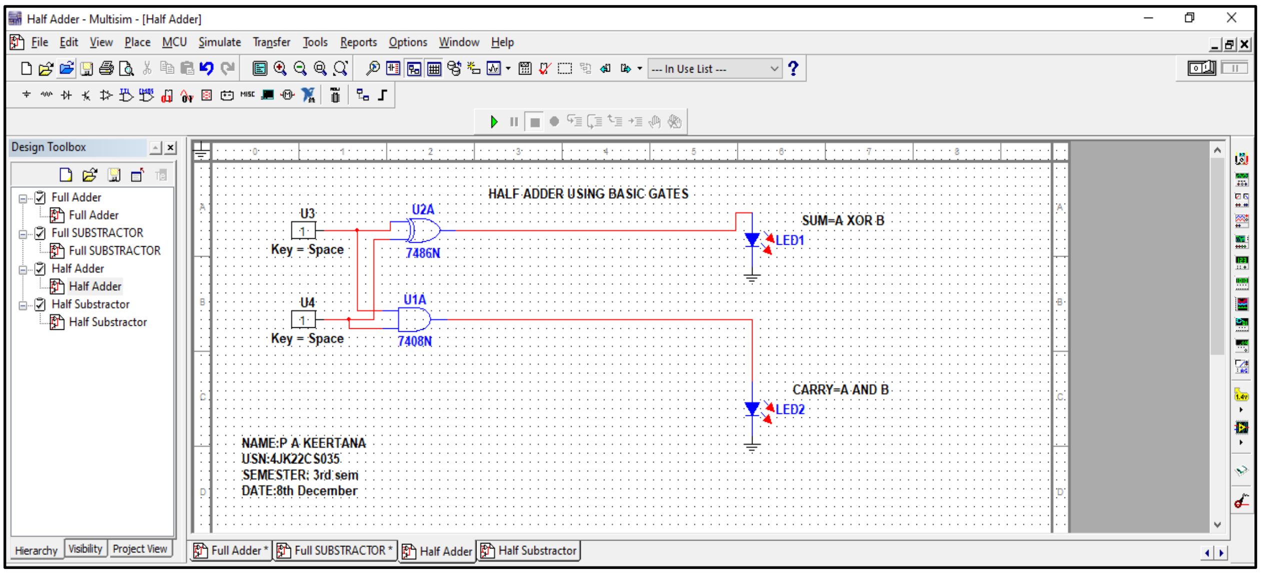
- Computer with Modelsim
- HP Computer i4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk
- Softwares: Modelsim - 5.7c

Procedure / activity:

1. Open Multisim software on your computer.
2. Click on "File" and select "New" to create a new schematic.
3. Drag and drop components from the component toolbar onto the workspace.
4. Connect the components using wires.Specify values and properties for components.
5. Add a power supply to the circuit. Connect it appropriately (Vcc and GND).
6. Double-click on components to open their properties.
7. Set parameters, such as resistor values, capacitor values, etc.
8. Click on "Simulate" in the toolbar.
9. Set simulation parameters like start time, stop time, and time step.
10. Add instruments to measure and observe circuit behavior.
11. Click on the "Run" button (play icon) to start the simulation.
12. Observe and analyze waveforms and measurements from the instruments. Check for correct circuit behavior.
13. If unexpected results occur, review the circuit, component values, and connections. Make necessary adjustments and rerun the simulation.
14. Save your schematic and simulation results.
15. Document the circuit design, simulation parameters, and observed behavior.
16. Export simulation results or circuit diagrams for sharing or further analysis.

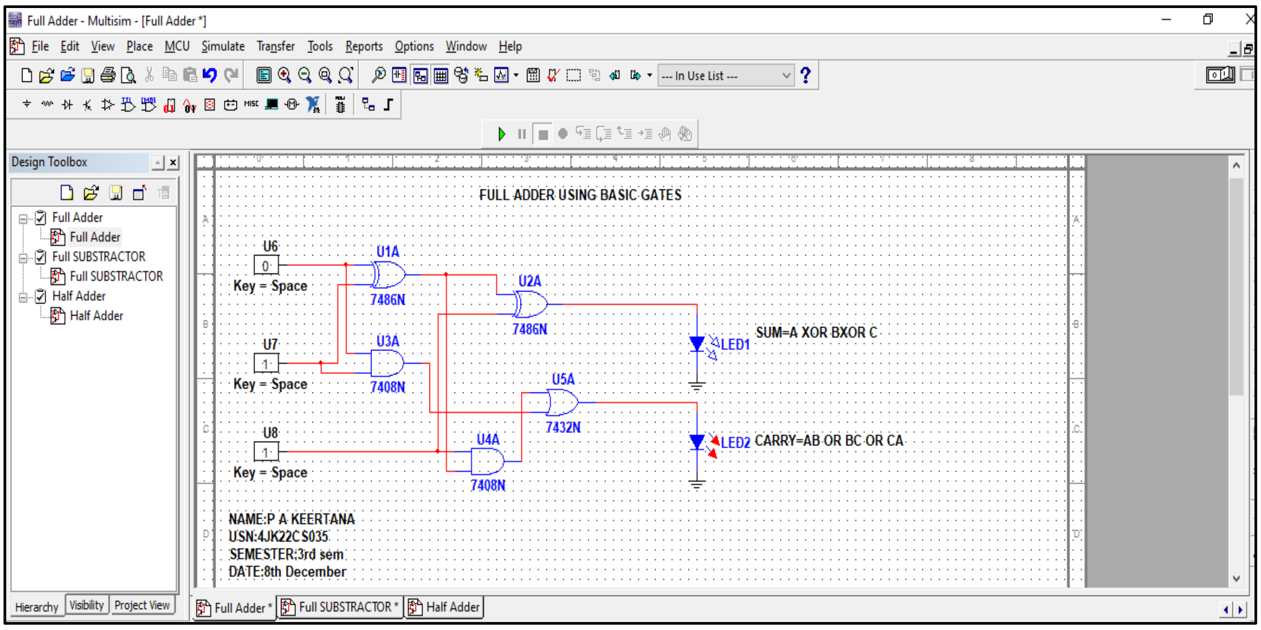
Simulation Output:

HALF ADDER



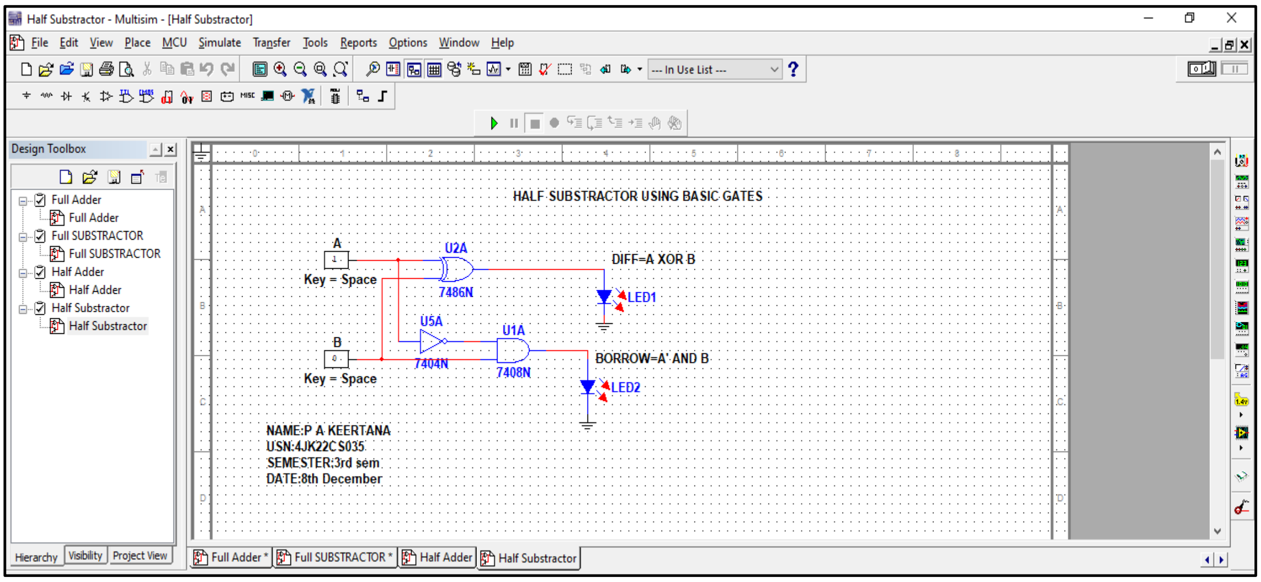
NAME:P A KEERTANA
USN:4JK22CS035
SEMESTER: 3rd sem
DATE:8th December

FULL ADDER



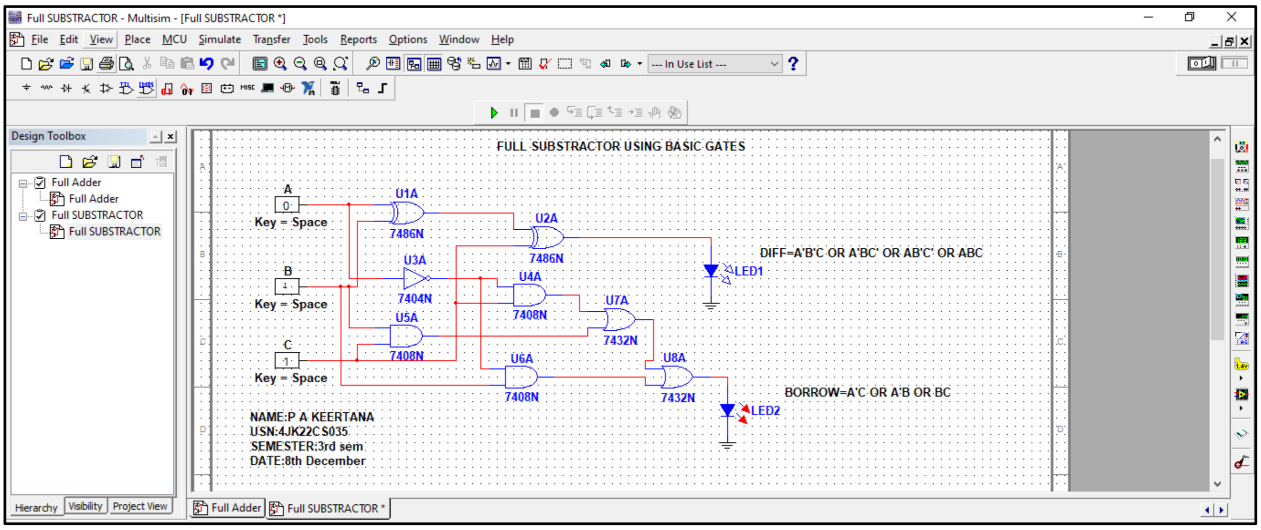
NAME:P A KEERTANA
USN:4JK22CS035
SEMESTER:3rd sem
DATE:8th December

HALF SUBSTRACTOR

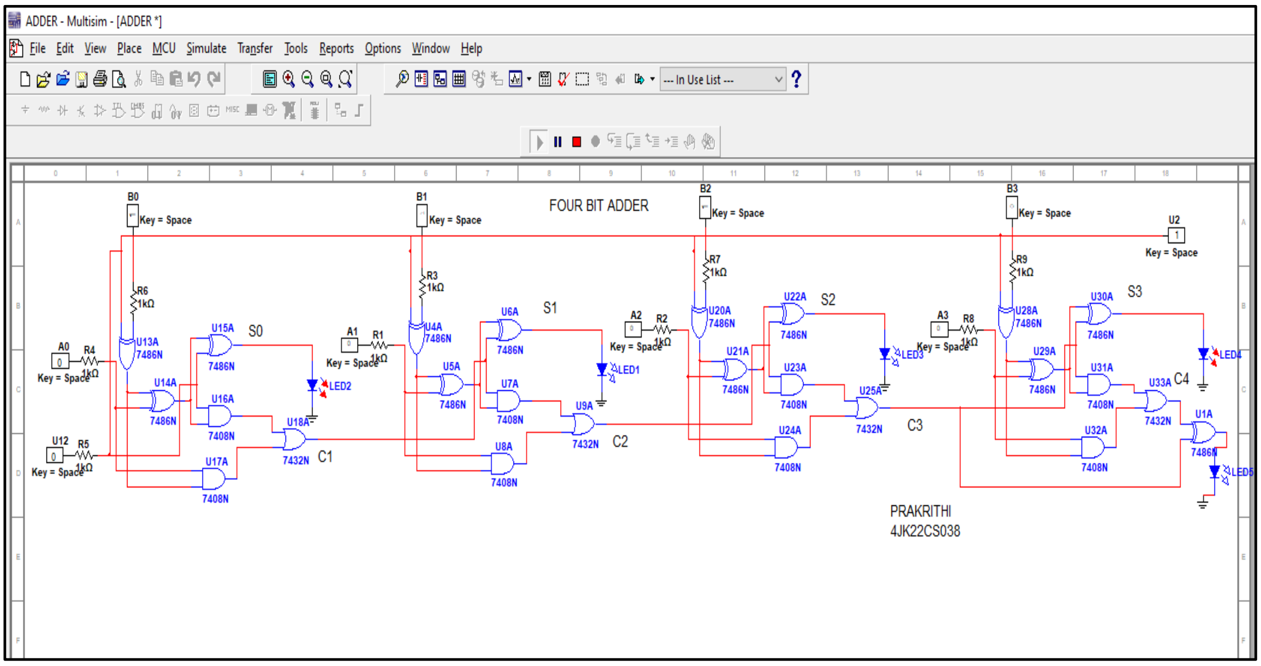


NAME:P A KEERTANA
USN:4JK22CS035
SEMESTER:3rd sem
DATE:8th December

FULL SUBSTRACTOR



4-BIT ADDER & SUBSTRACTOR


ADDER - Multisim - [ADDER *]
File Edit View Place MCU Simulate Transfer Tools Reports Options Window Help
FOUR BIT ADDER
B0
B1
B2
B3
U2
Key = Space
A0
A1
A2
A3
S0
S1
S2
S3
C1
C2
C3
C4
LED1
LED2
LED3
LED4
LED5
PRAKRITHI
4JK22CS038

Results & conclusions: Simulated 4-bit full adder and subtractor in Multisim, verified correct outputs, efficient gate use, and stable performance. Achieved accurate, optimized, and versatile circuit design.

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?**
   2

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) Perform bitwise 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 basic gates in a software 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-bit full 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 full subtractor?**

    Answer: One consideration is managing borrow inputs for each bit subtraction, and careful handling 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 and Behavioural 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**.**

Theory:

As mentioned previously, the module is the basic building block for modeling hardware with the Verilog HDL. The logic of a module can be described in anyone (or a combination) of the following modeling styles:

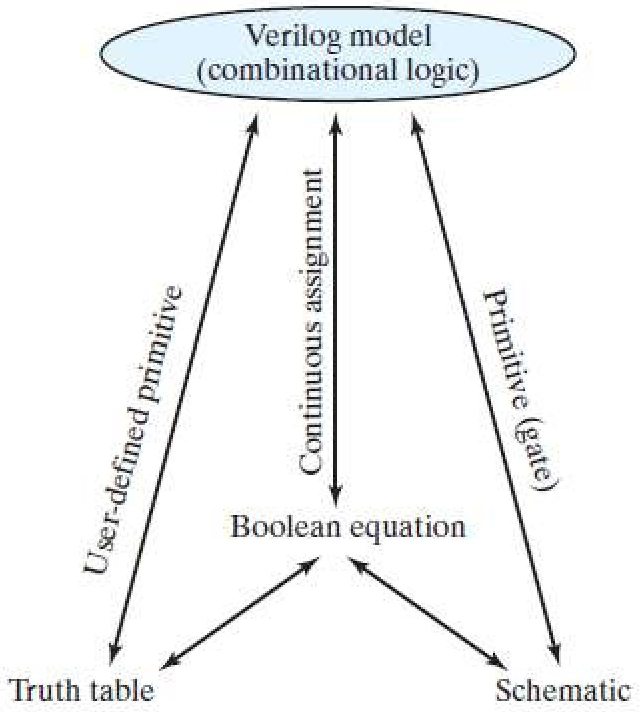


FIGURE 3.1: Relationship of Verilog constructs to truth tables, Boolean equations, and schematics

- **Gate-level modeling** using instantiations of predefined and user-defined primitive gates.
  - Gate-level (structural) modeling describes a circuit by specifying its gates and how they are connected with each other.

- **Dataflow modeling** using continuous assignment statements with the keyword assign.
    - Dataflow modeling is used mostly for describing the Boolean equations of combinational logic.
- **Behavioral modeling** using procedural assignment statements with the keyword always.
    - behavioral modeling that is used to describe combinational and sequential circuits at a higher level of abstraction.

Combinational logic can be designed with truth tables, Boolean equations, and schematics; Verilog has a construct corresponding to each of these "classical" approaches to design: user-defined primitives, continuous assignments, and primitives, as shown in Fig. 3.1. There is one other modeling style, called switch-level modeling. It is sometimes used in the simulation of MOS transistor circuit models, but not in logic synthesis. We will not consider switch-level modeling.

Equipment / components required:

- Computer with Modelsim SoftwareSpecifications:
- HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk
- Softwares: Modelsim - 5.7c, Xilinx - 14.1i.

Procedure / activity:

1. Open a Xilinx IDE, close old projects.
2. Create a new project through the project navigator icon.
3. Add VHDL/Verilog new source to the project depending on the user requirement.
4. Select the Verilog HDL mode.
5. Assign the inputs and outputs for the system to design.
6. And write the code as given below for the functionality.
7. Synthesize the code and correct the syntax errors if any.
8. Isim Simulator and Behavioral check syntax the Simulation behavioral model.
9. Observe the output timing waveform and verify it with the truth table.

Verilog code:

Structural-	**Data Flow-**	**Behavioural Model-**

module logicg(a,
b,y0,y1,y2,y3,y4,y5,y6);
input a,b;
output y0,y1,y2,y3,y4,y5,y6;
not g1(y0,a);
and g2(y1,a,b);
or g3(y2,a,b);
nand g4(y3,a,b);
nor g5(y4,a,b);
xor g6(y5,a,b);
xnor g7(y6,a,b);
endmodule
	module ldataflow(a,
	b,y0,y1,y2,y3,y4,y5,y6);
	input a,b;
	output y0,y1,y2,y3,y4,y5,y6;
	assign y0=!a;
	assign y1=a&b;
	assign y2=a|b;
	assign y3=!(a&b);
	assign y4=!(a|b);
	assign y5=a^b;
	assign y6=!(a^b);
	endmodule
		module logicgB(a,
		b,y0,y1,y2,y3,y4,y5,y6);
		input a,b;
		output y0,y1,y2,y3,y4,y5,y6;
		reg y0;
		reg y1;
		reg y2;
		reg y3;
		reg y4;
		reg y5;
		reg y6;
		always @(a,b)

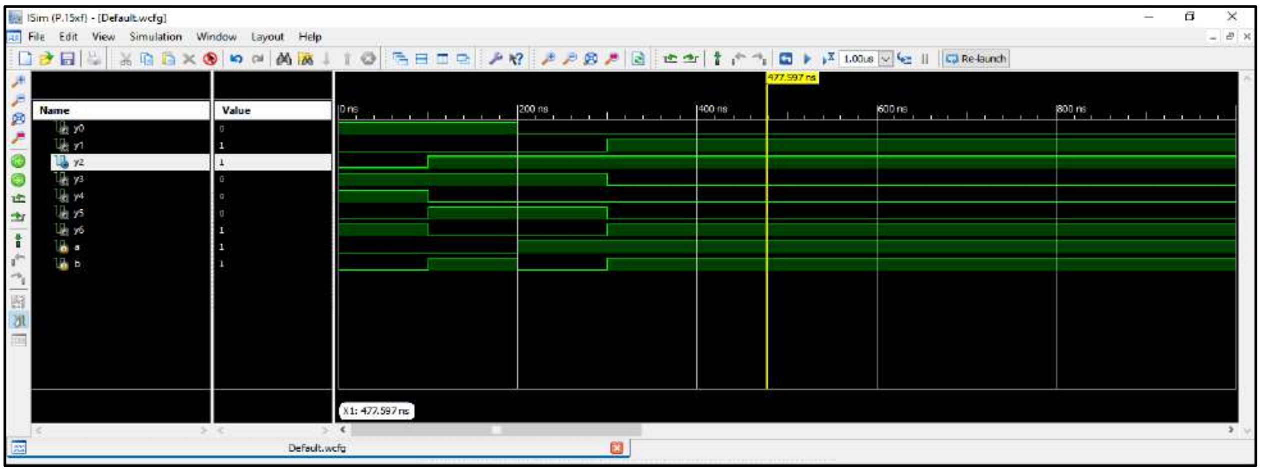
		begin
		y0=!a;
		y1=a&b;
		y2=a|b;
		y3=!(a&b);
		y4=!(a|b);
		y5=a^b;
		y6=!(a^b);
		end
		endmodule

Verilog Testbench code:

Structural-	**Data Flow-**	**Behavioural Model-**

module logicg_tb;
reg a;
reg b;
wire y0;
wire y1;
wire y2;
wire y3;
wire y4;
wire y5;
wire y6;
logicg uut (.a(a), .b(b),
.y0(y0), .y1(y1),
.y2(y2), .y3(y3),
.y4(y4), .y5(y5),
.y6(y6));
initial begin
a = 0; b = 0;#100;
a = 0; b = 1;#100;
a = 1; b = 0;#100;
a = 1; b = 1;#100;
end
endmodule
	module ldataflow_tb;
	reg a;
	reg b;
	wire y0;
	wire y1;
	wire y2;
	wire y3;
	wire y4;
	wire y5;
	wire y6;
	ldataflow uut (.a(a), .b(b),
	.y0(y0), .y1(y1),
	.y2(y2), .y3(y3),
	.y4(y4), .y5(y5),
	.y6(y6));
	initial begin
	a = 0; b = 0;#100;
	a = 0; b = 1;#100;
	a = 1; b = 0;#100;
	a = 1; b = 1;#100;
	end
	endmodule
		module logicgB_tb;
		reg a;
		reg b;
		wire y0;
		wire y1;
		wire y2;
		wire y3;
		wire y4;
		wire y5;
		wire y6;
		logicgB uut (.a(a), .b(b),
		.y0(y0), .y1(y1),
		.y2(y2), .y3(y3), .y4(y4),
		.y5(y5), .y6(y6));
		initial begin
		a = 0; b = 0; #100;
		a = 0; b = 1; #100;
		a = 1; b = 0; #100;
		a = 1; b = 1; #100;
		end
		endmodule

Simulation Output:



Results & conclusions: The Verilog code for simple circuits using structural, Data flow and Behavioral model are verified.

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.

Theory:

Adder is a combinational digital circuit that is used for adding two numbers. A typical adder circuit produces a sum bit (denoted by S) and a carry bit (denoted by C) as the output. Adder circuits are of two types: Half adder and Full adder.

Half-Adder: A combinational logic circuit that performs the addition of two data bits, A and B, is called a half-adder. Addition will result in two output bits; one of which is the sum bit, S, and the other is the carry bit, C.

Full-Adder: The half-adder does not take the carry bit from its previous stage into account. This carry bit from its previous stage is called carry-in bit. A combinational logic circuit that adds two data bits, A and B, and a carry-in bit Cin, is called a full-adder.

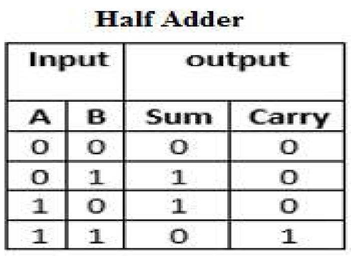
Subtractor is a combinational digital circuit that is used for subtracting two numbers. A typical subtractor circuit produces a diff bit (denoted by D) and a borrow bit (denoted by B)as the output. Subtractor circuits are of two types: Half subtractor and Full subtractor.

Half Subtractor: A combinational logic circuit that performs the subtraction of two data bits, A and B, is called a half-adder. Subtraction will result in two output bits; one of which is the diff bit, D, and the other is the borrow bit, B.

Full Subtractor: A combinational logic circuit that subtractor two data bits, A and B and a borrow in bit, is called full subtractor.

Using Basic Gates

Half Adder

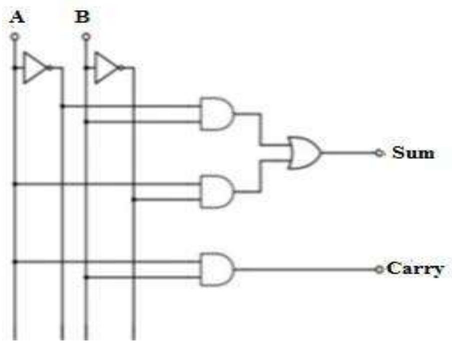


Half Adder

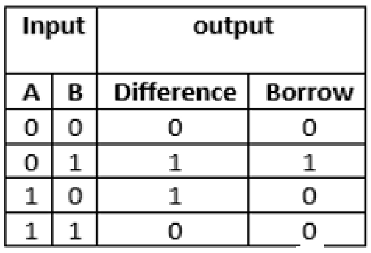
Input		output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Sum= $\bar{A}B+A\bar{B}$

Carry=AB



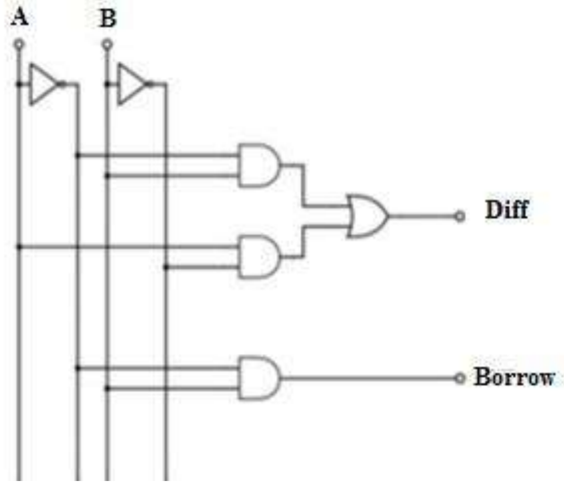
Half Subtractor



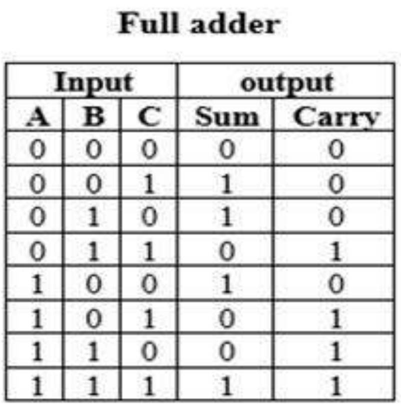
Input		output	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Difference= $\bar{A}B+A\bar{B}$

Borrow= $\bar{A}B$



Full Adder

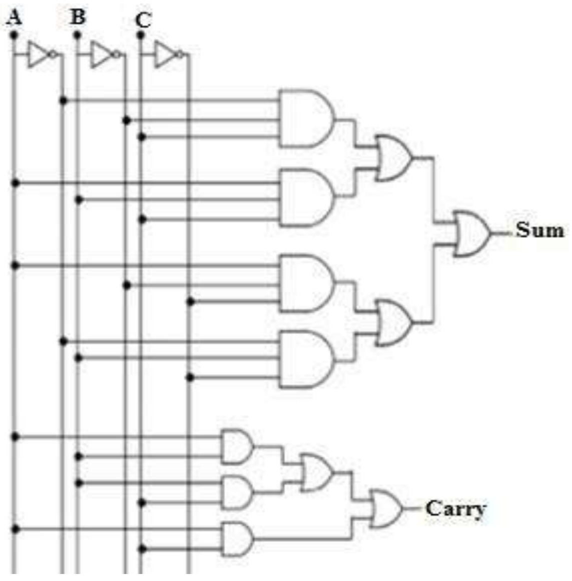


Full adder

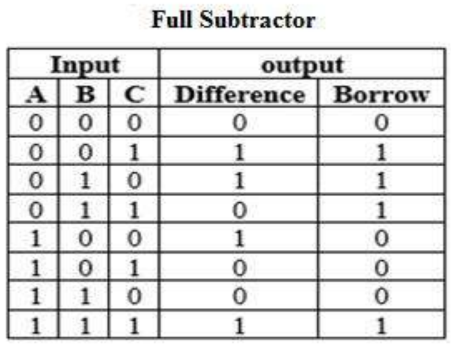
Input			output	
A	B	C	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sum = $\bar{a}\,\bar{b}\,c + \bar{a}\,b\,\bar{c} + a\,\bar{b}\,\bar{c} + a\,b\,c$

Carry = $a{\cdot}b + a{\cdot}c + b{\cdot}c$



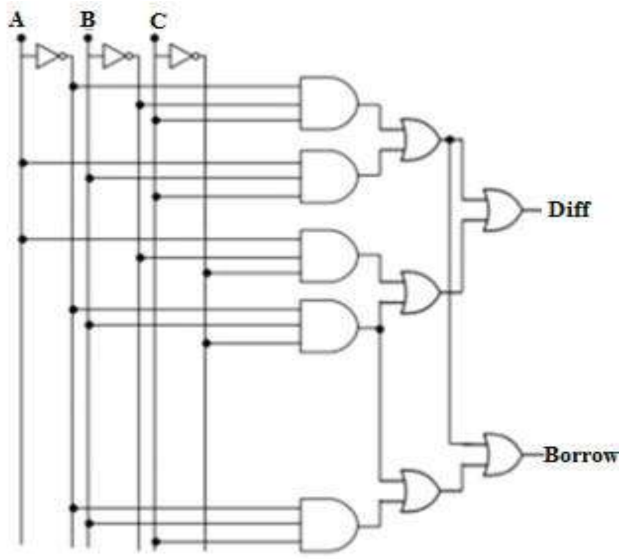
Full Subtractor



Full Subtractor

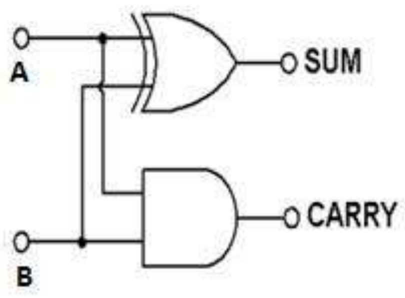
Input			output	
A	B	C	Difference	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$\text{Difference} = \bar{a}\,\bar{b}\,c + \bar{a}\,b\,\bar{c} + a\,\bar{b}\,\bar{c} + a\,b\,c$$
$$\text{Borrow} = \bar{a}\,c + \bar{a}\,b + b\,c$$



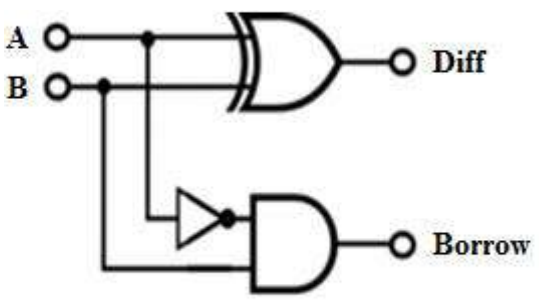
Using EX-OR Gates

Half Adder



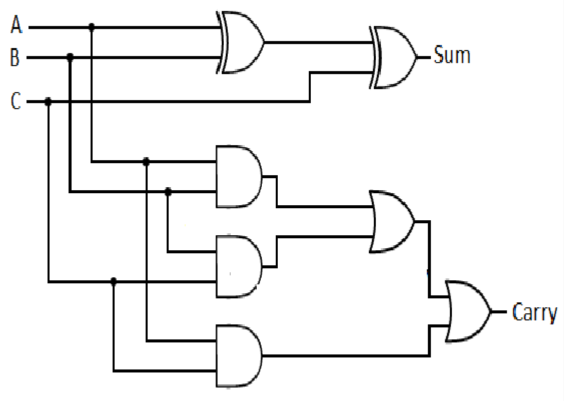
Sum= A⊕B
Carry= AB

Half Subtractor



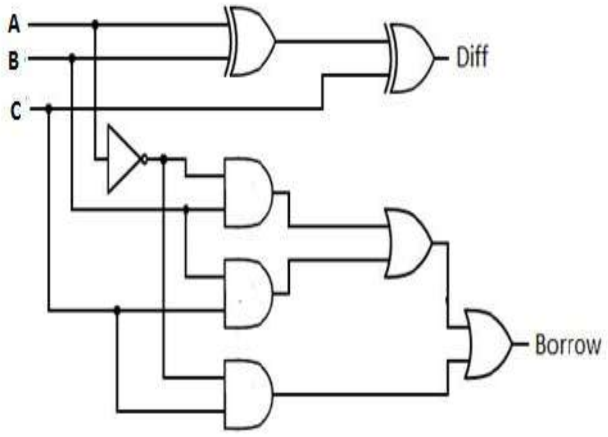
Diff= A⊕B
Borrow=A'B

Full Adder



Sum=A⊕B⊕C
Carry=AB+BC+AC

Full Subtractor



Diff= A⊕B⊕C
Borrow=A'B+BC+A'C

Equipment / components required:

- Computer with Modelsim SoftwareSpecifications:
- HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk
- Softwares: Modelsim - 5.7c, Xilinx - 14.1i.

Procedure / activity:

1. Open a Xilinx IDE, close old projects.
2. Create a new project through the project navigator icon.
3. Add VHDL/Verilog new source to the project depending on the user requirement.
4. Select the Verilog HDL mode.
5. Assign the inputs and outputs for the system to design.
6. And write the code as given below for the functionality.
7. Synthesize the code and correct the syntax errors if any.
8. Isim Simulator and Behavioral check syntax the Simulation behavioral model.
9. Observe the output timing waveform and verify it with the truth table.

Verilog code:

Half Adder-	**Half Substractor-**

module halfadder(a,b,sum,carry);
  input a,b;
  output sum,carry;
  assign sum=a^b;
  assign carry=a&b;
endmodule
	module HS(a,b,diff,borrow);
	input a,b;
	output diff,borrow;
	assign diff=a^b;
	assign borrow=!(a)&b;
	endmodule

Full Adder-	**Full Substractor-**

module FA(a,b,c,sum,carry);
input a,b,c;
output sum,carry;
assign sum=a^b^c;
assign carry=(a&b)||(b&c)||(c&a);
endmodule
	module FS(a,b,c,diff,borrow);
	input a,b,c;
	output diff,borrow;
	assign diff=a^b^c;
	assign borrow=((!a)&b)|(b&c)|((!a)&c);
	endmodule

Verilog Testbench code:

Half Adder-	**Half Substractor-**

module halfadder_tb;
    reg a;
    reg b;
    wire sum;
    wire carry;
    halfadder uut (
        .a(a),
        .b(b),
	module HS_tb;
	    reg a;
	    reg b;
	    wire diff;
	    wire borrow;
	    HS uut (
	        .a(a),
	        .b(b),

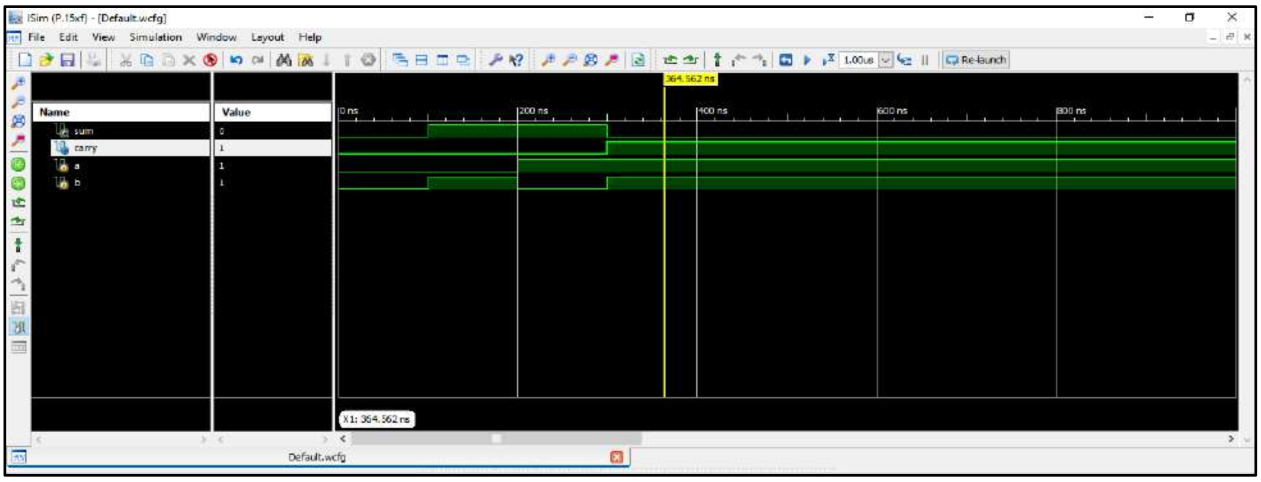
.sum(sum),
.carry(carry));
initial begin
a = 0; b = 0; #100;
a = 0; b = 1; #100;
a = 1; b = 0; #100;
a = 1; b = 1; #100;
end
endmodule
	.diff(diff),
	.borrow(borrow));
	initial begin
	a = 0; b = 0; #100;
	a = 0; b = 1; #100;
	a = 1; b = 0; #100;
	a = 1; b = 1; #100;
	end
	endmodule

Full Adder-	**Full Substractor-**

module FA_tb;
reg a;
reg b;
reg c;
wire sum;
wire carry;
FA uut (
.a(a),
.b(b),
.c(c),
.sum(sum),
.carry(carry));
initial begin
a = 0;b = 0;c = 0;#100;
a = 0;b = 0;c = 1;#100;
a = 0;b = 1;c = 0;#100;
a = 0;b = 1;c = 1;#100;
a = 1;b = 0;c =0;#100;
a = 1;b = 0;c = 1;#100;
a =1;b = 1;c = 0;#100;
a = 1;b = 1;c = 1;#100;
end
endmodule
	module FS_tb;
	reg a;
	reg b;
	reg c;
	wire diff;
	wire borrow;
	FS uut (
	.a(a),
	.b(b),
	.c(c),
	.diff(diff),
	.borrow(borrow));
	initial begin
	a = 0;b = 0;c = 0;#100;
	a = 0;b = 0;c = 1;#100;
	a = 0;b = 1;c = 0;#100;
	a = 0;b = 1;c = 1;#100;
	a = 1;b = 0;c = 0;#100;
	a = 1;b = 0;c = 1;#100;
	a = 1;b = 1;c = 0;#100;
	a = 1;b = 1;c = 1;#100;
	end
	endmodule

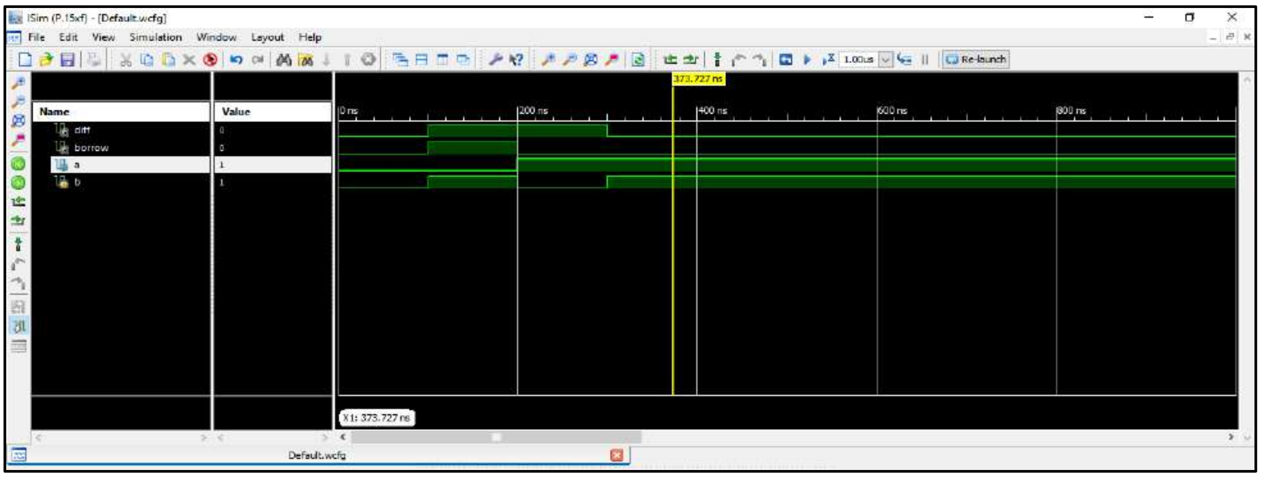
Simulation Output:

Half Adder-

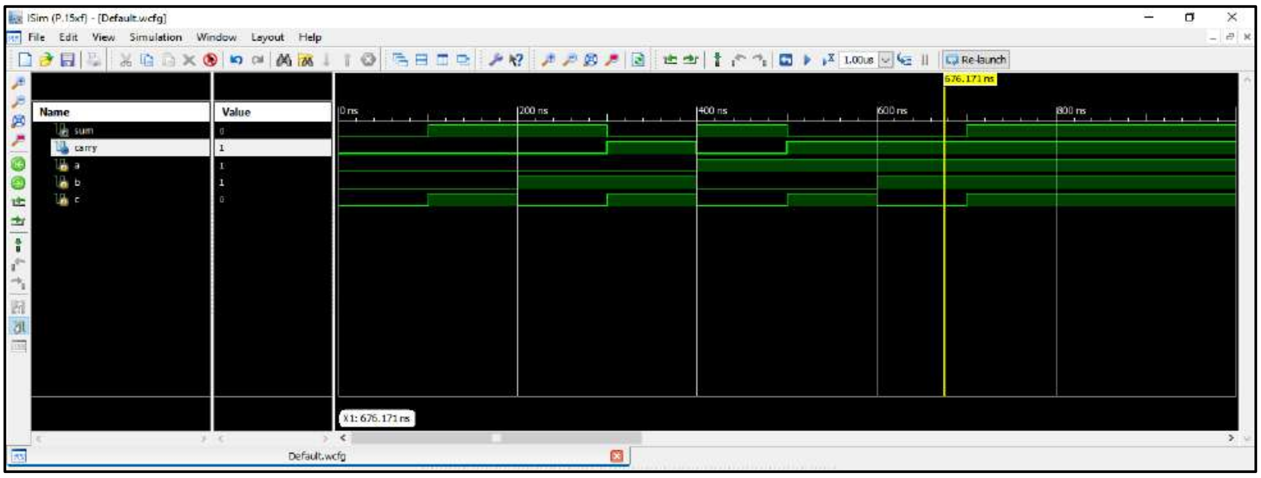


Name	Value
sum	0
carry	1
a	1
b	1

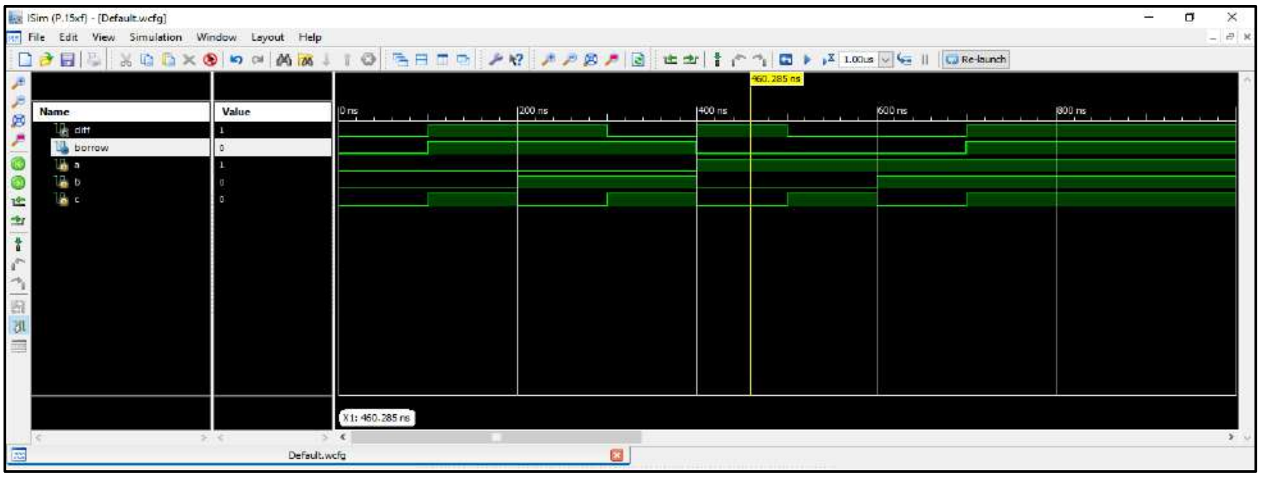
Half Substractor-



Full Adder-



Full Substractor



Results & conclusions: The Verilog code for adders and subtractors are verified.

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.

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.

9. 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.

10. 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.

Theory: BCD or Binary coded decimal is a way of representing decimal digits in binary form. Generally 4 bits are used to represent values 0 to 9.

Decimal Digit	BCD 8 4 2 1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0

Decimal Digit	BCD 8 4 2 1
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

A BCD adder, takes in two BCD numbers as inputs, and outputs a BCD digit along with a carry output. If the sum of the BCD digits is less than or equal to 9, then we don't have a problem. But if its greater than 9, we have to convert it into BCD format. This is done by adding 6 to the sum and taking only the least significant 4 bits. The MSB bit is output as carry.

Consider the below BCD addition:

1001 + 1000 = 10001

9 +   8 =   17

The output 17 is more than 9. So we add 6 to it. So we get,

17+6 = 23 (in binary 23 is 10111)

Now the least significant 4 bits (which is "0111") represent the units digit and the MSB(4th bit which is '1') bit represents the tens digit. Since the range of input is 0 to 9, the maximum output is 18. If you consider a carry it becomes 19. This means at the output side we need a 4 bit sum and a 1 bit carry to represent the most significant digit. These binary numbers are listed below

The condition for a correction and an output carry can be expressed by the Boolean Function

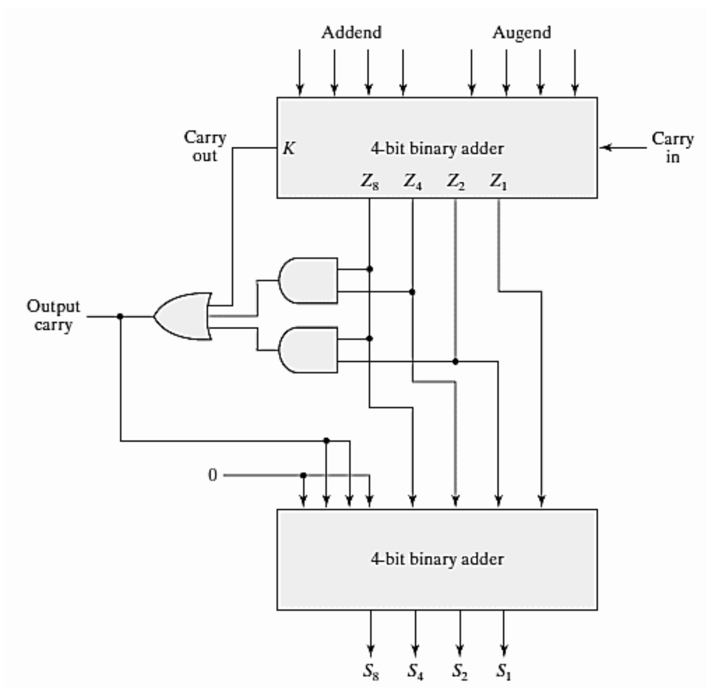
$$C = K + Z_8Z_4 + Z_8Z_2$$

When C = 1, it is necessary to add 0110 to the binary sum and provide an output carry for the next stage. A BCD adder that adds two BCD digits and produces a sum digit in BCD is shown in Figure below.

Derivation of BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

Logic Diagram:



For multiple digit addition, you can connect the carry_out to the carry input of the next adder. A simple cascading network of these small adders is enough to realize the multiple digit BCD addition.

Equipment:

- Computer with Modelsim SoftwareSpecifications:
- HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk
- Softwares: Modelsim - 5.7c, Xilinx - 14.1i.

Procedure:

1. Open a Xilinx IDE, close old projects.
2. Create a new project through the project navigator icon.

3. Add VHDL/Verilog new source to the project depending on the user requirement.
4. Select the Verilog HDL mode.
5. Assign the inputs and outputs for the system to design.
6. And write the code as given below for the functionality.
7. Synthesize the code and correct the syntax errors if any.
8. Isim Simulator and Behavioral check syntax the Simulation behavioral model.
9. Observe the output timing waveform and verify it with the truth table.

Verilog code for BCD addition-Behavioral level:

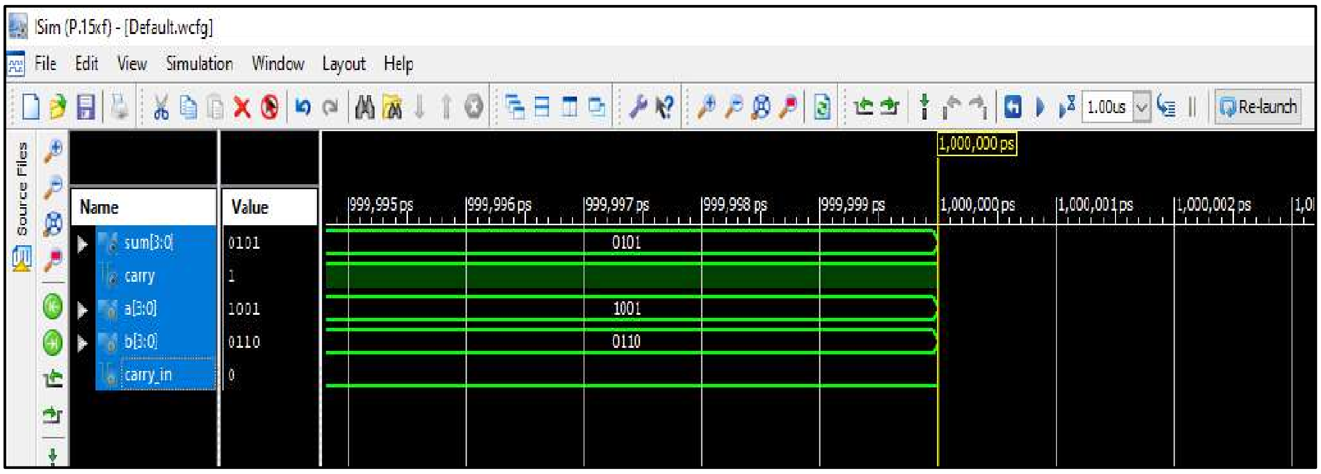
```
module deciadder(a,b,carry_in,sum,carry);
                        //declare the inputs and outputs of the module with their sizes.
  input [3:0] a,b;
  input carry_in;
  output [3:0] sum;
  output carry;
                                   //Internal variables
  reg [4:0] sum_temp;
  reg [3:0] sum;
  reg carry;
                                  //always block for doing the addition
  always @(a,b,carry_in)
  begin
       sum_temp = a+b+carry_in;                    //add all the inputs
   if(sum_temp > 9)
      begin
      sum_temp = sum_temp+6;                      //add 6, if result is more than 9.
      carry = 1;  //set the carry output
      sum = sum_temp[3:0];
      end
    else
      begin
      carry = 0;
      sum = sum_temp[3:0];
    end
  end
endmodule
```

Verilog Testbench code for BCD adder:

```
module deciadder_tb;
      reg [3:0] a;
      reg [3:0] b;
      reg carry_in;
      wire [3:0] sum;
```

```
        wire carry;
    deciadder uut (.a(a), .b(b), .carry_in(carry_in), .sum(sum), .carry(carry));
        initial begin
            // Initialize Inputs
            a = 9;
            b = 6;
            carry_in = 0;
            #100;
        end
    endmodule
```

Simulation waveform:



Name	Value
sum[3:0]	0101
carry	1
a[3:0]	1001
b[3:0]	0110
carry_in	0

<u>Results & conclusions:</u> The Verilog code for Decimal adder is simulated and verified.

<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.

Theory:

- A multiplexer (or data selector, abbreviated as MUX) has a group of data inputs (2^n) and a group of control inputs (n) (also called as select inputs).
- The control inputs are used to select one of the data inputs and connect it to the output terminal.

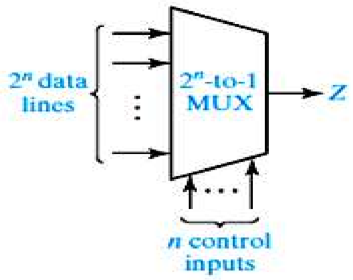


Figure 6.1: General block diagram of 2^n : 1 multiplexer

- A general block diagram of 2^n : 1 multiplexer is shown in the fig.3.6.
- A 2 to 1 multiplexer requires 1 select input, 4 to 1 multiplexers require 2 select inputs and 8 to 1 multiplexers require 3 select inputs.

2:1 Multiplexer

- A 2 to 1 multiplexer has 2 data inputs, 1 select input and 1 output.

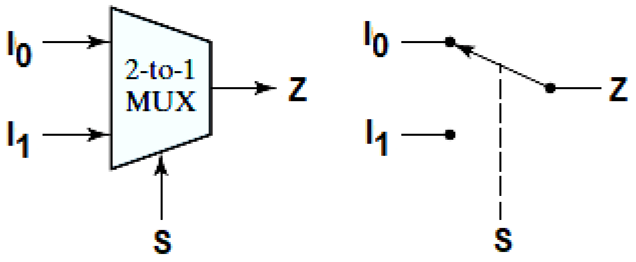


Figure 6.2: 2 to 1 Multiplexer and Switch Analog

- When the select (control) input A is 0, the switch is in the upper position and the MUX output is $Z = I_0$.
- When the select (control) input A is 1, the switch is in the lower position and the MUX output is $Z = I_1$.
- In other words, a MUX acts like a switch that selects one of the data inputs (I_0 or I_1) and transmits it to the output.

Truth Table:

Select (S)	Output (Z)
0	I_0
1	I_1

- The logic equation for the 2-to-1 MUX can be written as:

$$\mathbf{Z = \bar{S}I_0 + SI_1}$$

Logic Diagram:

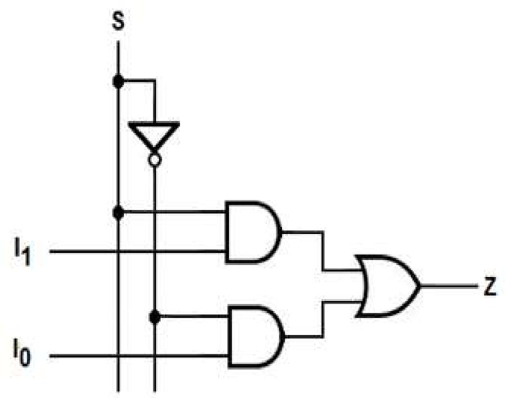


Figure 6.3: Logic diagram of 2:1 Multiplexer

4:1 Multiplexer

- A 4 to 1 multiplexer has 4 data inputs, 2 select inputs and 1 output.
- The 4 to 1 MUX acts like a four-position switch that transmits one of the four inputs to the output.

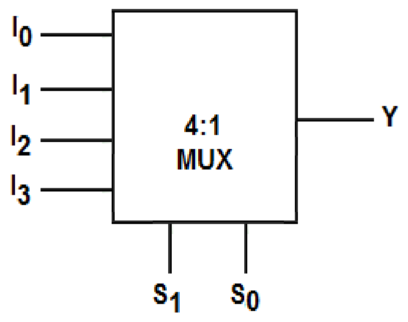


Figure 6.4: 4 to 1 Multiplexer

- Two select (control) inputs (S_1 and S_0) are needed to select one of the four inputs. If the control inputs are S_1S_0 = 00, the output is I_0; similarly, for the control inputs 01, 10, and 11 give outputs of I_1, I_2, and I_3, respectively.

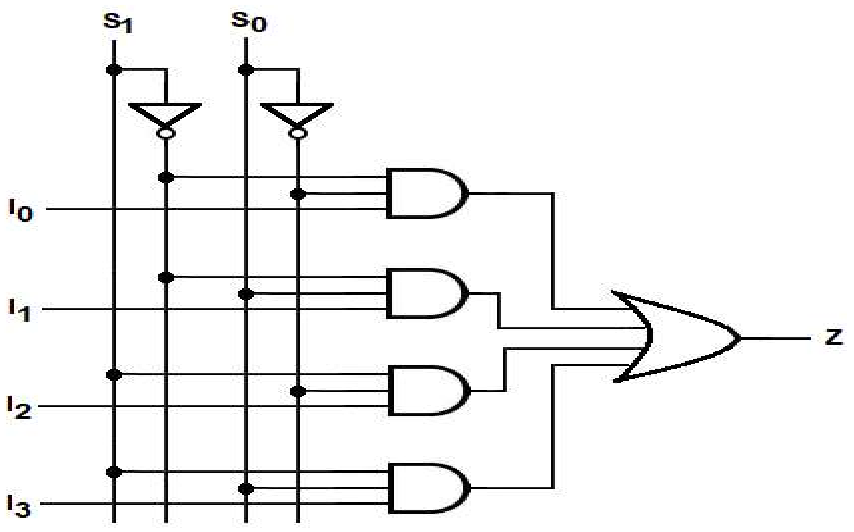
Truth Table:

Select (S_1)	Select (S_0)	Output (Z)
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

- The logic equation for 4-to-1 multiplexer can be written as

$$\mathbf{Z = \bar{S}_1\bar{S}_0I_0 + \bar{S}_1S_0I_1 + S_1\bar{S}_0I_2 + S_1S_0I_3}$$

Logic Diagram:



8:1 Multiplexer

- An 8 to 1 multiplexer has 8 data inputs, 3 select inputs and 1 output. 8 to 1 MUX selects one of eight data inputs using three select inputs

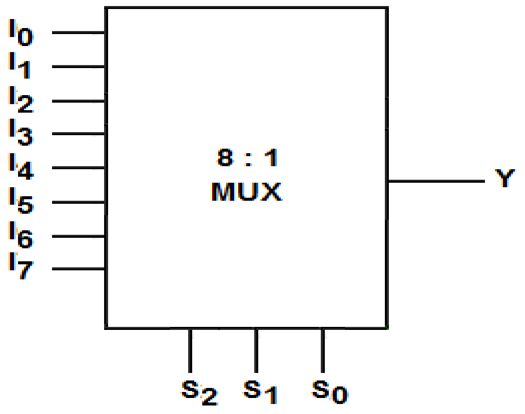


Figure 6.5: 8 to 1 Multiplexer

- The 8-to-1 MUX acts like an eight-position switch that transmits one of the eight inputs to the output.

Truth Table:

elect (S_2)	Select (S_1)	Select (S_0)	Output (Z)
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

- The logic equation for the 8 to1 MUX can be written as:

$$Z = \bar{S}_2\bar{S}_1\bar{S}_0I_0 + \bar{S}_2\bar{S}_1S_0I_1 + \bar{S}_2S_1\bar{S}_0I_2 + \bar{S}_2S_1S_0I_3 + S_2\bar{S}_1\bar{S}_0I_4 + S_2\bar{S}_1S_0I_5 + S_2S_1\bar{S}_0I_6 + S_2S_1S_0I_7$$

Equipment / components required:

- Computer with Xilinx - 14.7i. SoftwareSpecifications:
- HP Computer i4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk
- Softwares: Xilinx - 14.7i.

Procedure / activity:

1. Open a Xilinx IDE, close old projects.
2. Create a new project through the project navigator icon.
3. Add VHDL/Verilog new source to the project depending on the user requirement.
4. Select the Verilog HDL mode.
5. Assign the inputs and outputs for the system to design.
6. And write the code as given below for the functionality.
7. Synthesize the code and correct the syntax errors if any.
8. Isim Simulator and Behavioral check syntax the Simulation behavioral model.
9. Observe the output timing waveform and verify it with the truth table.

Verilog code:

2:1 MUX	4:1 MUX	8:1 MUX

module two_to_one_MUX
(I0,I1,S,Y);
input I0;
input I1;
input S;
output Y;
wire S_bar,w1,w2;
not g1(S_bar,S);
and (w1,I0,S_bar);
and (w2,I1,S);
or (Y,w1,w2);
endmodule
	module four_to_one_MUX
	(I1,I2,I3,I4,S1,S2,Y);
	input I1,I2,I3,I4;
	input S1,S2;
	output Y;
	
	assign
	Y=((!S1)&(!S2)&I1)|((!S1)&(S2)&I2)|(S1&(!S2)&I3)|(S1&S2&I4);
	endmodule
		module eight_to_one_MUX
		(I1,I2,I3,I4,I5,I6,I7,I8,S1,S2,S3,Y);
		input I1,I2,I3,I4,I5,I6,I7,I8;
		input S1,S2,S3;
		output Y;
		
		assign Y=((!S1)&(!S2)&(!S3)&I1)|((!S1)&(!S2)&S3&I2)|((!S1)&S2&(!S3)&I3)|((!S1)&S2&S3&I4)|(S1&(!S2)&(!S3)&I5)|(S1&(!S2)&S3&I6)|(S1&S2&(!S3)&I7)|(S1&S2&S3&I8);
		endmodule

Verilog Testbench code:

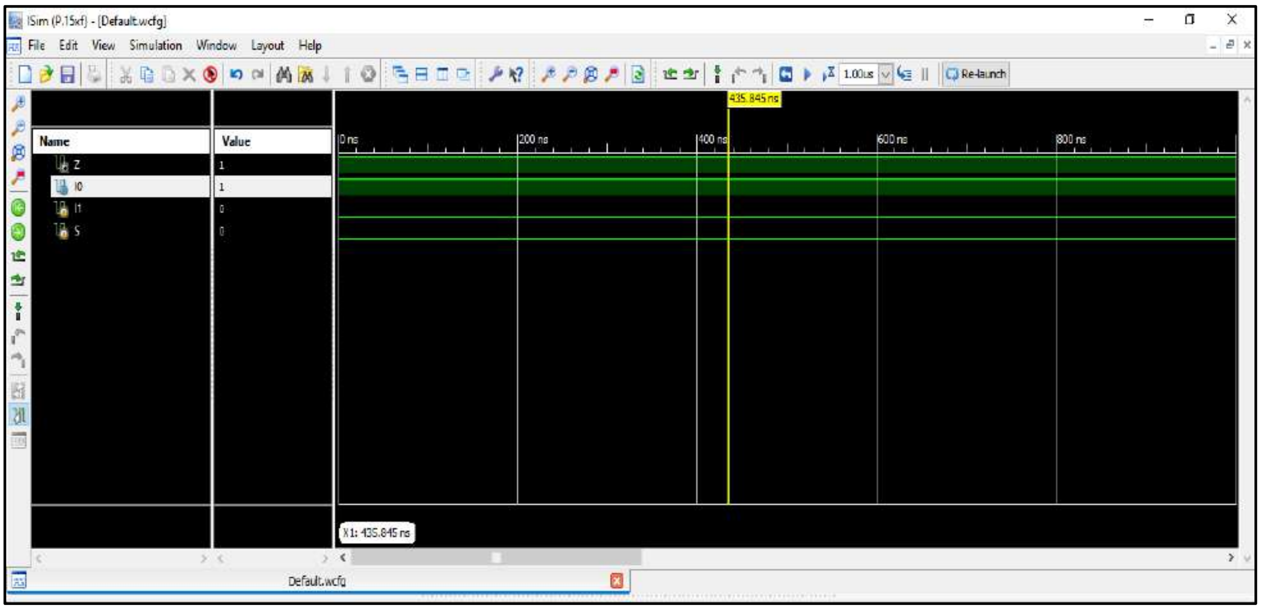
2:1 MUX	4:1 MUX	8:1 MUX

module two_to_one_MUX_tb;
reg I0;
reg I1;
reg S;
wire Y;
two_to_one_MUX uut (
.I0(I0),
	module four_to_one_MUX_tb;
	reg I1;
	reg I2;
	reg I3;
	reg I4;
	reg S1;
	reg S2;
		module eight_to_one_MUX1;
		reg I1;
		reg I2;
		reg I3;
		reg I4;
		reg I5;
		reg I6;

.I1(I1),
.S(S),
.Y(Y));
initial begin
I0 = 1; I1 = 0;
S = 0; #100;
end
endmodule
	wire Y;
	four_to_one_MUX uut (
	.I1(I1),
	.I2(I2),
	.I3(I3),
	.I4(I4),
	.S1(S1),
	.S2(S2),
	.Y(Y));
	initial begin
	I1 = 0;I2 = 0;I3 = 0;I4 = 1;
	S1 = 0;S2 = 1; #100;
	end
	endmodule
		reg I7;
		reg I8;
		reg S1;
		reg S2;
		reg S3;
		wire Y;
		eight_to_one_MUX uut (
		.I1(I1), .I2(I2), .I3(I3), .I4(I4),
		.I5(I5), .I6(I6), .I7(I7), .I8(I8),
		.S1(S1), .S2(S2), .S3(S3),
		.Y(Y));
		initial begin
		I1 = 1;I2 = 1;I3 = 1; I4 = 1;I5 =
		1;I6 = 0;I7 = 1;I8 = 1;
		S1 = 1; S2 = 0; S3 = 1; #100;
		end
		endmodule

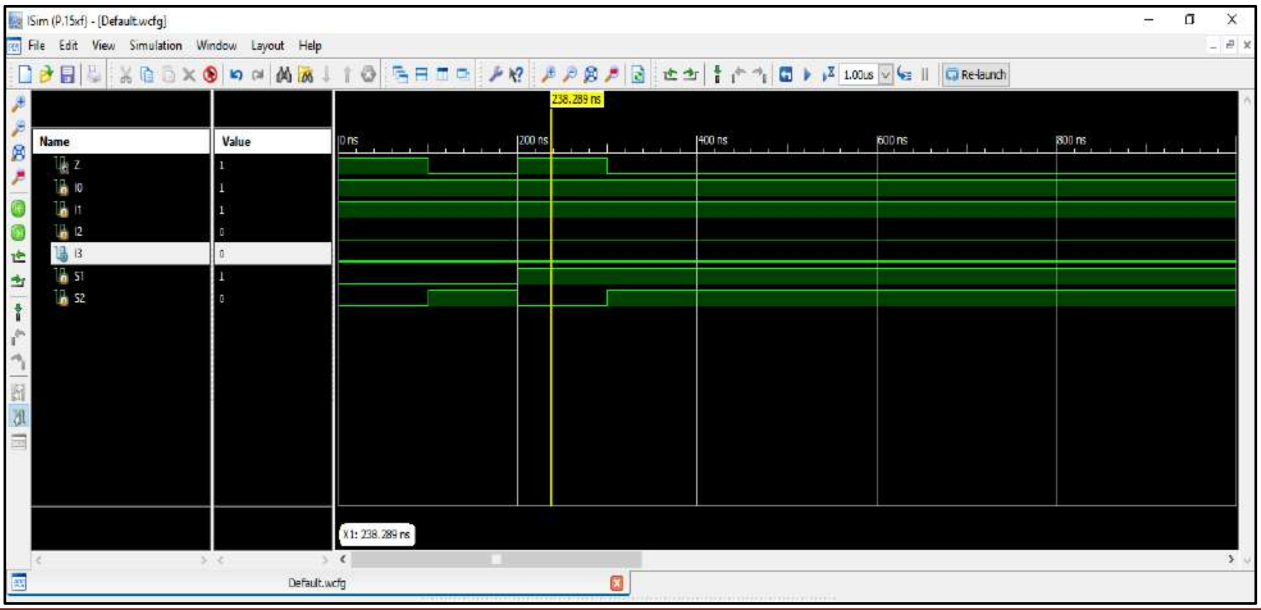
Simulation Output:

2:1 MUX



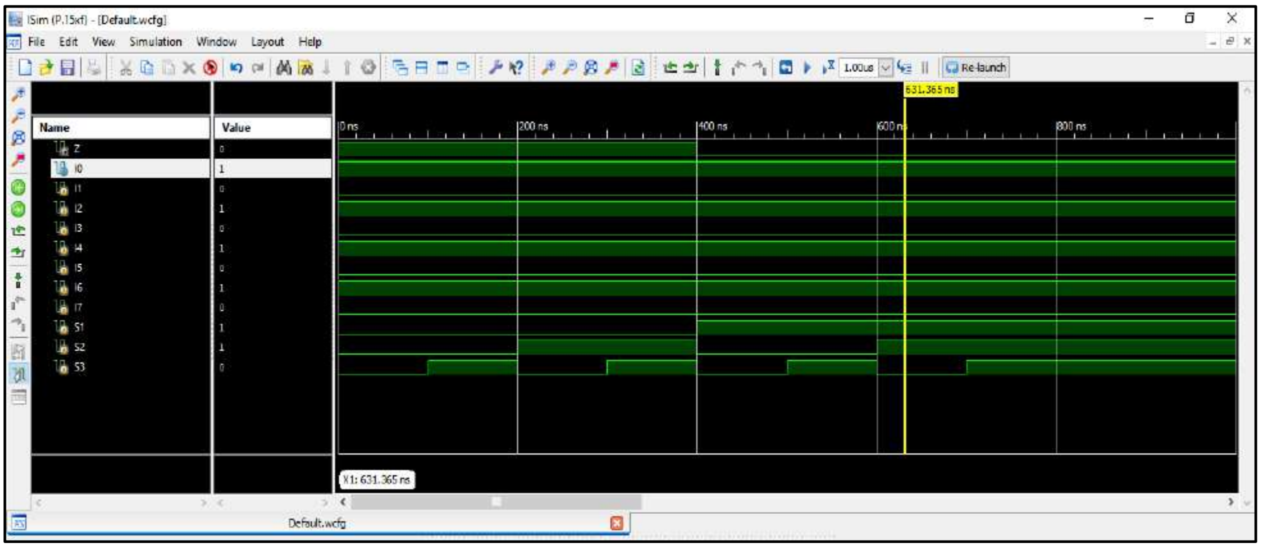
Name	Value
Z	1
I0	1
I1	0
S	0

4:1-MUX



Name	Value
Z	1
I0	1
I1	1
I2	0
I3	0
S1	1
S2	0

8:1 MUX



Results & conclusions:

The Verilog code for Different types of multiplexers like 2:1, 4:1 and 8:1 is simulated and verified**.**

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.

Theory:

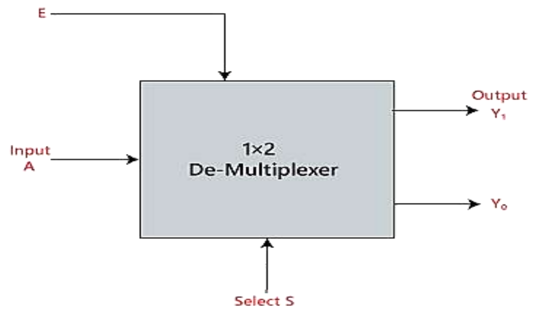
A De-multiplexer is a combinational circuit that has only 1 input line and 2^N output lines. Simply, the multiplexer is a single-input and multi-output combinational circuit. The information is received from the single input lines and directed to the output line. On the basis of the values of the selection lines, the input will be connected to one of these outputs. De-multiplexer is opposite to the multiplexer.

Unlike encoder and decoder, there are n selection lines and 2^n outputs. So, there is a total of 2^n possible combinations of inputs. De-multiplexer is also treated as **De-mux**.

There are various types of De-multiplexer which are as follows:

1×2 De-multiplexer: In the 1 to 2 De-multiplexer, there are only two outputs, i.e., Y_0, and Y_1, 1 selection lines, i.e., S_0, and single input, i.e., A. Based on the selection value, the input will be connected to one of the outputs. The block diagram and the truth table of the 1×2 multiplexer are given below.

Block Diagram:



Truth Table:

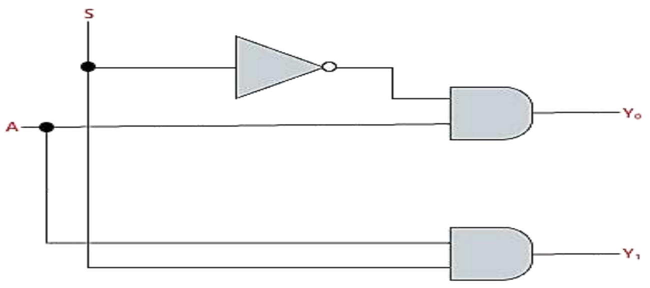
INPUTS	Output	
S_0	Y_1	Y_0
0	0	A
1	A	0

The logical expression of the term Y is as follows:

$$Y_0=S_0'.A$$

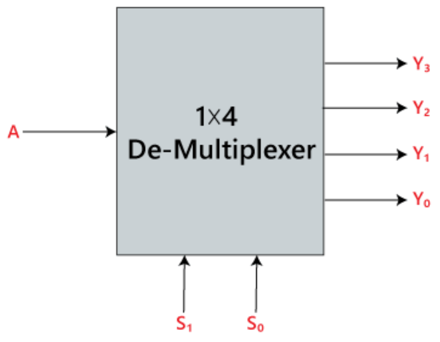
$$Y_1=S_0.A$$

Logical circuit of the above expressions is given below:



1×4 De-multiplexer: In 1 to 4 De-multiplexer, there are total of four outputs, i.e., Y_0, Y_1, Y_2, and Y_3, 2 selection lines, i.e., S_0 and S_1 and single input, i.e., A. Based on the combination of inputs which are present at the selection lines S_0 and S_1, the input be connected to one of the outputs. The block diagram and the truth table of the 1×4 multiplexer is given below.

Block Diagram: **Truth Table:**



INPUTS		Output			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	A
0	1	0	0	A	0
1	0	0	A	0	0
1	1	A	0	0	0

The logical expression of the term Y is as follows:

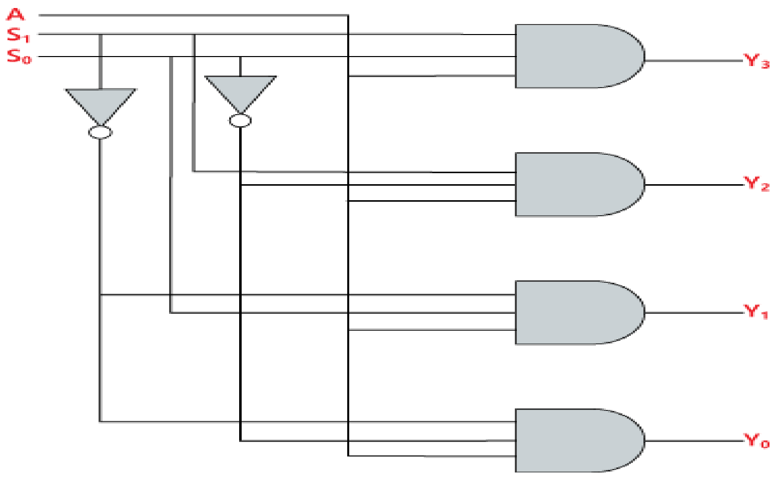
$\mathbf{Y_0=S_1' S_0' A}$

$\mathbf{y_1=S_1' S_0 A}$

$\mathbf{y_2=S_1 S_0' A}$

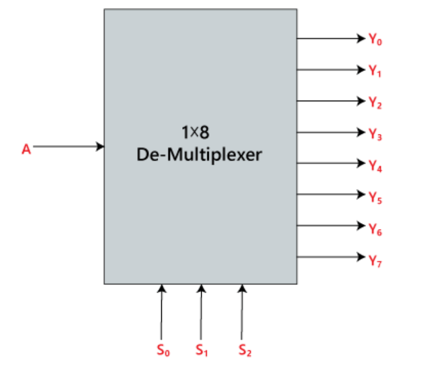
$\mathbf{y_3=S_1 S_0 A}$

Logical circuit of the above expressions is given below:



1×8 De-multiplexer: In 1 to 8 De-multiplexer, there are total of eight outputs, i.e., Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7, 3 selection lines, i.e., S_0, $S_1$and S_2 and single input, i.e., A. On the basis of the combination of inputs which are present at the selection lines S^0, S^1 and S_2, the input will be connected to one of these outputs. The block diagram and the truth table of the 1×8 de-multiplexer is given below.

Block Diagram:	Truth Table:



INPUTS			Output							
S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	A
0	0	1	0	0	0	0	0	0	A	0
0	1	0	0	0	0	0	0	A	0	0
0	1	1	0	0	0	0	A	0	0	0
1	0	0	0	0	0	A	0	0	0	0
1	0	1	0	0	A	0	0	0	0	0
1	1	0	0	A	0	0	0	0	0	0
1	1	1	A	0	0	0	0	0	0	0

The logical expression of the term Y is as follows:

$\mathbf{Y_0=S_0'.S_1'.S_2'.A}$

$\mathbf{Y_1=S_0.S_1'.S_2'.A}$

$\mathbf{Y_2=S_0'.S_1.S_2'.A}$

$\mathbf{Y_3=S_0.S_1.S_2'.A}$

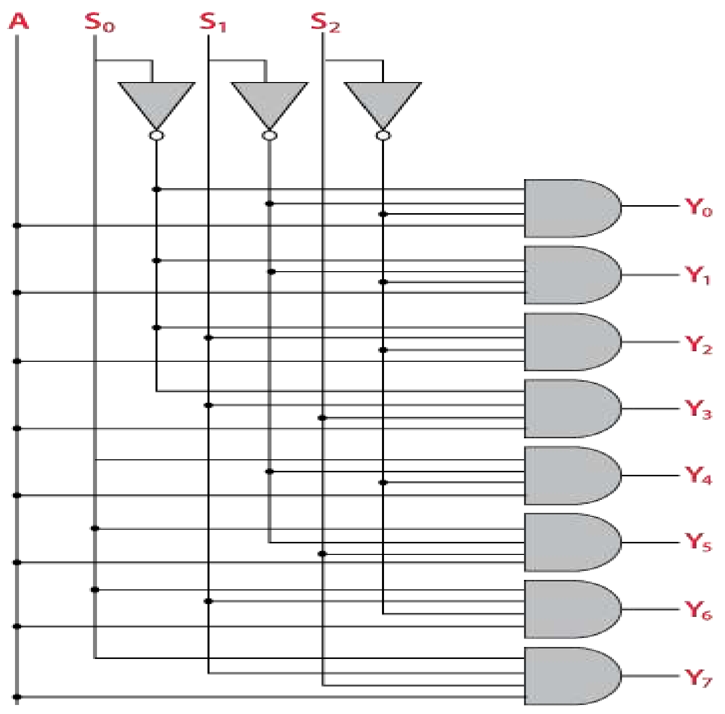
$\mathbf{Y_4=S_0'.S_1'.S_2\ A}$

$\mathbf{Y_5=S_0.S_1'.S_2\ A}$

$\mathbf{Y_6=S_0'.S_1.S_2\ A}$

$\mathbf{Y_7=S_0.S_1.S_3.A}$

Logical circuit of the above expressions is given below:



Equipment / components required:

- Computer with Xilinx - 14.7i.
- HP Computer i4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk
- Software: Xilinx ISE v14.7 IDE

Procedure / activity:

1. Open a Xilinx IDE, close old projects.
2. Create a new project through the project navigator icon.
3. Add VHDL/Verilog new source to the project depending on the user requirement.
4. Select the Verilog HDL mode.
5. Assign the inputs and outputs for the system to design.
6. And write the code as given below for the functionality.
7. Synthesize the code and correct the syntax errors if any.
8. Isim Simulator and Behavioral check syntax the Simulation behavioral model.
9. Observe the output timing waveform and verify it with the truth table.

Verilog code:

1:2 DEMUX	**1:4 DEMUX**	**1:8 DEMUX**

module demux12_BM(S0,A,Y);
   input A, S0;
   wire A, S0;
   output [1:0] Y;
   reg [1:0] Y;
always @ (A or S0)
   begin
   case(S0)
   1'b0: Y={A,1'b0};
   default: Y={1'b0, A};
   endcase
   end
endmodule
	module
	one_four_demux_BM(S,A,Y);
	input A;
	wire A;
	input [1:0] S;
	wire [1:0] S;
	output [3:0] Y;
	reg [3:0] Y;
	
	always @(A or S) begin
	
	case(S)
	1'b00: Y= {A,3'b000};
	1'b01: Y= {1'b0, A,2'b00};
	1'b10: Y= {2'b00, A,1'b0};
	default: Y= {3'b000, A};
	endcase
	end
	endmodule
		module
		one_eight_demux_BM(S,A,Y);
		input A;
		wire A;
		input [2:0] S;
		wire [2:0] S;
		output [7:0] Y;
		reg [7:0] Y;
		
		always @(A or S)
		begin
		
		case(S)
		0: Y={A,7'b0000000};
		1: Y= {1'b0, A,6'b000000};
		2: Y= {2'b00, A,5'b00000};
		3: Y= {3'b000, A,4'b0000};
		4: Y= {4'b0000, A,3'b000};
		5: Y= {5'b00000, A,2'b00};
		6: Y= {6'b000000, A,1'b0};
		default: Y= {7'b0000000, A};
		endcase
		end
		endmodule

Verilog Testbench code:

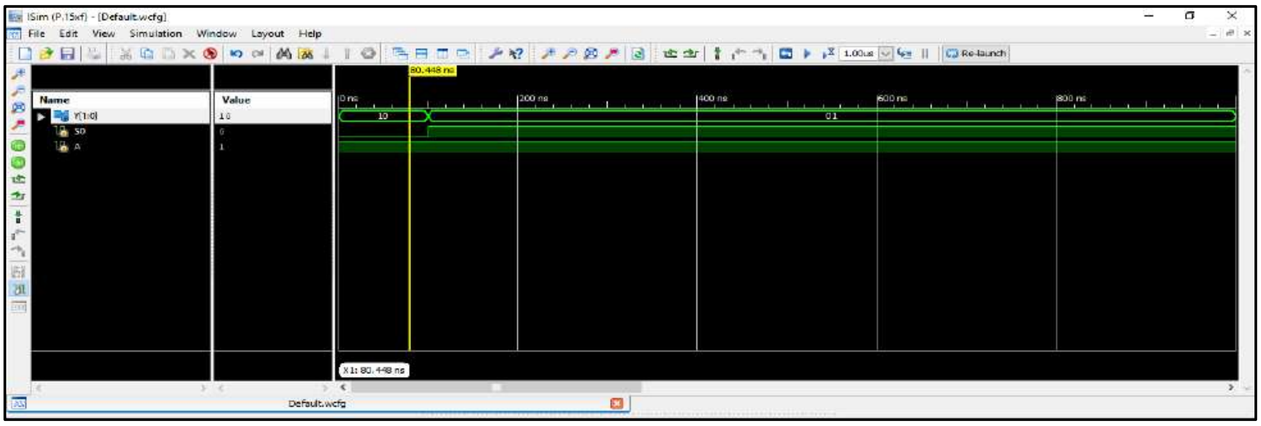
1:2 DEMUX	**1:4 DEMUX**	**1:8 DEMUX**

module demux12_BM_tb;
        reg S0;
        reg A;
        wire [1:0] Y;
  demux12_BM uut (
                .S0(S0),
                .A(A),
                .Y(Y));
	module
	one_four_demux_BMtb;
	        reg [1:0] S;
	        reg A;
	        wire [3:0] Y;
	one_four_demux_BM uut (
	                .S(S),
	                .A(A),
		module
		one_eight_demux_BMtb;
		        reg [2:0] S;
		        reg A;
		        wire [7:0] Y;
		one_eight_demux_BM uut (
		                .S(S),
		                .A(A),

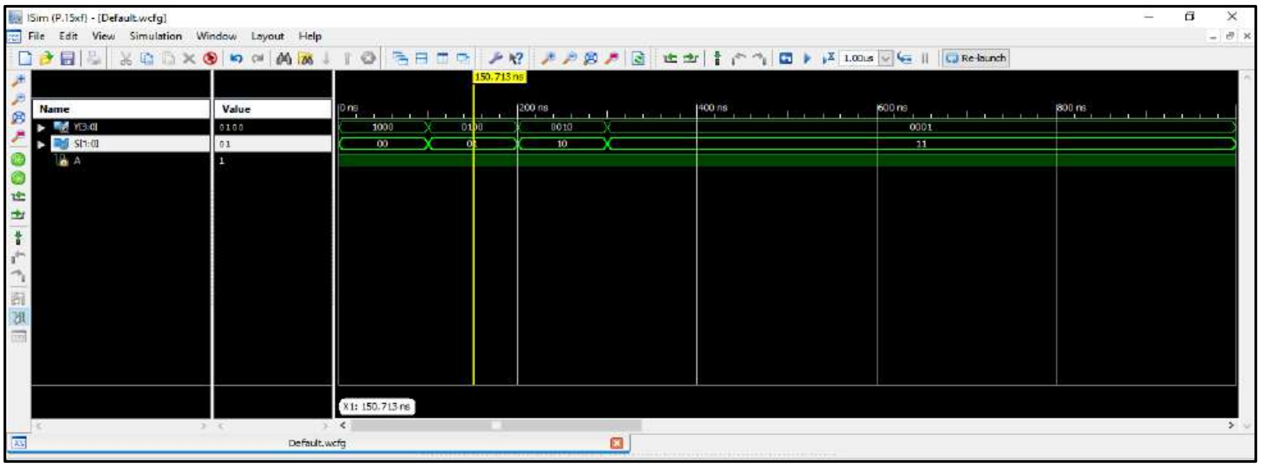
initial begin
S0 = 0; A = 1; #100;
S0 = 1; A = 1; #100;
end
endmodule
	.Y(Y));
	initial begin
	S = 0; A = 1; #100;
	S = 1; A = 1; #100;
	S = 2; A = 1; #100;
	S = 3; A = 1; #100;
	end
	endmodule
		.Y(Y));
		initial begin
		S = 0; A = 1; #100;
		S = 1; A = 1; #100;
		S = 2; A = 1; #100;
		S = 3; A = 1; #100;
		S = 4; A = 1; #100;
		S = 5; A = 1; #100;
		S = 6; A = 1; #100;
		S = 7; A = 1; #100;
		end
		endmodule

Simulation Output:

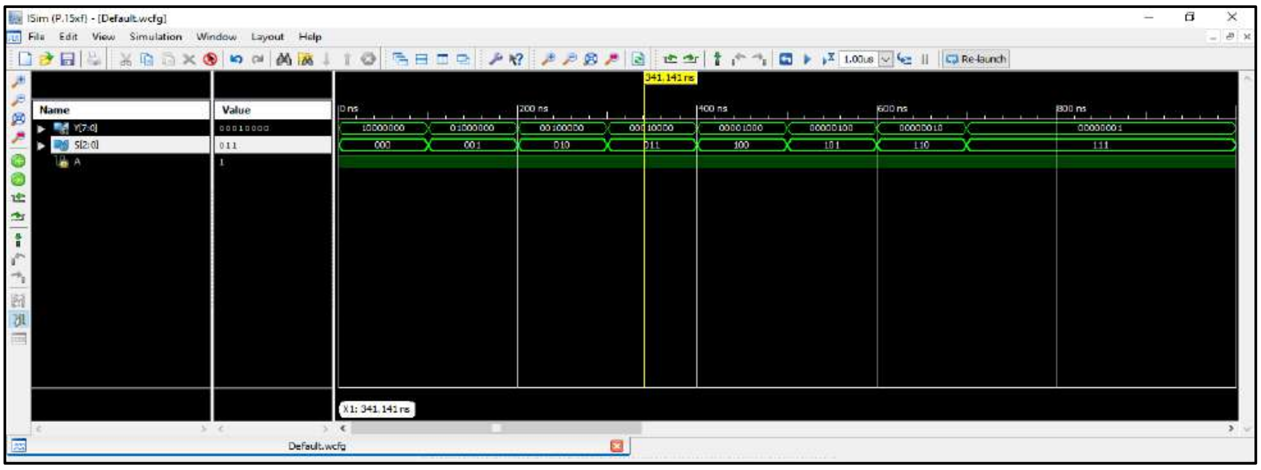
1:2 DEMUX



1:4 DEMUX



1:8 DEMUX



Results & conclusions:

The Verilog code for Different types of De-multiplexer like 1:2, 1:4 and 1:8 is simulated and verified.

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.

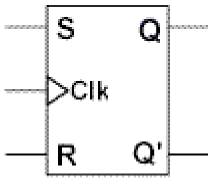
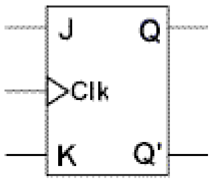
Objectives:

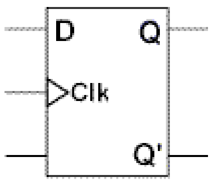
- Implement Verilog modules for SR, JK, and D flip-flops using behavioral modeling. Develop a testbench to simulate the behavior of the implemented flip-flops.
- Apply different input conditions to the flip-flops in the testbench to observe their responses.
- Verify that the flip-flops exhibit the expected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

Theory:

Flip-flops are synchronous bitable devices. The term synchronous means the output changes state only when the clock input is triggered. That is, changes in the output occur in synchronization with the clock. A flip-flop circuit has two outputs, one for the normal value and one for the complement value of the stored bit. Since memory elements in sequential circuits are usually flip-flops, it is worth summarizing the behavior of various flip-flop types before proceeding further. Flip-flops can be divided into four basic types: SR, JK, D and T. They differ in the number of inputs and in the response invoked by different value of input signals. The four types of flip-flops are defined in the Table 8.1. Each of these flip-flops can be uniquely described by its graphical symbol, its characteristic table, its characteristic equation or excitation table. All flip-flops have output signals Q and Q'.

Table 8.1: Flip-flops and their properties

Flip-Flop Name	Flip-Flop Symbol	Characteristic Table			Characteristic Equation	Excitation Table			
SR		**S**	**R**	**Q(next)**	Q(next) = S + R'Q SR = 0	**Q**	**Q(next)**	**S**	**R**
		0	0	Q		0	0	0	X
		0	1	0		0	1	1	0
		1	0	1		1	0	0	1
		1	1	?		1	1	X	0
JK		**J**	**K**	**Q(next)**	Q(next) = JQ' + K'Q	**Q**	**Q(next)**	**J**	**K**
		0	0	Q		0	0	0	X
		0	1	0		0	1	1	X
		1	0	1		1	0	X	1
		1	1	Q'		1	1	X	0

D		D | Q(next) 0 | 0 1 | 1	Q(next) = D	Q | Q(next) | D 0 | 0 | 0 0 | 1 | 1 1 | 0 | 0 1 | 1 | 1

Logic diagram

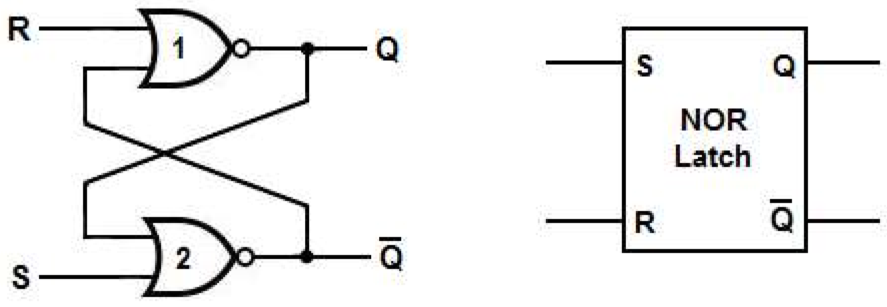
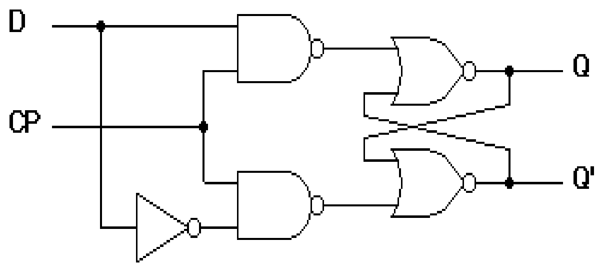


Figure 8.1: SR (a) NOR Gate latch (b) Symbol



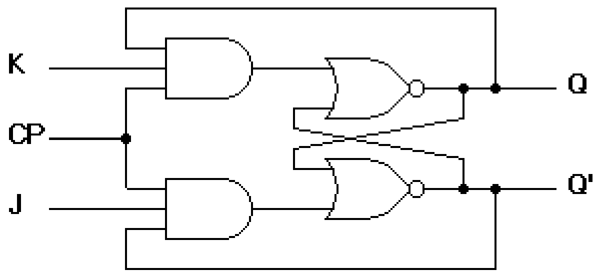


Figure 8.2: D- Flip Flop

Figure 8.3: JK Flip Flop

Graphs

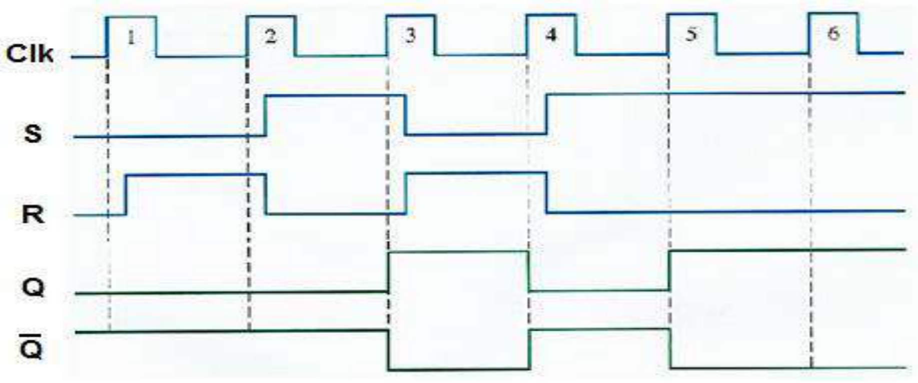


Figure 8.4: Timing diagram of SR Flip-Flop

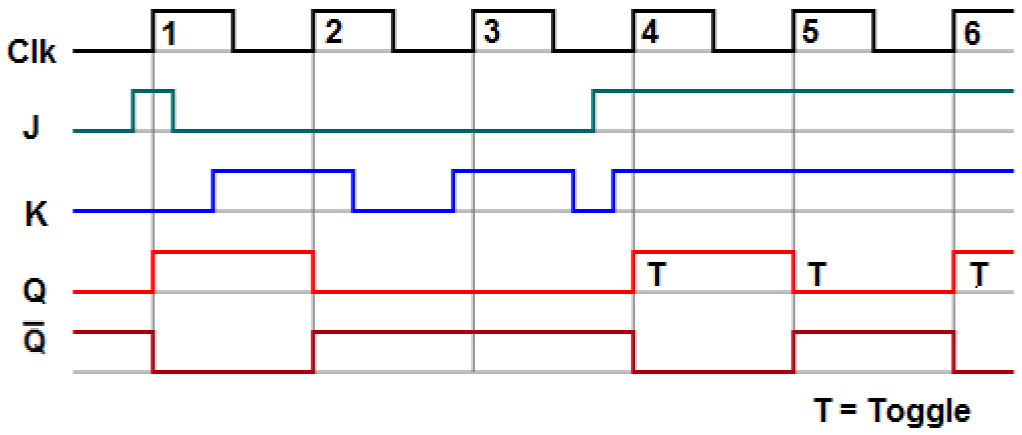


Figure 8.5: Timing diagram of JK Flip-Flop

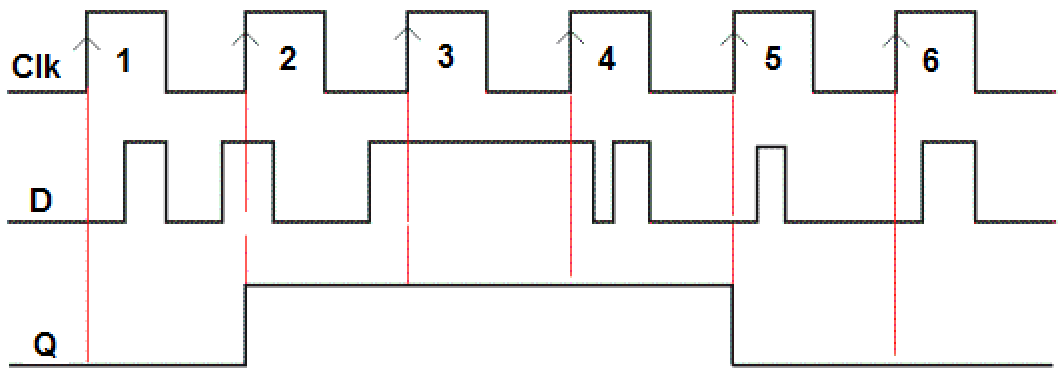


Figure 8.6: Timing diagram of D Flip-Flop

Equipment / components required:

- Computer with Modelsim SoftwareSpecifications:
- HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk
- Softwares: Modelsim - 5.7c, Xilinx - 14.1i.

Procedure:

1. Open a Xilinx IDE, close old projects.
2. Create a new project through the project navigator icon.
3. Add VHDL/Verilog new source to the project depending on the user requirement.
4. Select the Verilog HDL mode.
5. Assign the inputs and outputs for the system to design.
6. And write the code as given below for the functionality.
7. Synthesize the code and correct the syntax errors if any.
8. Isim Simulator and Behavioral check syntax the Simulation behavioral model.
9. Observe the output timing waveform and verify it with the truth table.

Verilog code:

1. SR Flip-Flop

```
module srlax(
   input wire S,
   input wire R,
   input output reg Q,
   output wire Q_bar );
always @(S, R) begin
if (S && ~R)
             Q <= 1'b1;
else if (~S && R)
             Q <= 1'b0;
else if (S==0 && R==0)
             Q <= Q;
else if (S==1 && R==1)
             Q <= 1'bz;
         end
 assign Q_bar = ~Q;
      endmodule
```

2. JK Flip Flop

```
module JK_FF(
  input wire j,
  input wire k,
  output reg Q,
  output wire Q_bar );

  always @(j, k) begin
    if (j && ~k)
      Q <= 1'b1;
    else if (~j && k)
      Q <= 1'b0;
    else if (j==0 && k==0)
      Q <= Q;
         else if (j==1 && k==1)
      Q <= !Q;
  end
  assign Q_bar = ~Q;
```

3. D Flip Flop

```
module D_FF(
  input wire D,
  output reg Q,
  output wire Q_bar
);

  always @(D) begin
     if (D==0)
      Q <= 0;
         else
      Q <= 1;
  end

  assign Q_bar = ~Q;
endmodule
```

```
endmodule
```

Verilog Testbench code:

1. SR Flip-Flop:

```
module srlax_tb;
          reg S;
          reg R;
          wire Q;
          wire Q_bar;
          srlax uut (
          .S(S), .R(R),
          .Q(Q), .Q_bar(Q_bar));
        initial begin
          S = 0; R = 0; #100;
          S = 1; R = 0; #100;
          S = 0; R = 1; #100;
          S = 1; R = 1; #100;
            end
   endmodule
```

2. JK Flip Flop

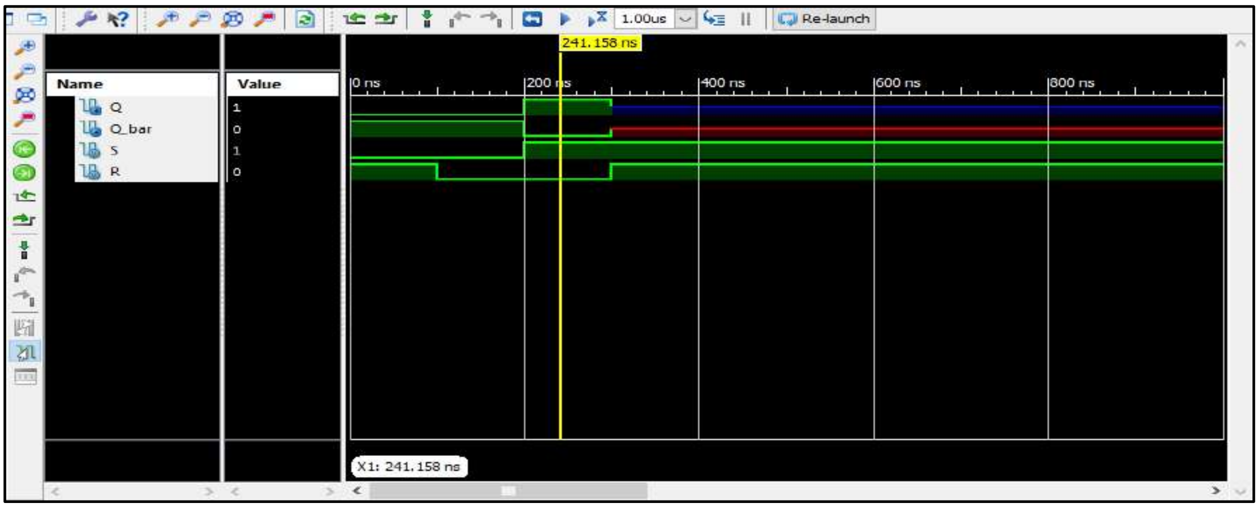
```
module JK_FF_tb;
          reg J;
          reg K;
          wire Q;
          wire Q_bar;
          JK_FF uut (
          .J(J), .K(K),
          .Q(Q), .Q_bar(Q_bar));
          initial begin
           J = 0; K = 0 ;#100;
          J = 1; K = 0; #100;
          J = 0; K = 1 ;#100;
          J = 1; K = 1; #100;
          end
endmodule
```

3. D Flip Flop

```
module D_FF_tb;
          reg D;
          wire Q;
          wire Q_bar;
          D_FF uut (
          .D(D),
          .Q(Q), .Q_bar(Q_bar));
             initial begin
                   D = 0; #100;
                   D = 1; #100;
             end
endmodule
```

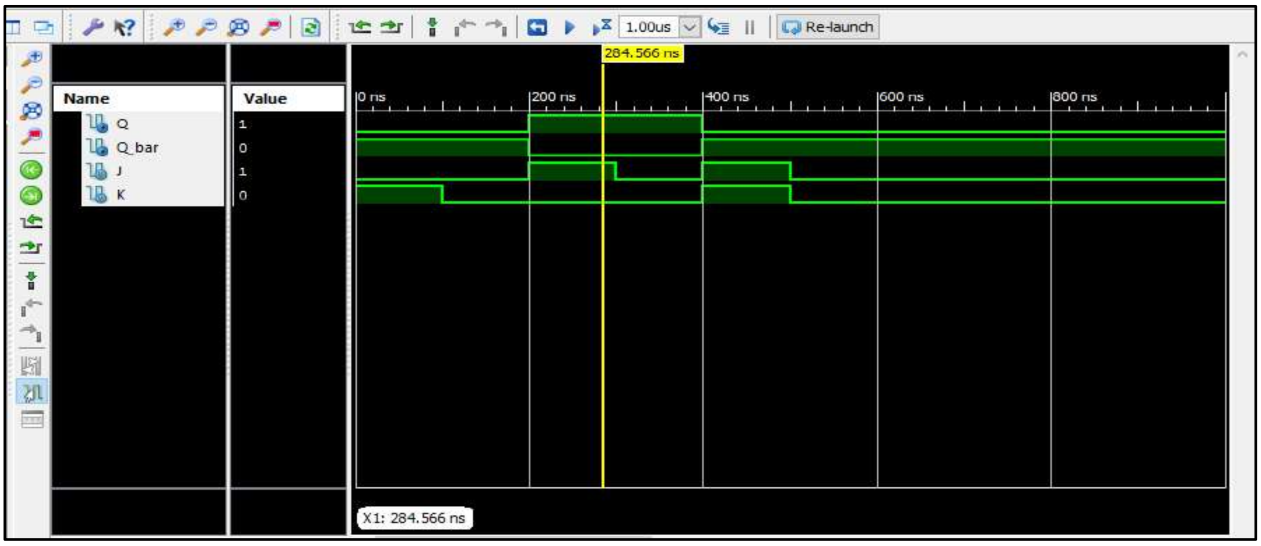
Simulation Output:

1. SR Flip-Flop



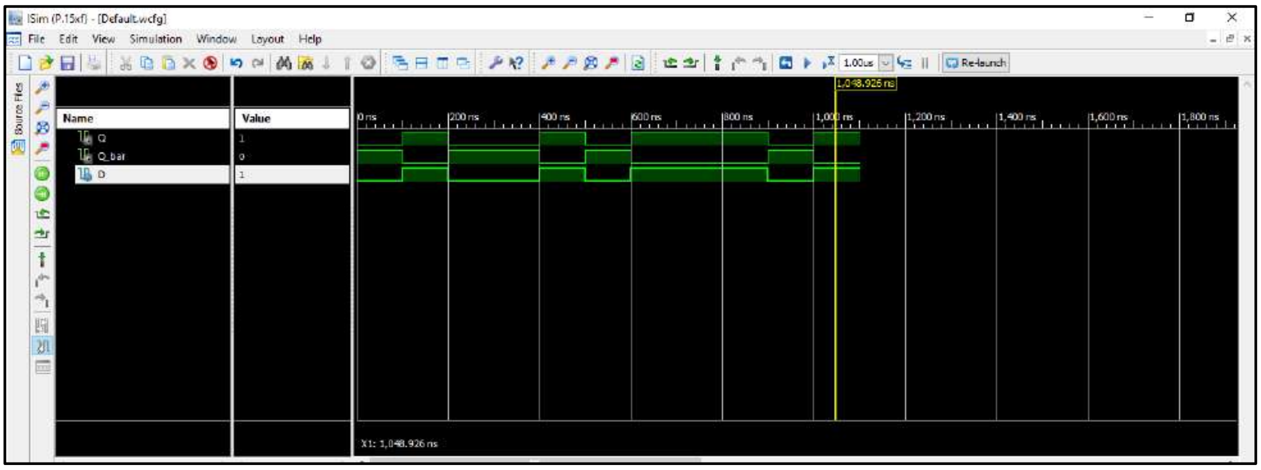
Name	Value
Q	1
Q_bar	0
S	1
R	0

2. JK Flip Flop



Name	Value
Q	1
Q_bar	0
J	1
K	0

3. D Flip Flop



Results & conclusions:

The Verilog code for Different types various types of Flip-Flops such as SR, JK and D.is simulated and verified

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 a versatile 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.