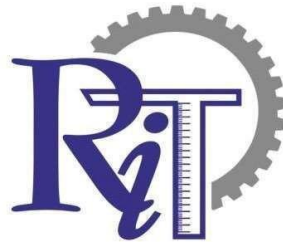


RAJEEV INSTITUTE OF TECHNOLOGY

HASSAN-573201



DIGITAL DESIGN AND COMPUTER ORGANIZATION

(BCS302)

As per VTU Syllabus/scheme for 3rd Semester



DEPARTMENT OF COMPUTER SCIENCE ENGINEERING (CS&E)

VISION & MISSION OF THE INSTITUTE

VISION:

- ❖ To be an academic institution in vibrant social & economic environment, striving continuously for excellence in education, research and technological service to the society

MISSION:

1. To achieve academic excellence in engineering and management through dedication to duty, offering state of the art education and faith in human values
2. To create and endure a community of learning among students, develop outstanding professionals with high ethical standards
3. To provide academic ambience conducive to the development, needs and growth of society and the industry

PROGRAM OUTCOMES

Graduation students of Bachelor of Mechanical Engineering program at Rajeev Institute of Technology will attain the following program outcomes in the field of Mechanical Engineering.

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

PO2- Problem analysis: Identify, formulate, 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 public health and safety, and 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 modeling 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 the 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 environment.

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

COURSE MODULE

Faculty Name/s: Chiranjeevi m.r			Academic Year:2024-2025				
Department: Information Science and Engineering							
Course Code	Course Title	Core/Elective	Prerequisite	Contact Hours			Total Hrs/ Sessions
				L	T	P	
BCS302	Digital Design and Computer Organization	Core		3	0	2	40

Course objectives: This course will enable students to:

CLO1: To demonstrate the functionalities of binary logic system

CLO2: To explain the working of combinational and sequential logic system.

CLO3: To realize the basic structure of computer System.

CLO4 : To illustrate the working of I/O operations and processing unit

Topics Covered as per Syllabus

MODULE-I

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.

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.

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.

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.

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.

PRACTICAL COMPONENT OF IPCC

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

List of Text Books

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

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.

CIE for the theory component of the IPCC (maximum marks 50)

- 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.

- 25 marks for the theory component are split into 15 marks for two Internal Assessment Tests (Two Tests, each of 15 Marks with 01-hour duration, are to be conducted) and 10 marks for other assessment methods mentioned in 22OB4.2. The first test at the end of 40-50% coverage of the syllabus and the second test after covering 85-90% of the syllabus.
- 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 to qualify in the CIE of the theory component of IPCC.

CIE for the practical component of the IPCC

- 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

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.

Course Outcomes: Students 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.

The Correlation of Course Outcomes (CO's) and Program Outcomes (PO's)

Subject Code:	BCS302	TITLE: Digital Design and Computer Organization						Faculty Name:	Chiranjeevi M.R				
Course Outcomes	Program Outcomes												
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	
CO-1	3	3	2	1								3	
CO-2	3	3	2	1								3	
CO-3	2	1	1									2	
CO-4	2	1	1									2	
CO-5	2	1	1									2	

Note: 3 = Strong Contribution 2 = Average Contribution 1 = Weak Contribution - = No Contribution

Course Outcomes	PSO 1	PSO 1
CO1		1
CO-2		1
CO-3		1
CO-4		1
CO-5		1

LIST OF EXPERIMENTS

Sl . No	Experiment	CO	PO
1	Given a 4-variable logic expression, simplify it using appropriate technique and simulate the same using basic gates.	CO1	1,2,3,4,12
2	Design a 4 bit full adder and subtractor and simulate the same using basic gates.	CO1	1,2,3,4,12
3	Design Verilog HDL to implement simple circuits using structural, Data flow and Behavioural model.	CO2	1,2,3,4,12
4	Design Verilog HDL to implement Binary Adder-Subtractor – Half and Full Adder, Half and Full Subtractor.	CO2	1,2,3,4,12
5	Design Verilog HDL to implement Decimal adder.	CO2	1,2,3,4,12
6	Design Verilog program to implement Different types of multiplexer like 2:1, 4:1 and 8:1.	CO2	1,2,3,4,12
7	Design Verilog program to implement types of De-Multiplexer.	CO2	1,2,3,4,12
8	Design Verilog program for implementing various types of Flip-Flops such as SR, JK, T and D.	CO2	1,2,3,4,12

PSpice

INTRODUCTION

In PSpice the program we run in order to draw circuit schematics is called CAPTURE. The program that will let us run simulations and see graphic results is called PSPICE. You can run simulation from the program where your schematic is. There are a lot of things we can do with PSpice, but the most important things for you to learn are

- Design and draw circuits
- Simulate circuits
- Analyze simulation results

PROCEDURE

1. Run the CAPTURE program.
2. Select File/New/Project from the File menu.
3. On the New Project window select Analog or Mixed A/D, and give a name to your project then click OK.
4. The Create PSpice Project window will pop up, select Create a blank project, and then click OK.
5. Now you will be in the schematic environment where you are to build your circuit.
6. Select Place→Part from the Place menu.
7. In search bar type the gate to be selected then click on ok
8. Use the mouse to place the gate where you want and then click to leave the gate there. You can continue placing as many gates as you need and once you have finished placing the gate right-click your mouse and select end mode.
9. To rotate the components there are two options: Rotate a component once it is placed: Select the component by clicking on it then Rotate it or Rotate the component before it is placed (using right click , rotate)
10. Select Place→Wire from the Place menu to establish the connection between input output and gate. Then right-click and select end wire

11. Select Place→Part from the Place menu, search digclock . mention online and offtime.
12. Select Place→net alias from the Place menu to label input and output.
13. Select Pspice from the menu, select new simulation profile , give name for new simulation click on create.
14. In simulation setting window give Run to time and maximum step size.
15. Select Pspice from the menu click on run, a schematic window will be opened.
16. In schematic window select trace add trace, select the trace expression required click on ok. Now graph of designed circuit can be observed.

Gates and their IC numbers

Gates	IC number
Not gate	7404
2 INPUT AND	7408
3 INPUT AND	7411
2 INPUT OR	7432
NAND	7400
NOR	7402
EX-OR	7486

Xlink

INTRODUCTION

ISim provides a complete, full-featured HDL simulator integrated within ISE. HDL simulation now can be an even more fundamental step within your design flow with the tight integration of the ISim within your design environment.

PROCEDURE

Double click on ISE design suit

File→ close project

File →new project→ name of the project→ next(check family Spartn3)→ next→ finish

On hierarchy window→ right click on specification (xc3s50-5pq208)→ new source→select verilog module→ give the file name→ next

Mention inputs and outputs → next→ finish

Write the program and save

To check error

On processes window double click on synthesis to check for error (if any error it will be indicated with the wrong symbol, any warning indicated with exclamatory mark or else it will show green tick mark for no error)

Any error or warning go to error/ warning window correct it save and double click on synthesis.

To write Testbench

Right click on filename.v → new source→VHDL testbench→Give the filename→ next→ Finish

To verify truth table

Go to simulation→sect the testbench created(uut file)

Go to Isim simulator→ double click on behavioral check syntax→ Simulate Behavioral Model

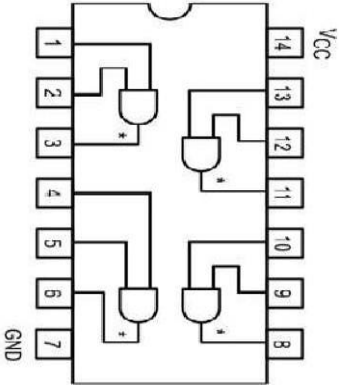

A simulator waveform window will be opened with inputs and outputs

Right click on input → force clock and specify the input timing for all the input and run the waveform→ verify with the truth table.

BASIC GATES

AND (7400):

The AND gate is so named because, if 0 is called "false" and 1 is called "true," the gate acts in the same way as the logical "and" operator. The following illustration and table show the circuit symbol and logic combinations for an AND gate. (In the symbol, the input terminals are at left and the output terminal is at right.) The output is "true" when both inputs are "true." Otherwise, the output is "false." In other words, the output is 1 only when both inputs one AND two are 1.

Pin Diagram(7400)	Symbol	Truth Table		
		Input		Output
		A	B	Y=AB
		0	0	0
		0	1	0
		1	0	0
		1	1	1

3 Input AND (7411):

3 Input AND Pin Diagram(7411)	Symbol	Truth Table			
		Input			Output
		A	B	C	Y=ABC
		0	0	0	0
		0	0	1	0
		0	1	0	0
		0	1	1	0
		1	0	0	0
		1	0	1	0
		1	1	0	0
		1	1	1	1

OR (7432):

The OR gate gets its name from the fact that it behaves after the fashion of the logical inclusive "or." The output is "true" if either or both of the inputs are "true." If both inputs are "false," then the output is "false." In other words, for the output to be 1, at least input one OR two must be 1.

Pin Diagram(7432)	Symbol	Truth Table		
		Input		Output
		A	B	$Y=A+B$
		0	0	0
		0	1	1
		1	0	1
		1	1	1

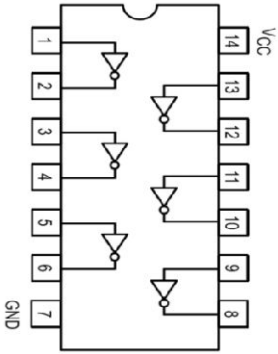
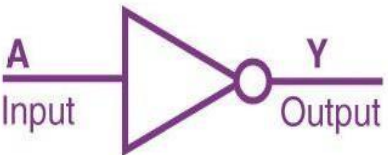
EX-OR (7486):

The XOR (exclusive-OR) gate acts in the same way as the logical "either/or." The output is "true" if either, but not both, of the inputs are "true." The output is "false" if both inputs are "false" or if both inputs are "true." Another way of looking at this circuit is to observe that the output is 1 if the inputs are different, but 0 if the inputs are the same.

Pin Diagram(7486)	Symbol	Truth Table		
		Input		Output
		A	B	$Y=A\oplus B$
		0	0	0
		0	1	1
		1	0	1
		1	1	0

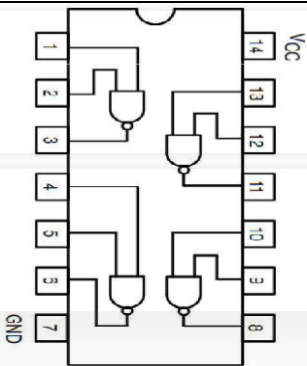
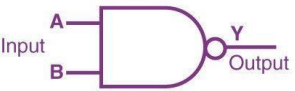
NOT (7404):

A logical inverter, sometimes called a NOT gate to differentiate it from other types of electronic inverter devices, has only one input. It reverses the logic state. If the input is 1, then the output is 0. If the input is 0, then the output is 1.

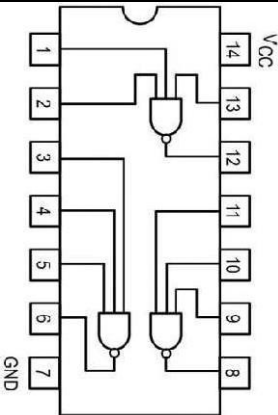
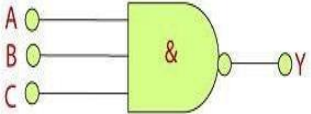
Pin Diagram(7404)	Symbol	Truth Table		
		Input	Output	
		A	Y=	
		0	1	
		1	0	

UNIVERSAL GATES:**NAND (7400):**

The NAND gate operates as an AND gate followed by a NOT gate. It acts in the manner of the logical operation "and" followed by negation. The output is "false" if both inputs are "true." Otherwise, the output is "true."

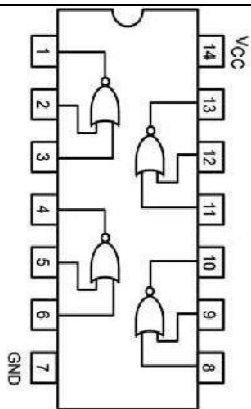
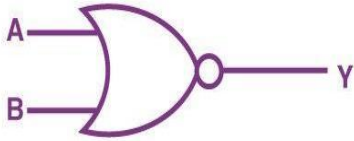
Pin Diagram(7400)	Symbol	Truth Table		
		Input		Output
		A	B	Y=
		0	0	1
		0	1	1
		1	0	1
		1	1	0

3 Inputs NAND (7411):

3 Pin Diagram(7410)	Symbol	Truth Table			
		Input			Output
		A	B	C	Y=
		0	0	0	1
		0	0	1	1
		0	1	0	1
		0	1	1	1
		1	0	0	1
		1	0	1	1
		1	1	0	1
		1	1	1	0

NOR (7402):

The NOR gate is a combination OR gate followed by an inverter. Its output is "true" if both inputs are "false." Otherwise, the output is "false."

Pin Diagram(7402)	Symbol	Truth Table		
		Input		Output
		A	B	Y=
		0	0	1
		0	1	0
		1	0	0
		1	1	0

EXPERIMENT NO 1

GIVEN A 4-VARIABLE LOGIC EXPRESSION, SIMPLIFY IT USING APPROPRIATE TECHNIQUE AND SIMULATE THE SAME USING BASIC GATES.

AIM: To simplify 4-variable logic expression using appropriate technique and implement the same using basic gates.

DESCRIPTION: The most practical use of Boolean algebra is to simplify logic circuits. A Boolean expression can be implemented directly in a logic circuit. The number of terms and operations in a Boolean expression is directly related to the number of logic components. Through Boolean algebra simplification, a Boolean expression is translated to another form with less number of terms and operations. A logic circuit for the simplified Boolean expression performs the identical function with fewer logic components as compared to its original form. Additionally, the simplified Boolean expression when implemented to a logic circuit is reliable with a reduced cost. Boolean expressions may be simplified by applying a series of Boolean algebra laws or K-Map.

Karnaugh's Map: 4 Variables Karnaugh's Map or K-Map is an alternate method to solve or minimize the Boolean expressions based on AND, OR & NOT gates logical expressions or truth tables. The four variables A, B, C & D are the binary numbers which are used to address the min-term SOP of the Boolean expressions. The gray code conversion method is used to address the cells of K-MAP table.

The min-term SOP is often denoted by ABCD, 1s & 0s or decimal numbers. For example, the Boolean expression $F = \sum\{2, 6, 9, 11, 15\}$ represents the place values of the respective cells which have the higher values (binary 1s). The $F = \sum\{2, 6, 9, 11, 15\}$ can also be represented by $F = \sum\{0010, 0110, 1001, 1011, 1111\}$ or $F = \sum\{A'B'C'D, A'B'CD, A'BC'D, A'BCD, ABCD\}$. A is the most significant bit (MSB) and B is the least significant bit (LSB). Each variable A, B, C & D equals to value 1. Similarly, each inverted variable A, B, C & D equals to 0. Any 4 combinations of A, B, C, D, A, B, C & D represents the place values of 0 to 15 to address the cells of table in KMAP solver.

$$1. F(A, B, C, D) = \sum m(2, 6, 9, 11, 15) = A'B'C'D + A'B'CD + A'BC'D + A'BCD + ABCD$$

The decimal notation for the above given equation is

$$F(A, B, C, D) = \sum(2, 6, 9, 11, 15)$$

Variable Representation Table:

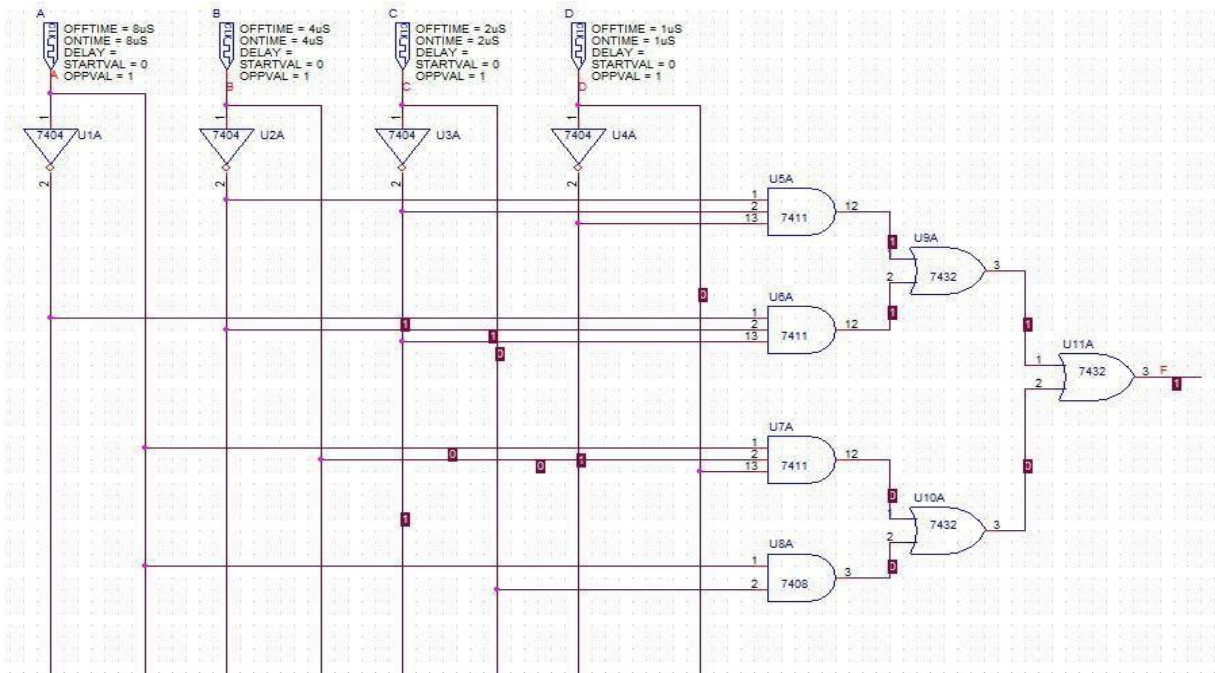
VARIABLES				REPRESENTATION	DECIMAL NOTATION
A	B	C	D		NUMBER
0	0	0	0	0000	0
0	0	0	1	0001	1
0	0	1	0	0010	2
0	0	1	1	0011	3
0	1	0	0	0100	4
0	1	0	1	0101	5
0	1	1	0	0110	6
0	1	1	1	0111	7
1	0	0	0	1000	8
1	0	0	1	1001	9
1	0	1	0	1010	10
1	0	1	1	1011	11
1	1	0	0	1100	12
1	1	0	1	1101	13
1	1	1	0	1110	14
1	1	1	1	1111	15

Simplification using Karnaugh's Map:

	CD	00	01	11	10
AB	00	1	1	0	0
	01	0	0	1	1
	11	0	1	1	1
	10	1	0	0	0

$$\Sigma m(0, 1, 2, 3, 5, 6, 7, 10, 11, 12, 13, 14) = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}CD + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}BCD + A\overline{B}\overline{C}\overline{D} + A\overline{B}\overline{C}D + AB\overline{C}\overline{D} + AB\overline{C}D + ABC\overline{D} + ABCD$$

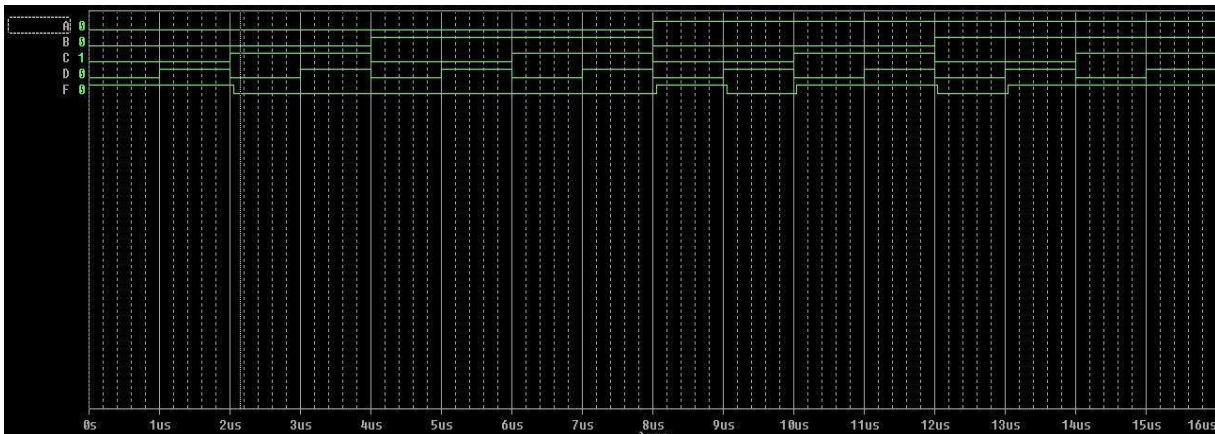
Design using basic gates:



TRUTH TABLE:

Decimal	INPUT				OUTPUT
	A	B	C	D	$F = ABC + ABC + AC + AB$
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

OUTPUT



RESULT

4-variable logic expression is simplified using appropriate technique and is implement using basic gates.

EXPERIMENT NO 2

DESIGN A 4 BIT FULL ADDER AND SUBTRACTOR AND SIMULATE THE SAME USING BASIC GATES.

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

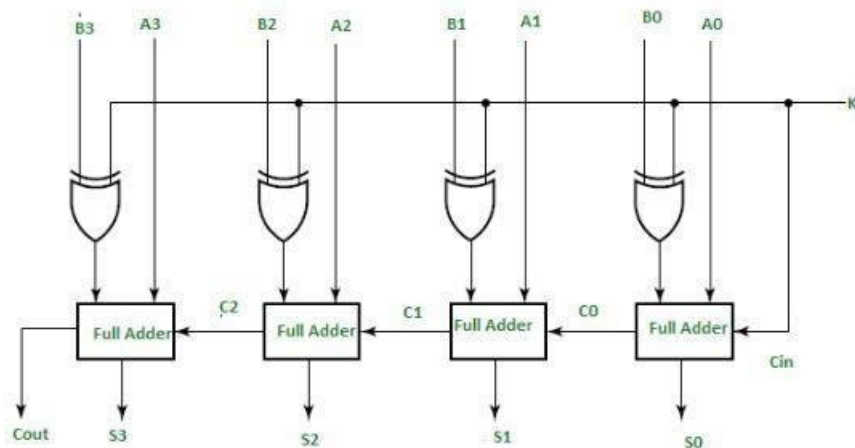
Components Required: Logic gates IC-7486, 7408, 7432, or 7483, Digital IC trainer kit, Patch Cords.

Description: In Digital Circuits, A Binary Adder-Subtractor is capable of both the addition and subtraction of binary numbers in one circuit itself. The operation is performed depending on the binary value the control signal holds. It is one of the components of the ALU (Arithmetic Logic Unit). This Circuit requires prerequisite knowledge of Ex-or Gate, Binary Addition and Subtraction, and Full Adder. Let's consider two 4-bit binary numbers A and B as inputs to the Digital Circuit for the operation with digits

A0 A1 A2 A3 for A

B0 B1 B2 B3 for B

The circuit consists of 4 full adders since we are performing operations on 4-bit numbers. There is a control line K that holds a binary value of either 0 or 1 which determines that the operation is carried out is addition or subtraction.



As shown in the figure, the first full adder has a control line directly as its input(input carry Cin). The input A0 (The least significant bit of A) is directly input in the full adder. The third input is the ex-or of B0 and K. The two outputs produced are Sum/Difference (S0) and Carry (C0). If the value of K (Control line) is 1, the output of B0(ex-or)K=B0'(Complement B0). Thus the operation

would be $A+(B0')$. Now 2's complement subtraction for two numbers A and B is given by $A+B'+C_{in}$. This suggests that when $K=1$, the operation being performed on the four-bit numbers is subtraction. Similarly If the Value of $K=0$, $B0$ (ex-or) $K=B0$. The operation is $A+B$ which is simple binary addition. This suggests that When $K=0$, the operation is performed on the four-bit numbers in addition.

Then C_0 is serially passed to the second full adder as one of its outputs. The sum/difference S_0 is recorded as the least significant bit of the sum/difference. A_1, A_2, A_3 are direct inputs to the second, third and fourth full adders. Then the third input is the B_1, B_2, B_3 EX-ORed with K to the second, third and fourth full adder respectively. The carry C_1, C_2 are serially passed to the successive full adder as one of the inputs. C_3 becomes the total carry to the sum/difference. S_1, S_2, S_3 are recorded to form the result with S_0 .

4 bit Full adder/Subtractor:

$K=0$ Adder

$K=1$ Subtractor

Example for Addition

$K=0, C_{in}=0$

$A=0101$

$B=1010$

Sum=1111

$C_{out}=0$

$A=1100$

$B=1000$

Sum=0100

$C_{out}=1$

Example for subtraction

$K=1, C_{in}=1$

$A=0101$

$B=1010$

1's complement of $B=0101$

2's complement of $B=0101+1=0110$

Difference = $A + 2$'s complement of $B = 1011$

Borrow = 0

$A = 1100$

$B = 1000$

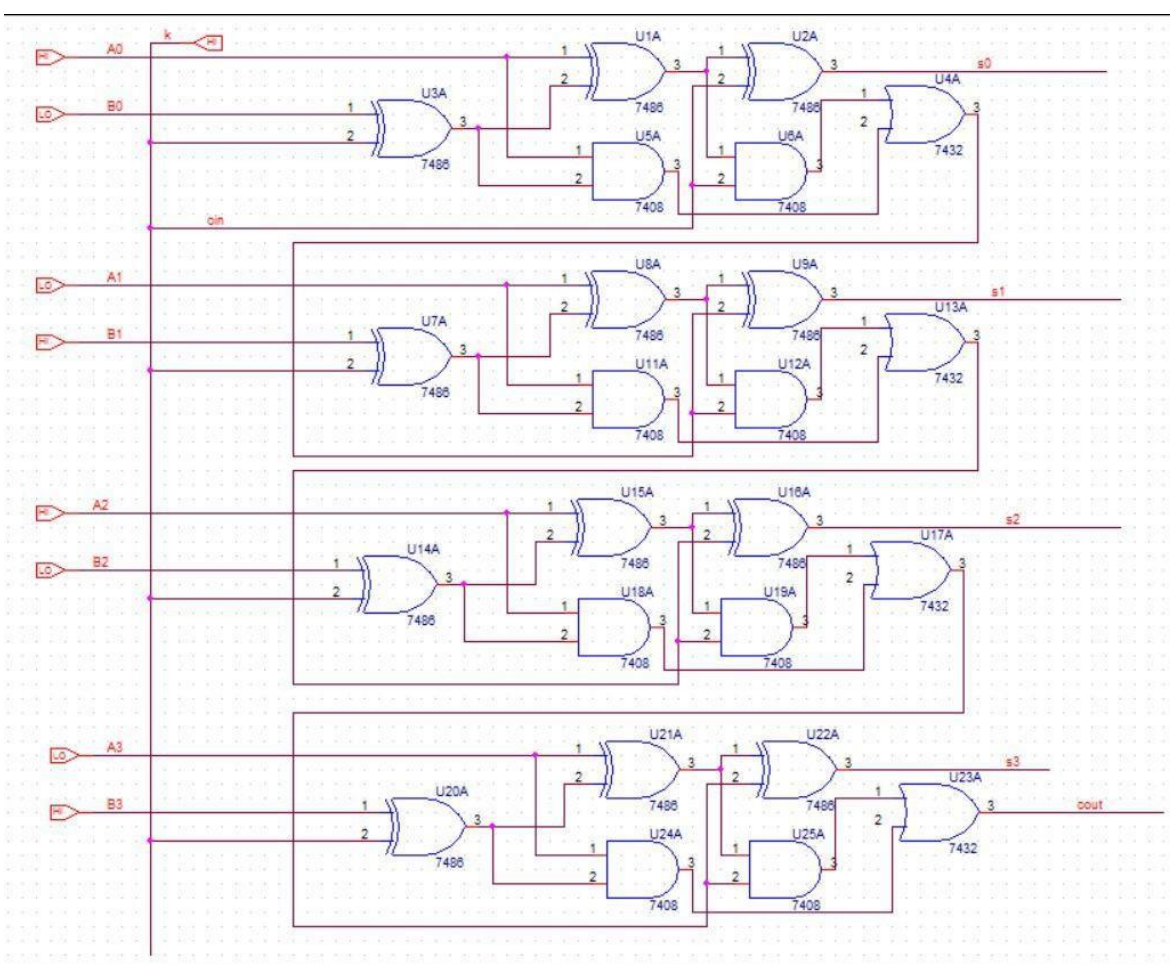
1's complement of $B = 0111$

2's complement of $B = 0111 + 1 = 1000$

Difference = $A + 2$'s complement of $B = 0100$

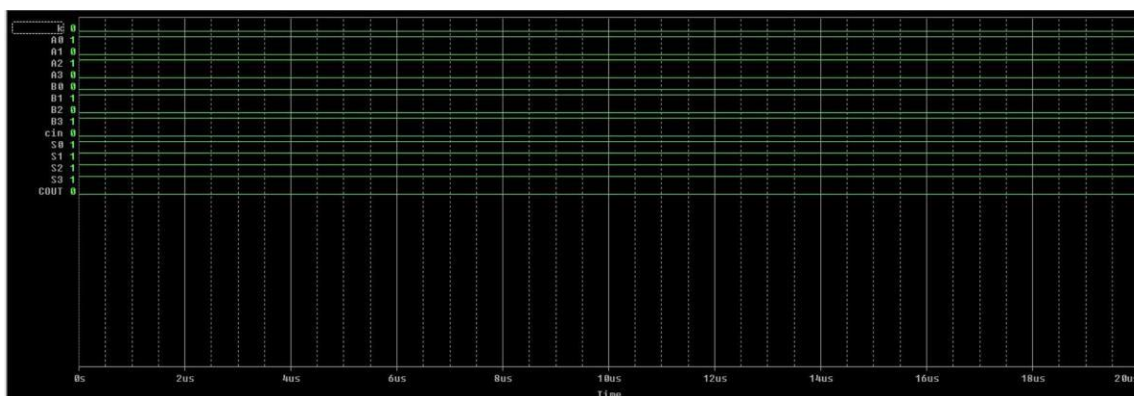
Borrow = 1

Design using basic gates:



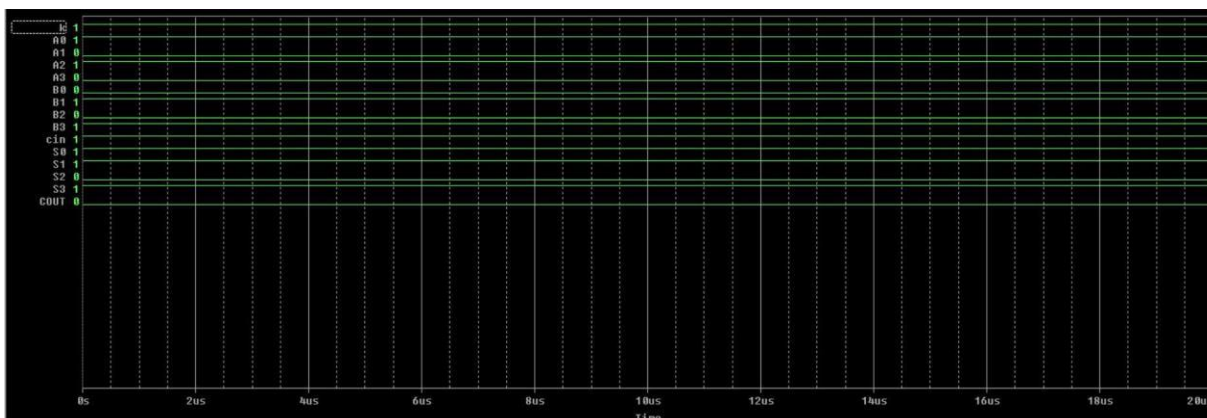
4 bit Full Adder Output Screen:

K=0



4 bit Full Subtractor Output Screen:

K=1



Result: 4 bit full adder and subtractor is designed and simulated the same using basic gates sum and difference are verified using Suitable Simulator Software

EXPERIMENT NO 3

DESIGN VERILOG HDL TO IMPLEMENT SIMPLE CIRCUITS USING STRUCTURAL, DATA FLOW AND BEHAVIOURAL MODEL

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

Description:

Structural:

This is the basic level of modeling in terms of logic gates and the connections between these gates. Most digital designs are now done at the gate level or higher levels of abstractions. At gate level, the circuit is described in terms of gates say AND, OR etc. Hardware design at this level is intuitive for a user who is familiar with the basic knowledge of Digital logic Design. This allows the user to see a direct correspondence between the Verilog Description and the Circuit Diagram.

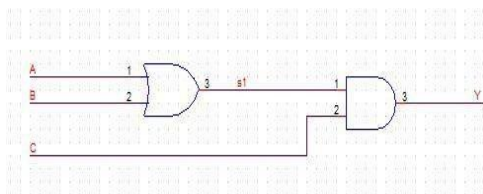
Data Flow:

The design at this level specifies how the data flows between the hardware registers and how the data is processed. For small circuits the gate level modeling works well as the number of gates is limited. However, in complex designs the designers may have to concentrate on implementing the function than bother about the gates. Verilog allows a circuit to be designed in terms of the data flow between registers and how a design processes data rather than instantiation of gates using expressions (=), operators like (&, |, ?) etc.. and continuous assignments(the assign statement).

Behavioural Model:

This is the highest level of abstraction provided by Verilog. The design at this level is similar to an algorithm. This design is very similar to „C“ programming. A module can be implemented in terms of the desired design algorithm without looking into the hardware details using structured procedures (like always and initial), conditional statements (like if and else) and multi way branching.

Circuit:



Truth table

Input			Output
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Verilog HDL Program for structural Modelling

```
module simple_circuit (  
input A,  
input B,  
input C,  
output Y,  
);  
wire s1;  
or o1(s1,A,B);  
and a1(Y,s1,C)  
endmodule
```

Input Value for Force Clock

Signal Name	A	B
Value Radix	Binary	Binary
Leading Edge Value	0	0
Trailing Edge Value	1	1
Starting At Time Offset	0	0
Cancel After Time Offset	4ps	4ps
Duty Cycle	50	50
Period	4	2

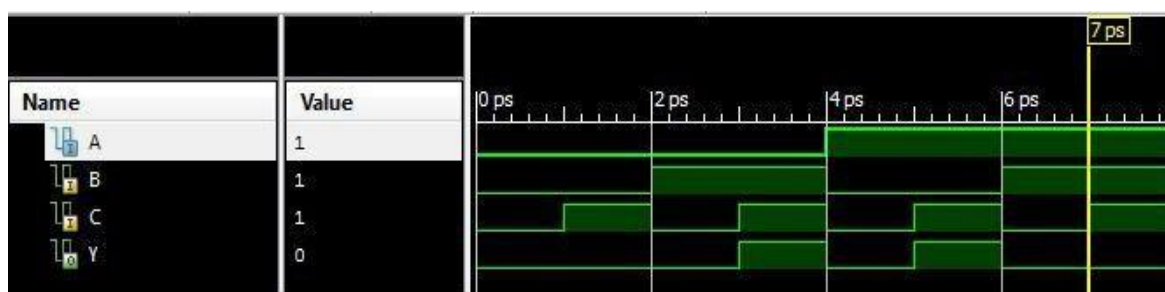
Output Screen



Verilog HDL Program for Data flow Modelling

```
module simple_circuit (
input A,
input B,
input C,
output Y,);
assign Y=((A|B)&C);
endmodule
```

Output Screen

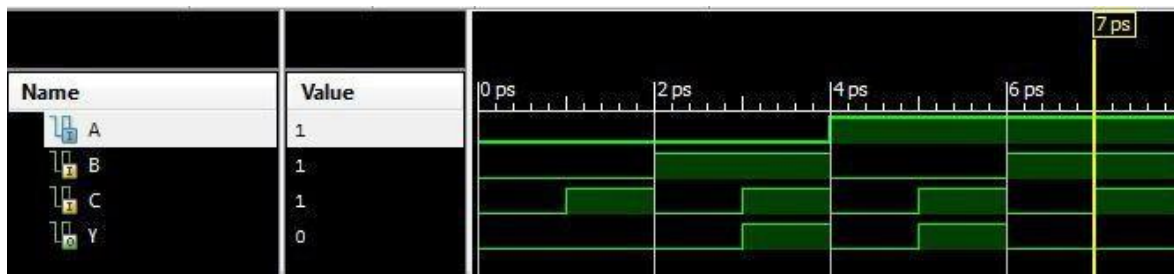


Verilog HDL Program for Behavioural Modelling

```
module simple_circuit(
input A,
input B,
input C,
```

```
output reg Y
);
always@(*)
begin
case({A,B})
3'b000:Y=0;
3'b001:Y=0;
3'b010:Y=0;
3'b011:Y=1;
3'b100:Y=0;
3'b101:Y=1;
3'b110:Y=0;
3'b111:Y=1;
default:Y=0;
endcase
end
endmodule
```

Output Screen



Result: Verilog HDL to implement simple circuits using structural, Data flow and Behavioural model is designed and simulated using Verilog HDL Simulator Software.

EXPERIMENT NO 4

DESIGN VERILOG HDL TO IMPLEMENT BINARY ADDER-SUBTRACTOR – HALF AND FULL ADDER, HALF AND FULL SUBTRACTOR

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

Software requirement: ISE Design Suite 14.7(Xlink), Isim Simulator.

Description

Adder:

A Binary Adder is one kind of digital circuit mainly used for executing the arithmetic operation of two binary numbers like addition. The binary adder can be designed with full adder circuits by connecting them in series. The output carry of a first full adder is connected to the input of the second full adder. These circuits are categorized into a half adder, full adder & parallel adders.

- **Half Adder**

A half adder is one kind of electronic circuit used to perform the addition of two binary numbers. The half adder adds two binary digits and generates two outputs like the output and carries value. The inputs of the half adder are A & B whereas the outputs are the sum and carry. The general representation utilizes logic gates like an AND gate & an XOR logic gate.

- **Full Adder**

A full adder is one kind of electronic circuit used to perform the addition of three binary numbers. The full adder adds three binary digits and generates two outputs like the output and carries value. The inputs of the half adder are A, B, and Cin whereas the outputs are sum and Cout. A full adder is the combination of two half adders where the logic gates like an AND & XOR gates are connected through an OR gate.

Subtractor:

Subtraction is an arithmetical function where one digit is subtracted from another digit to attain equal quantity. The digit from which another digit is to be subtracted is known as minuend. Similarly, the number which is subtracted from the minuend is known as a subtrahend. Same as to the binary addition, this also includes 4- feasible alternative operations where each subtrahend bit can be subtracted from the minuend bit.

However in the 2nd rule, the bit of minuend is lesser compared with the bit of subtrahend, therefore 1 is on loan to complete the subtraction. Related to the adder circuits, these circuits are also categorized like half subtractor, full subtractor & parallel subtractor.

- **Half Subtractor**

The combinational logic circuit like half subtractor is used to subtract two single bit digits. It includes two inputs as well as two outputs. The inputs are A, B whereas the outputs are borrow and difference.

- **Full Subtractor**

The combinational logic circuit like half subtractor is used to subtract two single bit digits. It includes three inputs as well as two outputs. The inputs are A, B and Bin whereas the outputs are Borrow & Difference. Please refer to this link to know more about the Full subtractor. Therefore, this subtractor includes the capability to execute the three bits subtraction by taking into consideration of the borrow in the lower significant stage.

Half Adder:

- **Truth table**

Input		Output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

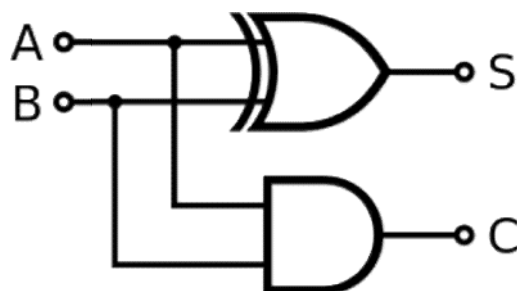
- **Karnaugh map (K-map)**

S	B	0	1
		0	1
A	0	0	1
	1	1	0

$$S = \overline{A}\overline{B} + A\overline{B} + A\overline{B} = \overline{A} \oplus B$$

C	B	0	1
		0	0
A	0	0	0
	1	0	1

$$C = AB$$

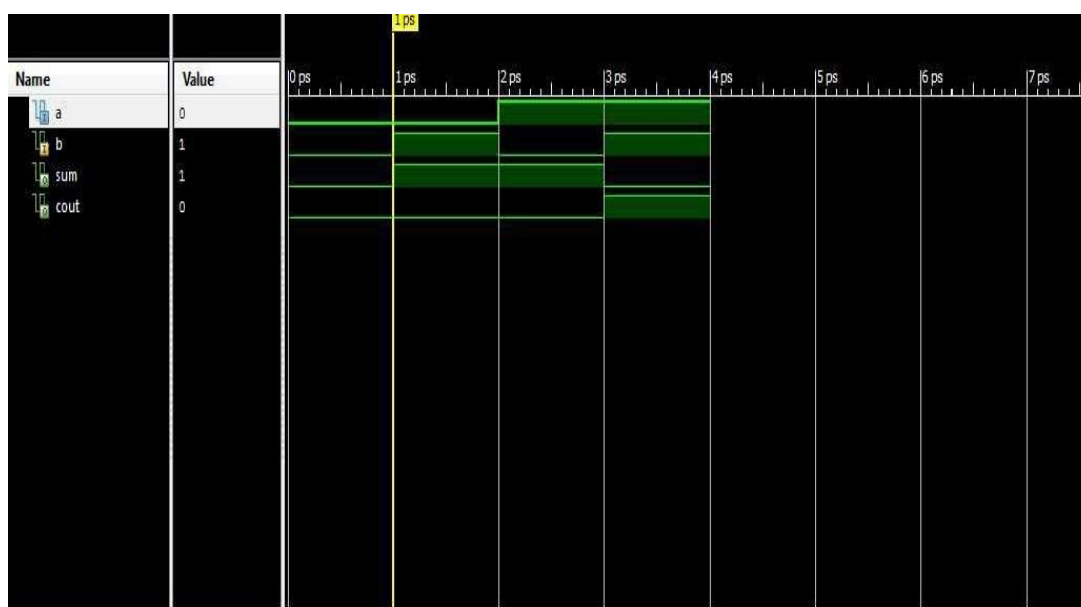
Circuit diagram**Verilog HDL Program**

```
module half_adder(  
input A,  
input B,  
output S,  
output C  
);  
assign S = A ^ B;  
    assign C = A & B;  
endmodule
```

Input Value for Force Clock

Signal Name	A	B
Value Radix	Binary	Binary
Leading Edge Value	0	0
Trailing Edge Value	1	1
Starting At Time Offset	0	0
Cancel After Time Offset	4ps	4ps
Duty Cycle	50	50
Period	4	2

Output Screen



Full Adder

Truth table

Input			Output	
	B	Cin	Sum	Cout
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

Karnaugh map (K-map)

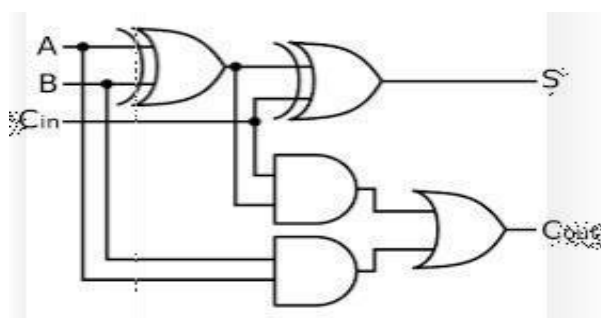
		BCin			
		00	01	11	10
Sum	A				
		00	01	11	10
0		0	1	0	1
1		1	0	1	0

$$\begin{aligned}
 & \overline{A}B\overline{C} + A\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}C + \overline{A}BC + A\overline{B}C + ABC \\
 &= \overline{A}B\overline{C} + A\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}C + \overline{A}BC + A\overline{B}C + ABC \\
 &= \overline{A}B\overline{C} \oplus \overline{A}\overline{B}C \oplus \overline{A}B\overline{C} \oplus \overline{A}\overline{B}C \oplus A\overline{B}C \oplus \overline{A}BC \oplus A\overline{B}C \oplus ABC \\
 &= \overline{A}B\overline{C} \oplus \overline{A}\overline{B}C \oplus A\overline{B}C \oplus ABC
 \end{aligned}$$

		BCin			
A	00	01	11	10	
	0	0	1	0	
1	0	1	1	1	

$Cout = A \cdot B + A \cdot C + B \cdot C$

Circuit diagram



Verilog HDL Program

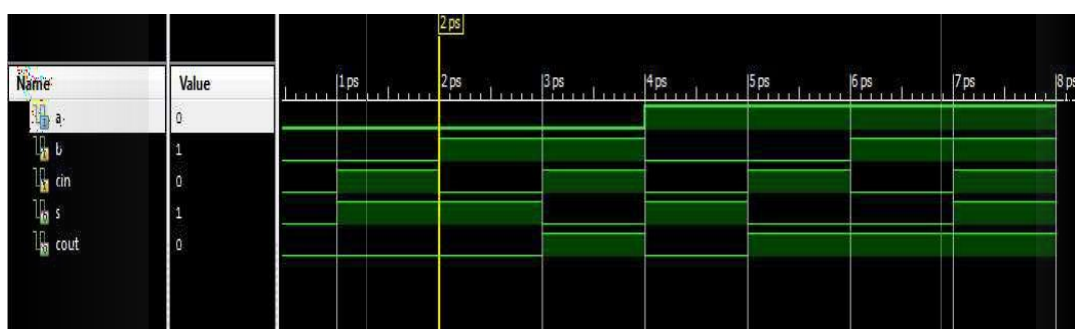
```

module full_adder (
input A,
input B,
input Cin,
output Sum,
output Cout
);
assign Sum = A ^ B ^ Cin;
assign C = (A & B) | (B & Cin) | (A & Cin);
endmodule

```

Input Value for Force Clock.

Signal Name	A	B	in
Value Radix	Binary	Binary	Binary
Leading Edge Value	0	0	0
Trailing Edge Value	1	1	1
Starting At Time Offset	0	0	0
Cancel After Time Offset	8ps	8ps	8ps
Duty Cycle	50	50	50
Period	8	4	2

Output Screen**Half Subtractor****Truth table**

Input		Output	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Karnaugh map (K-map)

Difference

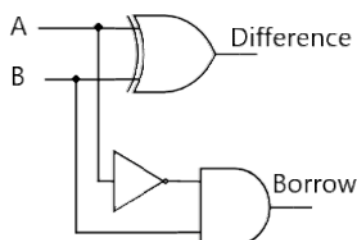
		B	
		0	1
A	0	0	1
	1	1	0

$$\sum m(1, 2) = 10 + 01 = 1 \oplus 0$$

		Borrow	
		A	B
	0	0	1
0	0	0	1
1	0	0	0

• 0000 = 0

Circuit diagram



Verilog HDL Program

```

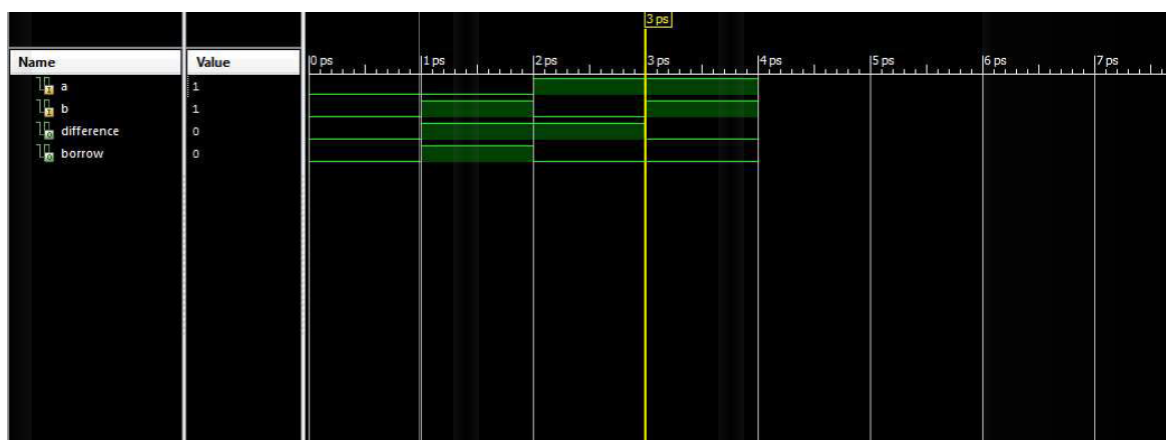
module half_subtractor(
input A,
input B,
output Difference,
output Borrow );
assign Difference = A ^ B;
assign C = ~A & B;
endmodule

```

Input Value for Force Clock.

Signal Name	A	B
Value Radix	Binary	Binary
Leading Edge Value	0	0
Trailing Edge Value	1	1
Starting At Time Offset	0	0
Cancel After Time Offset	4ps	4ps
Duty Cycle	50	50
Period	4	2

Output Screen



Full Subtractor

Truth table

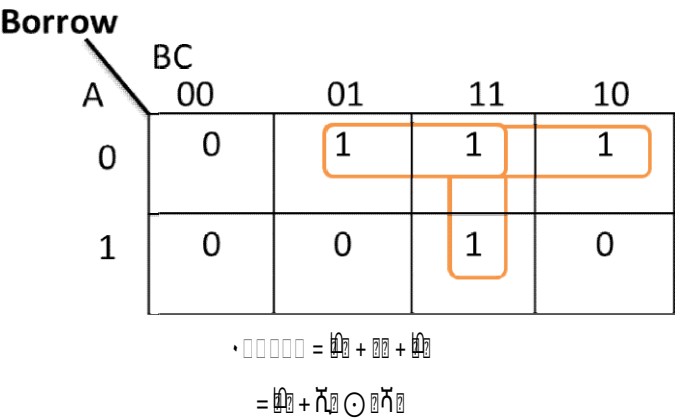
Input			Output	
A	B	C	Difference	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Karnaugh map (K-map)

Difference

		BC			
A		00	01	11	10
	0	0	1	0	1
	1	1	0	1	0

$$\begin{aligned}
 \Sigma m(1,2,3,5) &= m_1 + m_2 + m_3 + m_5 \\
 &= \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C} \\
 &= \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C} \\
 &= \bar{A}(\bar{B}C + B\bar{C}) + A(\bar{B}\bar{C} + B\bar{C}) \\
 &= \bar{A}(C \oplus \bar{C}) + A(\bar{B} \oplus B) \\
 &= \bar{A} \oplus A
 \end{aligned}$$



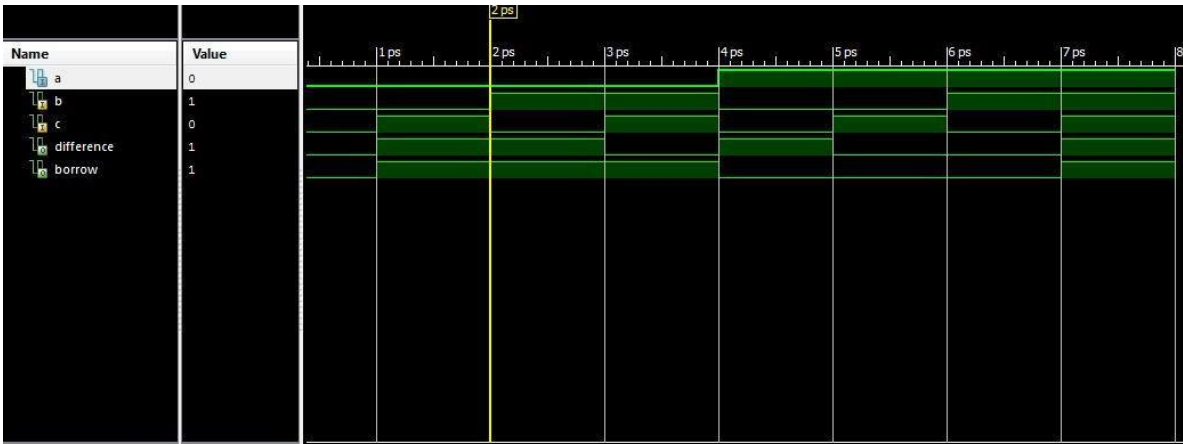
Verilog HDL Program

```
modulefull_subtractor(  
input A,  
input B,  
input C,  
output Difference,  
output Borrow  
);  
assign Difference = A ^ B ^ C;  
assign Borrow =( ~A & B) | (~(A ^ B) & C);  
endmodule
```

Input Value for Force Clock

Signal Name	A	B	C
Value Radix	Binary	Binary	Binary
Leading Edge Value	0	0	0
Trailing Edge Value	1	1	1
Starting At Time Offset	0	0	0
Cancel After Time Offset	8ps	8ps	8ps
Duty Cycle	50	50	50
Period	8	4	2

Output Screen



Result:Design Verilog HDL to implement Binary Adder-Subtractor, Half and Full Adder, Half and Full Subtractor truth table are verified using Verilog HDL Simulator Software.

EXPERIMENT NO 5

DESIGN VERILOG HDL TO IMPLEMENT DECIMAL ADDER

AIM: Design Verilog HDL to implement Decimal adder.

Software requirement: ISE Design Suite 14.7(Xlink), Isim Simulator.

Description

Decimal Adder The digital systems handles the decimal number in the form of binary coded decimal numbers (BCD). A BCD adder is a circuit that adds two BCD digits and produces a sum digit also in BCD. BCD numbers use 10 digits, 0 to 9 which are represented in the binary form 00 0 0 to 1 0 0 1, i.e. each BCD digit is represented as a 4-bit binary number.

The addition of two BCD numbers can be best understood by considering the three cases that occur when two BCD digits are added.

- **Sum Equals 9 or less**

	BCD for 6
6	0 1 1 0
	BCD for 3
+ 3	0 0 1 1
<hr/>	
9	1 0 0 1
	BCD for 9 and is Valid

The addition is carried out as in normal binary addition and the sum is 1 0 0 1, which is BCD code for 9.

- **Sum greater than 9**

The sum 1 1 1 0 is an invalid BCD number. This has occurred because the sum of the two digits exceeds 9. Whenever this occurs the sum has to be corrected by the addition of six (0110) in the invalid BCD number, as shown below

	BCD for 8
8	1 0 0 0
	BCD for 9
+ 9	1 0 0 1
<hr/>	
17	1 0 0 0 1
	BCD for 17 and is Invalid

After addition of 6 carry is produced into the second decimal position. Sum equals 9 or less with carry 1

```

      8    1 0 0 0   BCD for 8
+     9    1 0 0 1   BCD for 9
-----
    17    1 0 0 0 1   BCD for 17 and isInvalid

      0 1 1 0   ADD 6 BCD
-----
    1 0 1 1 1   Valid BCD 7 and Carry, BCD of 17

```

Truth Table

Binary Sum					S A M E C O D E	BCD Sum					Decimal
S'4	S'3	S'2	S'1	S'0		Carry	S3	S2	S1	S0	
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
After 9 BCD and Binary Sum are not Same											
0	1	0	1	0	N O T S A M E C O D E	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

Verilog HDL Program:

```

module decimal_adder(a,b,carry_in,sum,carry);
input [3:0] a,b;
input carry_in;
output [3:0] sum;
output carry;
reg [4:0] sum_temp;

```



```
reg [3:0] sum;
reg carry;
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
```

Output Screen

Name		Value	0 ps
▶ a[3:0]		1000	1000
▶ b[3:0]		1001	1001
▶ carry_in		0	
▶ sum[3:0]		0111	0111
▶ carry		1	
▶ sum_temp[4:0]		10111	10111

Result: Verilog HDL to implement Decimal adder is designed and truth table is verified using Verilog HDL Simulator Software.

EXPERIMENT NO 6

DESIGN VERILOG PROGRAM TO IMPLEMENT DIFFERENT TYPES OF MULTIPLEXER LIKE 2:1, 4:1 AND 8:1

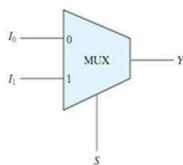
AIM: Design Verilog program to implement Different types of multiplexer like 2:1, 4:1 and 8:1.

Software requirement: ISE Design Suite 14.7(Xlink), Isim Simulator.

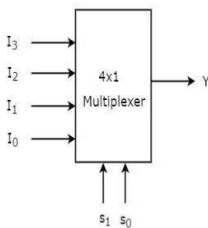
Description:

A multiplexer is a combinational circuit that has 2^n input lines and a single output line. Simply, the multiplexer is a multi-input and single-output combinational circuit. The binary information is received from the input lines and directed to the output line. On the basis of the values of the selection lines, one of these data inputs will be connected to the output.

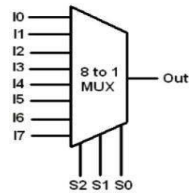
2:1 Multiplexer: In 2:1 multiplexer, there are only two inputs, i.e., I_0 and I_1 , 1 selection line, i.e., S and single outputs, i.e., Y . On the basis of the combination of inputs which are present at the selection line S^0 , one of these 2 inputs will be connected to the output.



4:1 Multiplexer: In the 4×1 multiplexer, there is a total of four inputs, i.e., I_0, I_1, I_2 and I_3 , 2 selection lines, i.e., S_0 and S_1 and single output, i.e., Y . On the basis of the combination of inputs that are present at the selection lines S^0 and S_1 , one of these 4 inputs are connected to the output.



8:1 Multiplexer: In the 8 to 1 multiplexer, there are total eight inputs, i.e., $I_0, I_1, I_2, I_3, I_4, I_5, I_6$, and I_7 , 3 selection lines, i.e., S_0, S_1 and S_2 and single output, i.e., $Y(out)$. On the basis of the combination of inputs that are present at the selection lines S_0, S_1 and S_2 , one of these 8 inputs are connected to the output.



2:1 Multiplexer

- Truth Table

Select I/P	Input		Output
S	I ₁	I ₀	Y
0	I ₁	I ₀	I ₀
1	I ₁	I ₀	I ₁

Verilog HDL Program

```
module mux_2to1(I1,I0,S,Y);
input I1,I0,S;
output Y;
reg Y;
always @ (S, I0 , I1)
begin
if (S == 0)
begin
Y = I0;
end
else
begin
Y = I1 ;
end
end
endmodule
```

Input Value for Force Clock

Signal Name	S	I0& I1
Value Radix	Binary	Binary
Leading Edge Value	0	0
Trailing Edge Value	1	1
Starting At Time Offset	0	0
Cancel After Time Offset	4ps	4ps
Duty Cycle	50	50
Period	4	2

Output Screen



4:1 Multiplexer

- Truth Table

Select I/P		Input				Output
S ₁	S ₀	I ₃	I ₂	I ₁	I ₀	Y
0	0	I ₃	I ₂	I ₁	I ₀	I ₀
0	1	I ₃	I ₂	I ₁	I ₀	I ₁
1	0	I ₃	I ₂	I ₁	I ₀	I ₂
1	1	I ₃	I ₂	I ₁	I ₀	I ₃

Verilog HDL Program

```

module mux_4to1(I3,I2,I1,I0,S1,S0,Y);
input I3,I2,I1,I0,S1,S0;
output Y;
reg Y;
always @ (I3 or I2 or I1 or I0 or S1 or S0)
begin
if (S1 == 0 & S0 == 0)

```

```

Y = I0;
else if (S1 == 0 & S0 == 1)
Y = I1;
else if (S1 == 1 & S0 == 0)
Y = I2;
else
begin
Y = I3;
end
end
endmodule

```

Input Value for Force Clock

Signal Name	S1	S0	I3 to I0
Value Radix	Binary	Binary	Binary
Leading Edge Value	0	0	0
Trailing Edge Value	1	1	1
Starting At Time Offset	0	0	0
Cancel After Time Offset	8ps	8ps	8ps
Duty Cycle	50	50	50
Period	8	4	2

Output Screen



8:1 Multiplexer

- Truth Table

Select I/P			Input								Output
S ₂	S ₁	S ₀	I ₇	I ₆	I ₅	I ₄	I ₃	I ₂	I ₁	I ₀	Y
0	0	0	I ₇	I ₆	I ₅	I ₄	I ₃	I ₂	I ₁	I ₀	I ₀
0	0	1	I ₇	I ₆	I ₅	I ₄	I ₃	I ₂	I ₁	I ₀	I ₁
0	1	0	I ₇	I ₆	I ₅	I ₄	I ₃	I ₂	I ₁	I ₀	I ₂
0	1	1	I ₇	I ₆	I ₅	I ₄	I ₃	I ₂	I ₁	I ₀	I ₃
1	0	0	I ₇	I ₆	I ₅	I ₄	I ₃	I ₂	I ₁	I ₀	I ₄
1	0	1	I ₇	I ₆	I ₅	I ₄	I ₃	I ₂	I ₁	I ₀	I ₅
1	1	0	I ₇	I ₆	I ₅	I ₄	I ₃	I ₂	I ₁	I ₀	I ₆
1	1	1	I ₇	I ₆	I ₅	I ₄	I ₃	I ₂	I ₁	I ₀	I ₇

Verilog HDL Program

```

module mux_8to1 (I7, I6, I5, I4, I3, I2, I1, I0, S2, S1, S0, Y);
input I7, I6, I5, I4, I3, I2, I1, I0, S2, S1, S0;
output Y;
reg Y;
always @ (I7 or I6 or I5 or I4 or I3 or I2 or I1 or I0 or S2 or S1 or S0)
begin
if (S2 ==0 & S1 == 0 & S0 ==0)
Y = I0;
else if (S2 ==0 & S1 == 0 & S0 ==1)
Y = I1;
else if (S2 ==0 & S1 == 1 & S0 ==0)
Y = I2;
else if (S2 ==0 & S1 == 1 & S0 ==1)
Y = I3;
else if (S2 ==1 & S1 == 0 & S0 ==0)
Y = I4;
else if (S2 ==1 & S1 == 0 & S0 ==1)
Y = I5;
else if (S2 ==1 & S1 == 1 & S0 ==0)

```

```

Y = I6;
else
begin
Y = I7;
end
end
endmodule

```

Input Value for Force Clock

Signal Name	S1	S0	I7 to I0
Value Radix	Binary	Binary	Binary
Leading Edge Value	0	0	0
Trailing Edge Value	1	1	1
Starting At Time Offset	0	0	0
Cancel After Time Offset	16ps	16ps	16ps
Duty Cycle	50	50	50
Period	16	8	4

Output Screen



Result: Verilog program to implement Different types of multiplexer like 2:1, 4:1 and 8:1 is designed and truth table is verified using Verilog HDL Simulator Software.

EXPERIMENT NO 7

DESIGN VERILOG PROGRAM TO IMPLEMENT TYPES OF DE-MULTIPLEXER.

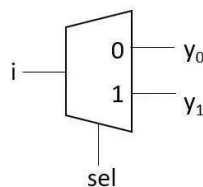
AIM: Design Verilog program to implement types of De-Multiplexer 1:2, 1:4, 1:8.

Software requirement: ISE Design Suite 14.7(Xlink), Isim Simulator.

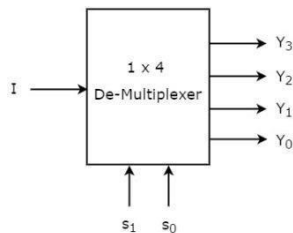
Description:

A De-multiplexer is a combinational circuit that has only 1 input line and 2^N output lines. Simply, the demultiplexer 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.

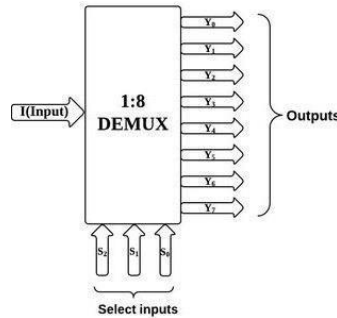
1:2 De-multiplexers: In the 1 to 2 De-multiplexer, there are only two outputs, i.e., Y_0 , and Y_1 , 1 selection lines, i.e., $S_0(\text{sel})$ and single input, i.e., i . On the basis of the selection value, the input will be connected to one of the outputs. The block diagram and the truth table of the 1×2 multiplexer are given below.



1:4 De-multiplexers: 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., I . On the basis of 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 are given below.



1:8 De-multiplexers: 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., I . 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 are given below.



1:2 De-multiplexers

- Truth Table

Select I/P	Input	Output	
Sel	I	Y_1	Y_0
0	I	0	I
1	I	I	0

Verilog HDL Program

```

module Demux_1to2(I,Sel,Y0,Y1);
input I,Sel;
output Y0,Y1;
assign Y0 = (I&(~S0));
assign Y1 = (I & S0);
endmodule

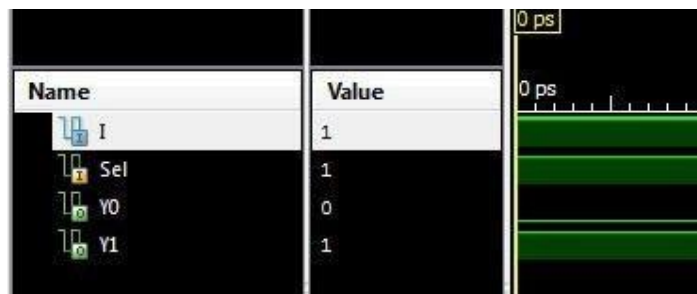
```

Input Value for Force Clock

Signal Name	S	I
Value Radix	Binary	Binary

Leading Edge Value	0	0
Trailing Edge Value	1	1
Starting At Time Offset	0	0
Cancel After Time Offset	4ps	4ps
Duty Cycle	50	50
Period	4	2

Output Screen



1:4 De-multiplexers

- Truth Table

Select I/P		Input	Output			
S ₁	S ₀	I	Y ₃	Y ₂	Y ₁	Y ₀
0	0	I	0	0	0	I
0	1	I	0	0	I	0
1	0	I	0	I	0	0
1	1	I	I	0	0	0

Verilog HDL Program

```

module Demux_1to4(I,S0,S1,Y0,Y1,Y2,Y3);
input I,S0,S1;
output Y0,Y1,Y2,Y3;
assign Y0 = (I & (~S1 & ~S0));
assign Y1 = (I & (~S1 & S0));
assign Y2 = (I & (S1 & ~S0));

```

```
assign Y3=(I &(S1&S0));
endmodule
```

Input Value for Force Clock

Signal Name	S1	S0	I
Value Radix	Binary	Binary	Binary
Leading Edge Value	0	0	0
Trailing Edge Value	1	1	1
Starting At Time Offset	0	0	0
Cancel After Time Offset	8ps	8ps	8ps
Duty Cycle	50	50	50
Period	8	4	2

Output Screen



1:8 De-multiplexers

- Truth Table

Select I/P			Input	Output							
S ₂	S ₁	S ₀	I	Y ₇	Y ₆	Y ₅	Y ₄	Y ₃	Y ₂	Y ₁	Y ₀
0	0	0	I	0	0	0	0	0	0	0	I
0	0	1	I	0	0	0	0	0	0	I	0
0	1	0	I	0	0	0	0	0	I	0	0
0	1	1	I	0	0	0	0	I	0	0	0
1	0	0	I	0	0	0	I	0	0	0	0
1	0	1	I	0	0	I	0	0	0	0	0

1	1	0	I	0	I	0	0	0	0	0	0
1	1	1	I	I	0	0	0	0	0	0	0

Verilog HDL Program

```
module Demux_1to8(I,S0,S1,S2,Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7);
```

```
input I,S0,S1,S2;
```

```
output Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7;
```

```
assign Y0 = (I & (~S2 & ~S1 & ~S0));
```

```
assign Y1 = (I & (~S2 & ~S1 & S0));
```

```
assign Y2 = (I & (~S2 & S1 & ~S0));
```

```
assign Y3 = (I & (~S2 & S1 & S0));
```

```
assign Y4 = (I & (S2 & ~S1 & ~S0));
```

```
assign Y5 = (I & (S2 & ~S1 & S0));
```

```
assign Y6 = (I & (S2 & S1 & ~S0));
```

```
assign Y7 = (I & (S2 & S1 & S0));
```

```
endmodule
```

Input Value for Force Clock

Signal Name	S2	S1	S0	I
Value Radix	Binary	Binary	Binary	Binary
Leading Edge Value	0	0	0	0
Trailing Edge Value	1	1	1	1
Starting At Time Offset	0	0	0	0
Cancel After Time Offset	16ps	16ps	16ps	16ps
Duty Cycle	50	50	50	50
Period	16	8	4	2

Output Screen

Name	Value	0 ps
I	1	
S0	1	
S1	1	
S2	1	
Y0	0	
Y1	0	
Y2	0	
Y3	0	
Y4	0	
Y5	0	
Y6	0	
Y7	1	

Result: Design Verilog program to implement Different types of De-multiplexer like 1:2, 1:4 and 1:8 and truth table is verified using Verilog HDL Simulator Software.

EXPERIMENT NO 8

DESIGN VERILOG PROGRAM FOR IMPLEMENTING VARIOUS TYPES OF FLIP-FLOPS SUCH AS SR, JK D AND T

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

Software requirement: ISE Design Suite 14.7(Xlink), Isim Simulator.

Description:

- **SR flip flop**

The SR flip flop is a 1-bit memory bistable device having two inputs, i.e., SET and RESET. The SET input 'S' set the device or produce the output 1, and the RESET input 'R' reset the device or produce the output 0. The SET and RESET inputs are labeled as S and R, respectively.

The SR flip flop stands for "Set-Reset" flip flop. The reset input is used to get back the flip flop to its original state from the current state with an output 'Q'. This output depends on the set and reset conditions, which is either at the logic level "0" or "1".

- **JK flip flop**

The JK flip flop is one of the most used flip flops in digital circuits. The JK flip flop is a universal flip flop having two inputs 'J' and 'K'. In SR flip flop, the 'S' and 'R' are the shortened abbreviated letters for Set and Reset, but J and K are not. The J and K are themselves autonomous letters which are chosen to distinguish the flip flop design from other types.

The JK flip flop work in the same way as the SR flip flop work. The JK flip flop has 'J' and 'K' flip flop instead of 'S' and 'R'. The only difference between JK flip flop and SR flip flop is that when both inputs of SR flip flop is set to 1, the circuit produces the invalid states as outputs, but in case of JK flip flop, there are no invalid states even if both 'J' and 'K' flip flops are set to 1.

- **D flip flop**

In SR NAND Gate Bistable circuit, the undefined input condition of SET = "0" and RESET = "0" is forbidden. It is the drawback of the SR flip flop. The D flip flop is the most important flip flop from other clocked types. It ensures that at the same time, both the inputs, i.e., S and R, are never equal to 1. The Delay flip-flop is designed using a gated SR flip-flop with an inverter connected between the inputs allowing for a single input D(Data).

- **T flip flop**

In T flip flop, "T" defines the term "Toggle". In SR Flip Flop, we provide only a single input called "Toggle" or "Trigger" input to avoid an intermediate state occurrence. Now, this flip-flop work as a Toggle switch. The next output state is changed with the complement of the present state output. This process is known as "Toggling".

SR flip flop

- **Truth Table**

Input		Output		Condition
S	R	Q	\bar{Q}	
0	0	0	1	No Change
0	1	1	0	SET
1	0	0	1	RESET
1	1	X	X	Invalid

Verilog HDL Program

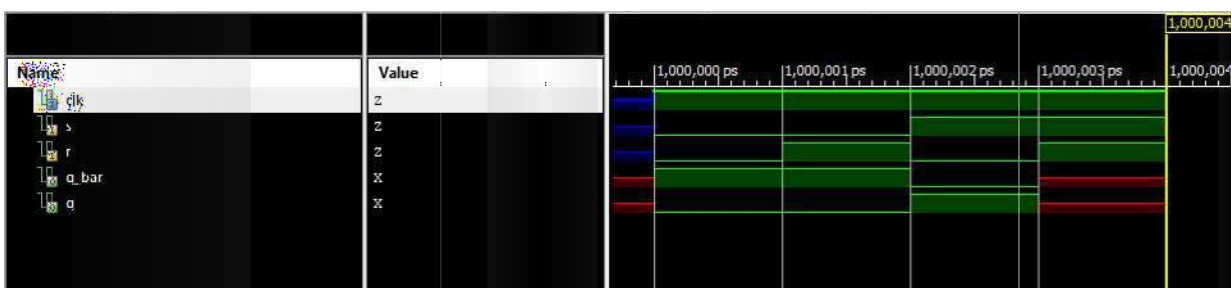
```

module SR_FF(

input clk,
input s,r,
output reg q,
output q_bar
);
always@(posedge clk)
begin
case({s,r})
2'b00: q = q; // No change
2'b01: q = 1'b0; // reset
2'b10: q = 1'b1; // set
2'b11: q = 1'bx; // Invalid inputs
endcase
end
assign q_bar = ~q;
endmodule

```

Output Screen



J K flip flop

- Truth Table

Input		Output		Condition
J	K			
0	0	0	1	No Change
0	1	1	0	SET
1	0	0	1	RESET
1	1	1	0	Toggle

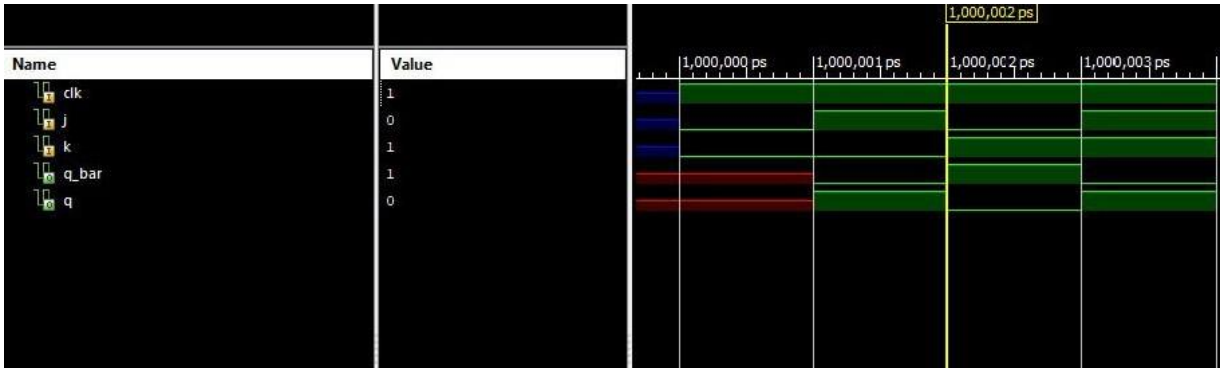
Verilog HDL Program

```

module JK_FF(
input clk,
input j,k,
output reg q,
output q_bar );
always@(posedge clk)
begin
case({j,k})
2'b00: q = q; // No change
2'b01: q = 1'b0; // reset
2'b10: q = 1'b1; // set
2'b11: q = ~q; // Invalid inputs
endcase
end
assign q_bar = ~q;
endmodule

```


Output Screen



D Flip flop

Input	Output		Condition
D			
0	0	0	RESET
1	1	0	SET

Verilog HDL Program

```
module D_FF(  
input clk,  
input d,  
output reg q,  
output q_bar  
);  
always@(posedge clk)  
begin  
case({d})  
1'b0: q = 1'b0; // reset  
1'b1: q = 1'b1; // set  
endcase  
end  
assign q_bar = ~q;  
endmodule
```

Output Screen**T Flip flop**

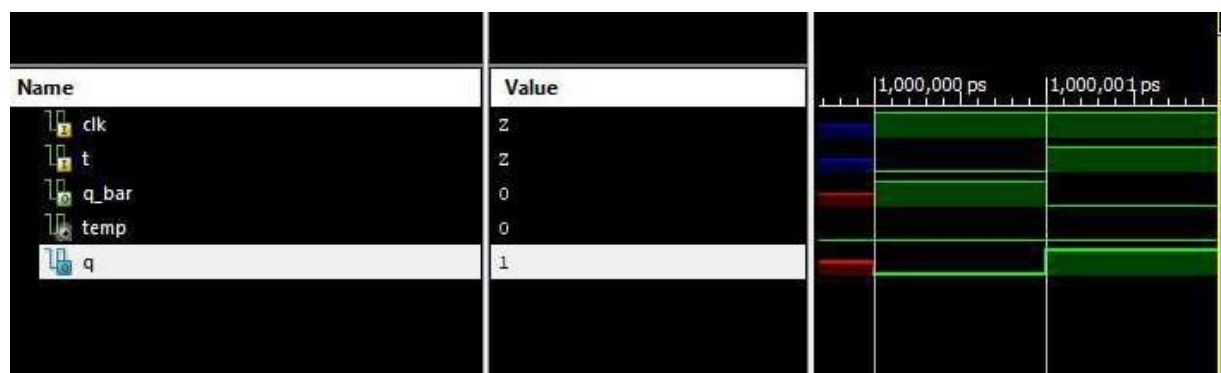
Input	Output		Condition
D			
0			No change
1			Complement

Verilog HDL Program

```

module T_FF(
input clk,
input d,
output reg q,
output q_bar
);
assign temp=0;
always@(posedge clk)
begin
case({t})
1'b0: q = temp; // no change
1'b1: q = ~temp; // complement
endcase
end
assign q_bar = ~q;
endmodule

```

Output Screen

Result: Design Verilog program to implement Different types of Flip flop like SR, JK, T and D and truth table is verified using Verilog HDL Simulator Software.