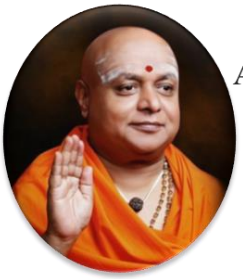
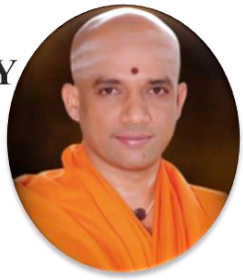
||Jai Sri Gurudev||



ADICHUNCHANAGIRI INSTITUTE OF TECHNOLOGY

CHIKKAMAGALURU



DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

LAB MANUAL

SEMESTER III

DIGITAL DESIGN AND COMPUTER ORGANIZATION
(BCS302)
(Academic Year 2024-2025)

IPCC: Integrated Professional Core Course

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

When all castes and creeds live together with a sense of equality and brotherhood, they will be truly happy and country strong. Collective organization of society has its rewards.

--Sri Sri Sri Dr. Balagangadharanatha Mahaswamiji

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi, Karnataka –590018

SEMESTER III

DIGITAL DESIGN AND COMPUTER ORGANIZATION

(BCS302)

(Academic Year 2024-2025)

IPCC: Integrated Professional Core Course

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

ADICHUNCHANAGIRI INSTITUTE OF TECHNOLOGY

8QRX+P7W, BEEKANAHALLI RURAL CHIKKAMAGALURU, KARNATAKA 577101

(2024-2025)

Mr. Abhishek N B.E., M.Tech,.
Assistant Professor,
Dept. of IS&E,
A.I.T Chikkamagaluru,

Dr. Sampath S B.E., M.Tech., Ph.D.,
Professor & Head,
Dept. of IS&E,
A.I.T Chikkamagaluru,

DEPARMENT VISION AND MISSION

The department was started in the year 1999-2000 with an intake of 60. Department comprises of highly qualified and experienced teaching staff and research scholars.

Department is performing well by securing VTU ranks in academics. With support from the institution, the department is constantly providing career guidance for the students in order to achieve good results and it has excellent placement records.

Department regularly conducts technical talks, seminars, short-term courses, hands-on training, and workshops for students and faculty to bridge the technical gap between educational institutions and industries.

VISION

To be recognized as a centre of excellence in information technology and allied areas with quality learning and research environment

MISSION

- Provide intellectual & professional leadership in ethical and social areas pertaining to information in contemporary society.
- Advancing the state of knowledge of information studies through research and development.
- Advancing the state of knowledge of information studies through research and development.

Program Educational Objectives (PEO's)

- **PEO1:** Graduates will be able to analyze, design and provide solutions to the problems in the field of information science and engineering
- **PEO2:** Graduates will be able to exhibit the quality of working in team and build leadership quality
- **PEO3:** Graduates will be able to learn new technologies that will be helpful in their professional success
- **PEO4:** To train students with good breadth of knowledge in core areas of Information Technology

Course objectives:

1. To demonstrate the functionalities of binary logic system
2. To explain the working of combinational and sequential logic system
3. To realize the basic structure of computer system
4. To illustrate the working of I/O operations and processing unit

Course outcome:

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.

Do's

- Maintain Silence inside the Laboratory.
- Maintain Cleanliness.
- Leave your footwear outside the Laboratory.
- Enter the required details in the Log Book.
- Handle the Laboratory Instrument Carefully.
- Bring Laboratory Record and the Observation Book regularly.
- Any malfunctioning of Computer should be brought to the notice of Laboratory In-charge immediately.
- Facilities can be utilized during the free slots with the permission of Laboratory In-charge.
- Always shut down your computer from start menu and ensure the terminal/computer is logged off property.

Don'ts

- Do not delete or copy the files from the system.
- Do not displace Laboratory equipment.
- Do not shut down your computer by switching off the power directly.

These are Strictly Prohibited

- Entering the Laboratory without uniform.
- Group discussion or chatting.
- Walking around the Laboratory.
- Mobile Phone and Pen Drive.

LIST OF CONTENTS

INTRODUCTION

BASIC GATES

There are seven basic gates:

- AND(7408)
- OR(7432)
- EX-OR(7486)
- NOT(7404)
- XNOR(74266)

UNIVERSAL GATES:

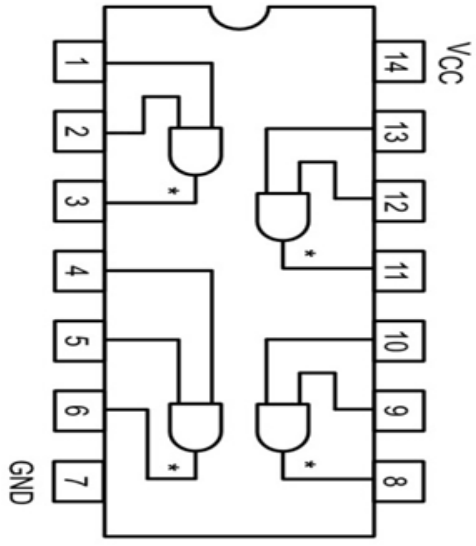
There are two universal gates:

- NAND(7400)
- NOR(7402)

BASIC GATES

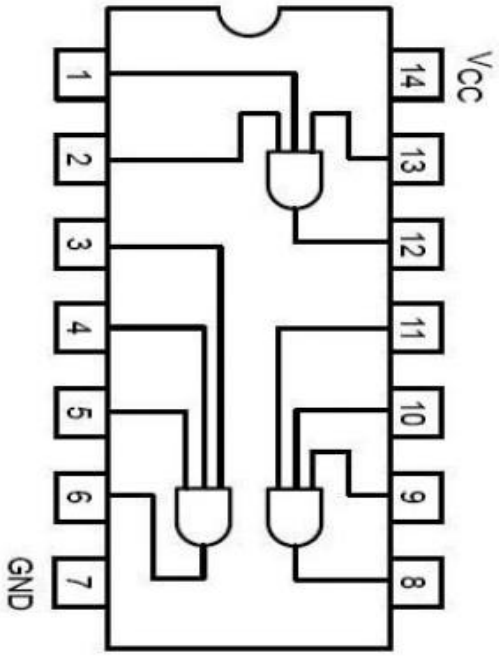
- **AND (7408):**

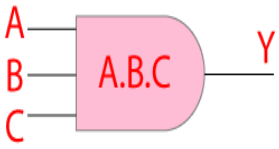
The AND gate is so named because, if 0 is called "false" and 1 is called "true," the gate acts in the same way as the logical "and" operator. The following illustration and table show the circuit symbol and logic combinations for an AND gate. (In the symbol, the input terminals are at left and the output 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.

<table>
<tr><th>Pin Diagram(7408)</th><th>Symbol</th><th>Truth Table</th></tr>
<tr>
<td> 
</td>
<td> 
</td>
<td>
<table>
<tr><th colspan="2">Input</th><th>Output</th></tr>
<tr><th>A</th><th>B</th><th>Y=AB</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
</tr>
</table>

- **3 Input AND (7411):**

3 Input AND Pin Diagram(7411)	Symbol	Truth Table



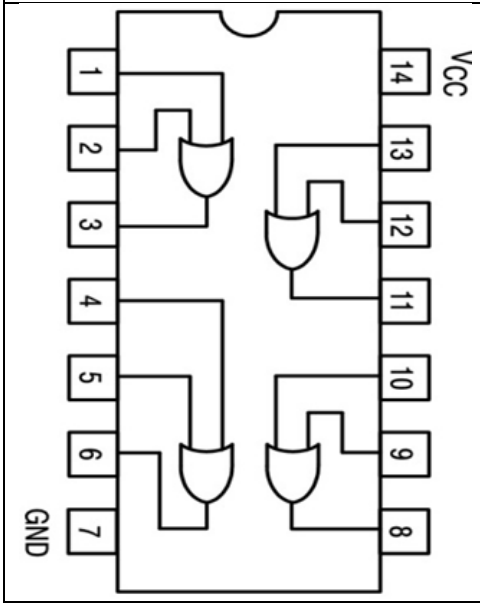


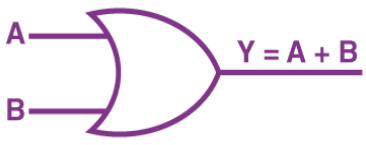
Input			Output
A	B	C	Y=ABC
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- **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

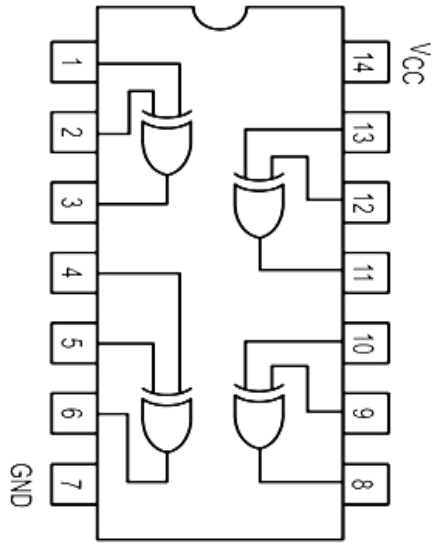
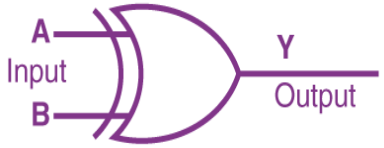




Input		Output
A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

- **EX-OR (7486):**

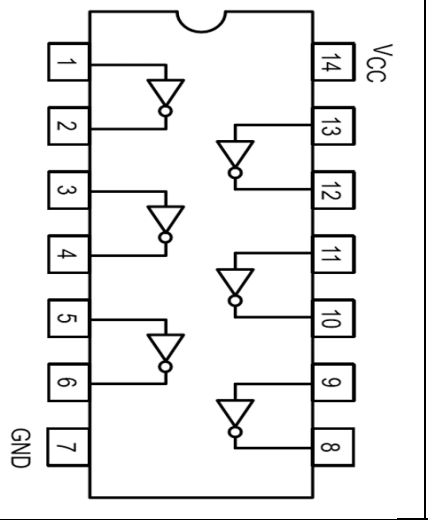
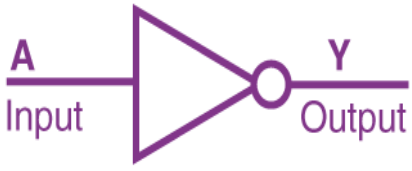
The XOR (exclusive-OR) gate acts in the same way as the logical "either/or." The output is "true" if either, but not both, of the inputs are "true." The output is "false" if both inputs are "false" or if both inputs are "true." Another way of looking at this circuit is to observe that the output is 1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

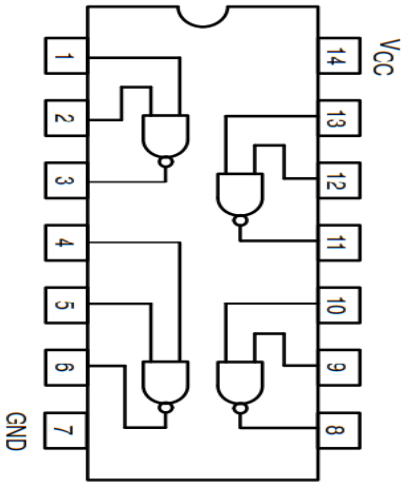
Input	Output
A	Y=$\bar{A}$
0	1
1	0

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."

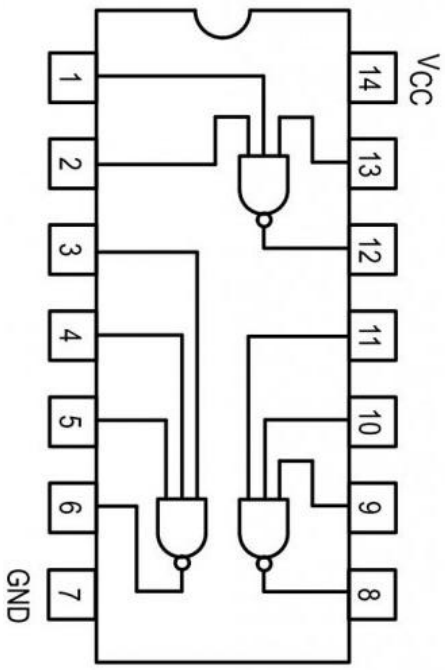
Pin Diagram(7400)	Symbol	Truth Table

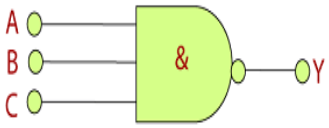


Input		Output
A	B	$Y=\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

- **3 Inputs NAND (7411):**

3 Pin Diagram(7410)	Symbol	Truth Table



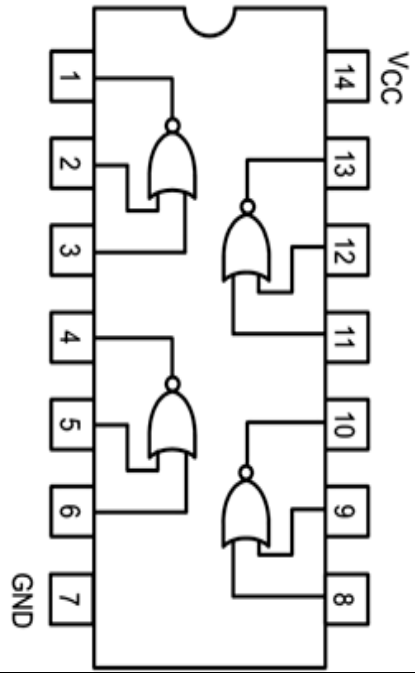


Input			Output
A	B	C	$Y=\overline{ABC}$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

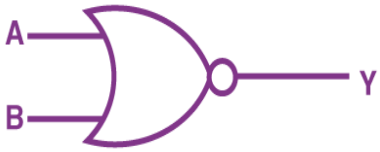
- **NOR (7402):**

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	$Y=\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

VHDL

VHDL is an acronym for VHSIC Hardware Description Language (VHSIC is an acronym for Very High Speed Integrated Circuits). It is a hardware description language that can be used to model a digital system at many levels of abstraction ranging from the algorithmic level to the gate level. The complexity of the digital system being modeled could vary from that of a simple gate to a complete digital electronic system, or anything in between. The digital system can also be described hierarchically. Timing can also be explicitly modeled in the same description.

The language not only defines the syntax but also defines very clear simulation semantics for each language construct. Therefore, models written in this language can be verified using a VHDL simulator. It is a strongly typed language and is often verbose to write. It inherits many of its features, especially the sequential language part, from the Ada programming language. Because VHDL provides an extensive range of modeling capabilities, it is often difficult to understand.

Fortunately, it is possible to quickly assimilate a core subset of the language that is both easy and simple to understand without learning the more complex features. This subset is usually sufficient to model

most applications. The complete language, however, has sufficient power to capture the descriptions of the most complex chips to a complete electronic system.

VHDL provides five different types of primary constructs, called design units. They are

1. Entity declaration
2. Architecture body
3. Configuration declaration
4. Package declaration
5. Package body

Styles of Modeling:

1. Data flow modeling (Design Equations)
2. Behavioral modeling (Explains Behaviour)
3. Structural modeling (Connection of sub modules)

Data flow modeling (Design Equations):

A dataflow model specifies the functionality of the entity without explicitly specifying its structure. This functionality shows the flow of information through the entity, which is expressed primarily using concurrent signal assignment statements and block statements. This is in contrast to the behavioral style of modeling described in the previous chapter, in which the functionality of the entity is expressed using procedural type statements that are executed sequentially. This chapter also describes resolution functions and their usage

Behavioral modeling (Explains Behaviour):

In this modeling style, the behavior of the entity is expressed using sequentially executed, procedural type code, a process statement is the primary mechanism used to model the procedural type behavior of an entity. This chapter describes the process statement and the various kinds of sequential statements that can be used within a process statement to model such behavior.

Irrespective of the modeling style used, every entity is represented using an entity declaration and at least one architecture body. The first two sections describe these in detail.

An architecture body describes the internal view of an entity. It describes the functionality or the structure of the entity. The syntax of an architecture body is an architecture body describes the internal view of an entity. It describes the functionality or the structure of the entity.

Structural modeling (Connection of sub modules):

The structural style of modeling. An entity is modeled as a set of components connected by signals, that is, as a net list. The behavior of the entity is not explicitly apparent from its model. The component instantiation statement is the primary mechanism used for describing such a model of an entity.

VHDL objects

1. **Constants:** Constant is an object which can only hold a single value that cannot be changed during the whole code.
   Example: constant number_of_bytes integer:=8;
2. **Variables:** A variable also holds a single value of a given type. The value of the variable may be changed during the simulation by using variable assignment operator. Variables are used in the processes and subprograms
   Variables are assigned by the assignment operator ":=".
   Variable index: Integer: =0;
3. **Signals:** Signals can be declared in architecture and used anywhere within the architecture. Signals are assigned by the assignment operator "<=".
   Signal sig1: std_logic;
   Sig1 <= '1'

PSPICE

The steps to simulation

1. Create a simulation project
2. Draw schematic to simulate
3. Establish a simulation profile
4. Set up simulation type
5. Simulate circuit
6. Analyse results in Probe

General Procedure to use for all simulation experiments:

Start--------Programs---------Orcad family lite edition---------Capture lite edition File

1. Select new & blank project, select Analog & mixed mode for new simulation (use open project for already existing project).
2. Layout appears select components from parts & place at required position on layout, connects components using wire, apply voltage marker at i/p & o/p to see wave forms.
3. If components are not available it is required to add by using add library present in window.
4. From menu bar select PSpice, new simulation profile to set simulation profile like Select Analysis type: Time domain (or) AC sweep as per experiment. Set Start value, Step size & End value then Save settings.
5. Run simulation observes & notes the output waveform.
6. Use Edit profile for any changes required in profile.
7. Simulation can be done for different values of component & supply.

Xilinx

Software package used for Digital experiment Xilinx 14.7, it is one of most popular software tool used to synthesize VHDL code. This tool includes many steps. To make user feel comfortable with the tool the steps are given below:-

These steps are common to all Digital experiment for simulation & synthesis part.

- Double click on Project navigator. (Assumed icon is present on desktop).
- Select NEW PROJECT in FILE MENU.

Enter following details as per your convenience

- Project name : new
- Project location: C:\create your folder
- Top level module : HDL
- In NEW PROJECT dropdown Dialog box, Choose your appropriate device specification.

Example is given below:

Device family**:** Spartan3/ Spartan6

Device: XC3S50

Package: PQ208

TOP Level Module: HDL

Synthesis Tool: XST

Simulation: Isim

Generate sim language: VHDL

In source window right click on specification, select new source

Enter the following details

Entity: sample

Architecture: Behavioural

Enter the input and output port and modes.

This will create sample.VHD source file. Click next and finish the initial Project preparation.

- Double click on synthesis. If error occurs edit and correct VHDL code.
- Top level module appears, again select new source & set test bench waveform.

  Double click on Lunch Isim (or any equivalent simulator if you are using) for functional simulation of your design

EXPERIMENT NO 1

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

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

Components Required: Logic gates IC-7404, 7408, 7432, 7411, Digital IC trainer kit, Patch Cords.

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.

The basic laws of Boolean algebra-the commutative laws for addition and multiplication, the associative laws for addition and multiplication, and the distributive law-are the same as in ordinary algebra.

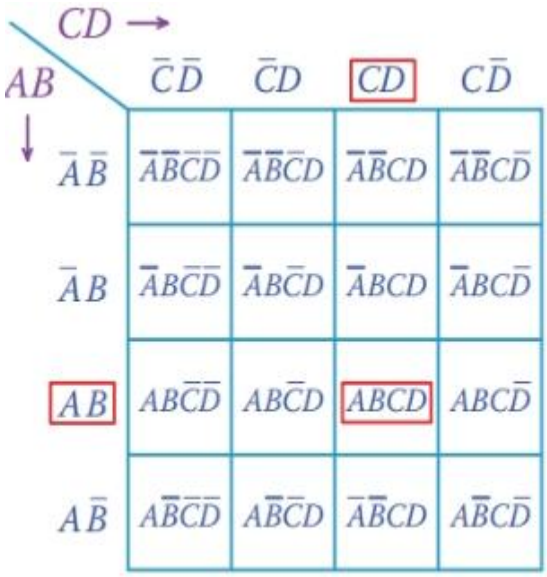
Karnaugh's Map:

4 Variables Karnaugh's Map often known as 4 variables K-Map. It's 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 KMAP table.

The min-term SOP is often denoted by ABCD, 1s & 0s or decimal numbers. For example, the Boolean expression $Y = \sum\{2, 6, 9, 11, 15\}$ represents the place values of the respective cells which have the higher values (binary 1s). The$= \sum\{2, 6, 9, 11, 15\}$ } can also be represented by $Y = \sum\{0010, 0110, 1001, 1011, 1111\}$ or $Y = \sum\{\bar{A}\bar{B}C\bar{D}, \bar{A}BC\bar{D}, A\bar{B}\bar{C}D, A\bar{B}CD, ABCD\}$ 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.

Variable Representation Table:

VARIABLES				REPRESENTATION	DECIMAL NOTATION
A	B	C	D	$ABCD$	NUMBER
0	0	0	0	$\bar{A}\bar{B}\bar{C}\bar{D}$	0
0	0	0	1	$\bar{A}\bar{B}\bar{C}D$	1
0	0	1	0	$\bar{A}\bar{B}C\bar{D}$	2
0	0	1	1	$\bar{A}\bar{B}CD$	3
0	1	0	0	$\bar{A}B\bar{C}\bar{D}$	4
0	1	0	1	$\bar{A}B\bar{C}D$	5
0	1	1	0	$\bar{A}BC\bar{D}$	6
0	1	1	1	$\bar{A}BCD$	7
1	0	0	0	$A\bar{B}\bar{C}\bar{D}$	8
1	0	0	1	$A\bar{B}\bar{C}D$	9
1	0	1	0	$A\bar{B}C\bar{D}$	10
1	0	1	1	$A\bar{B}CD$	11
1	1	0	0	$AB\bar{C}\bar{D}$	12
1	1	0	1	$AB\bar{C}D$	13
1	1	1	0	$ABC\bar{D}$	14
1	1	1	1	$ABCD$	15

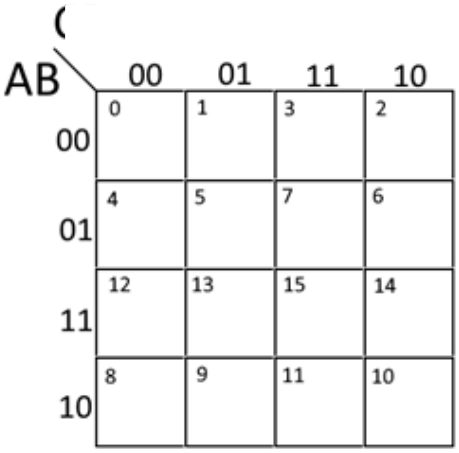


1. $f(A,B,C,D) = \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + A\bar{B}C\bar{D} + AB\bar{C}\bar{D} + AB\bar{C}D$

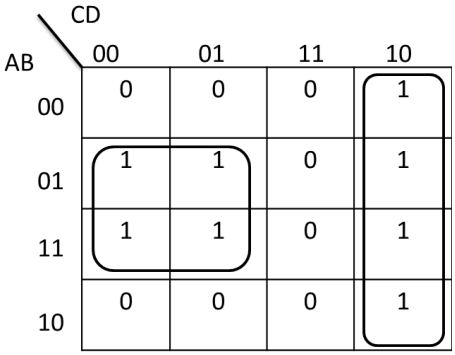
Write decimal notation from above given equation

$f(A,B,C,D) = \sum\{2,4,5,6,10,12,13,14\}$

Apply on Karnaugh's Map:



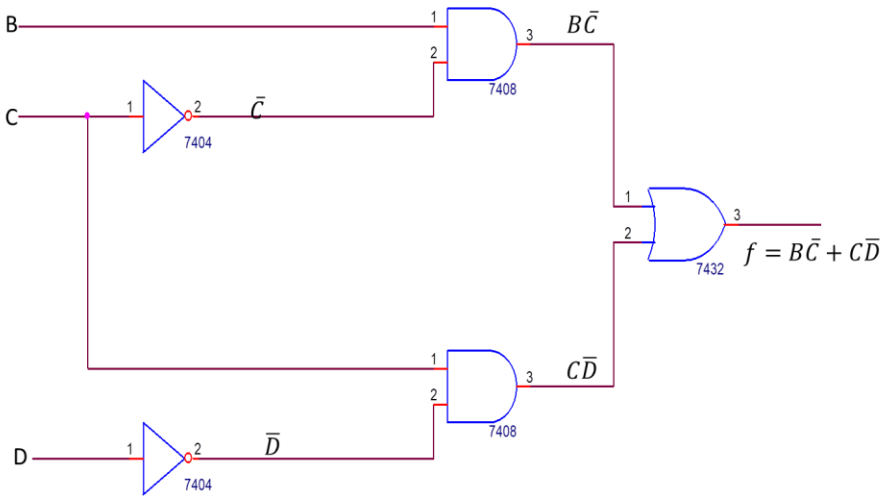
Simplification using Karnaugh's Map:



$$f(A,B,C,D) = B\bar{C} + C\bar{D}$$

$$f = B\bar{C} + C\bar{D}$$

Design using basic gets:



Truth Table:

Decimal	INPUT				OUTPUT
	A	B	C	D	$f = B\bar{C} + C\bar{D}$
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	0

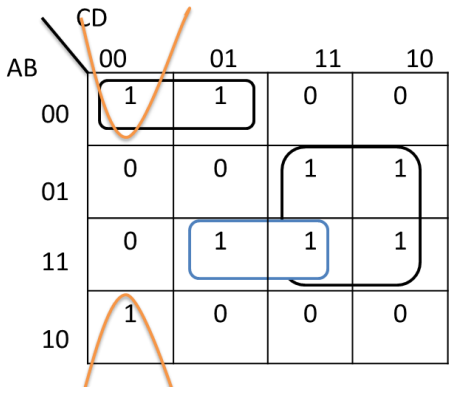
Result:

2. $f(A,B,C,D) = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D} + \bar{A}BCD + A\bar{B}\bar{C}\bar{D} + AB\bar{C}D + ABC\bar{D} + ABCD$

Write decimal notation from above given equation

$f(A,B,C,D) = \sum\{0,1,6,7,8,13,14,15\}$

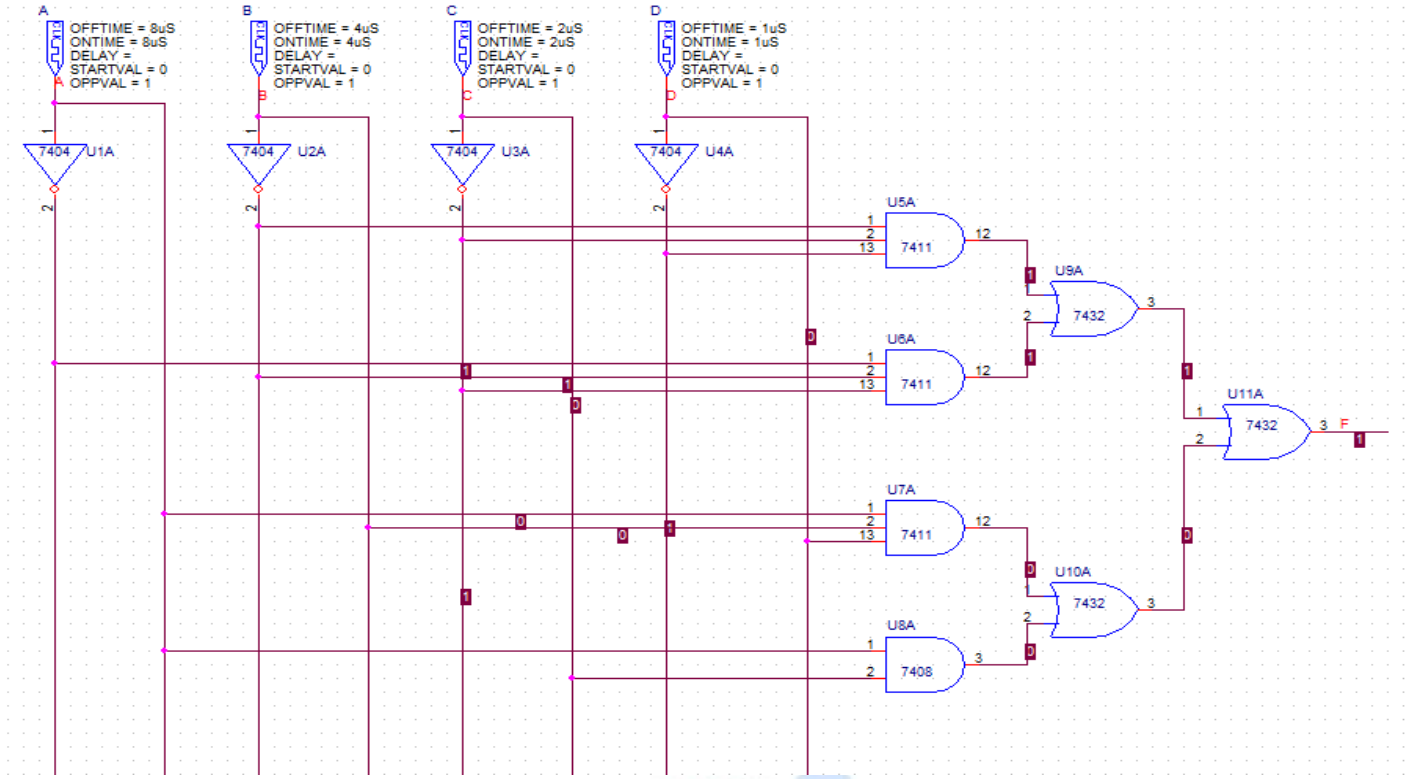
Simplification using Karnaugh's Map:



$$f(A,B,C,D) = \bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C} + ABD + BC$$

$$f = \bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C} + ABD + BC$$

Design using basic gates:



Circuit Diagram ***Click here***

Simulation Profile:

Type of analysis: TIME DOMAIN (TRANSIENT)

Run to time: 16us

Step size: 1us

Output Screen:

Truth Table:

Decimal	INPUT				OUTPUT
	A	B	C	D	$f = \bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C} + ABD + BC$
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

Result: Given a 4-variable logic expression, simplify it using appropriate technique and implement the same using basic gate. Truth table is verified using Suitable Simulator Software.

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.

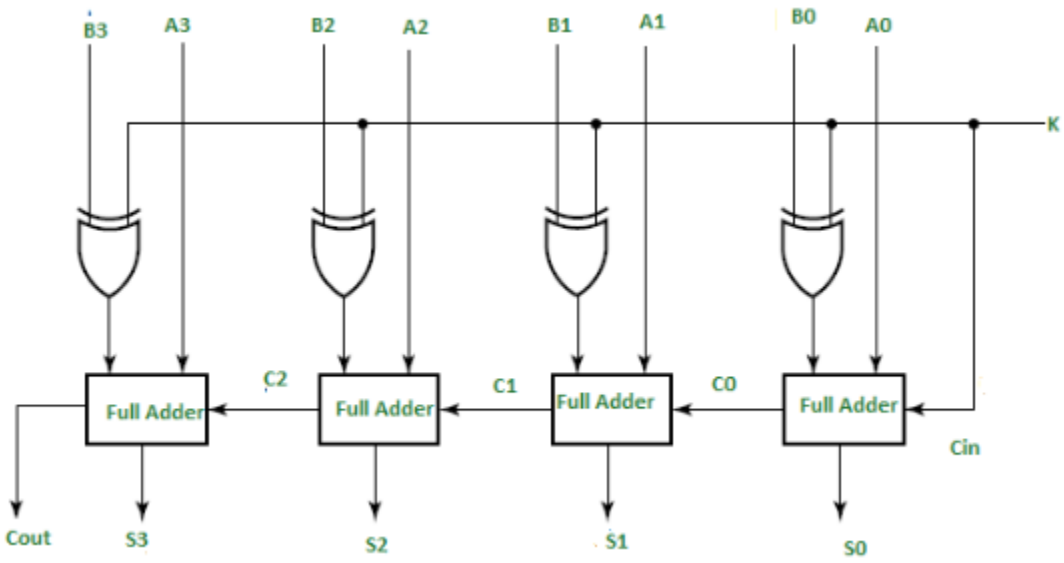
Components Required: Logic gates IC-7486, 7408, 7432, or 7483, Digital IC trainer kit, Patch Cords.

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.

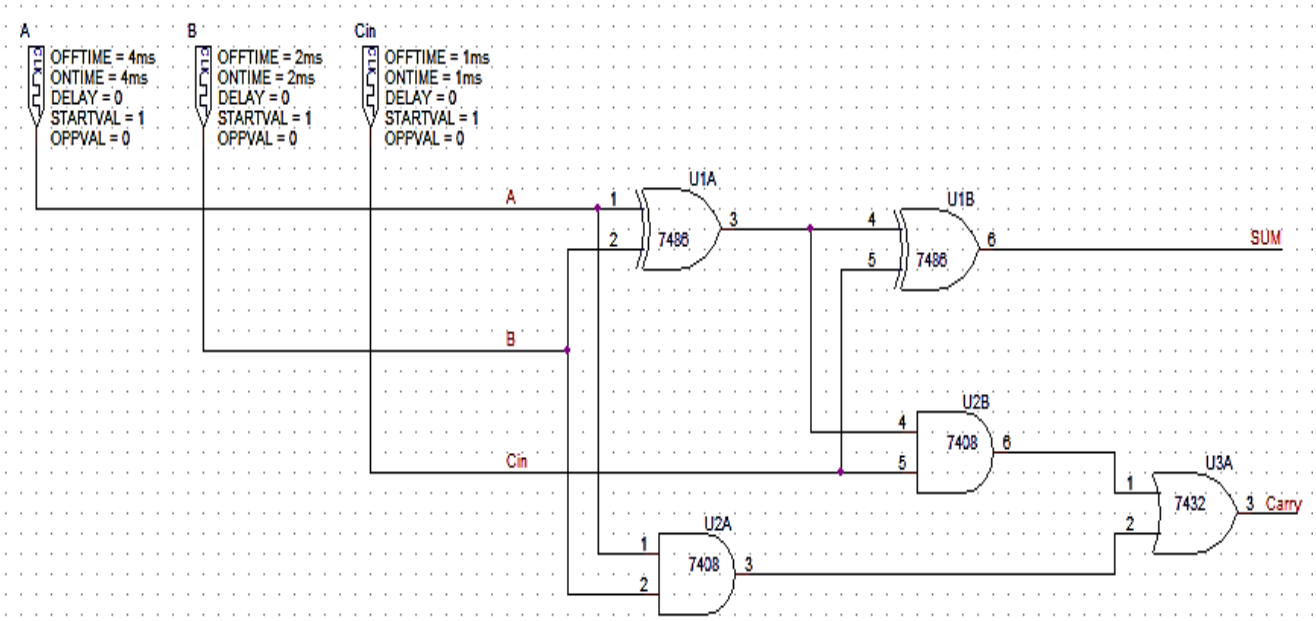


As shown in the figure, the first full adder has a control line directly as its input (input carry Cin), The input A0 (The least significant bit of A) is directly input in the full adder. The third input is the ex-or of B0 and K. The two outputs produced are Sum/Difference (S0) and Carry (C0). If the 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'+Cin. 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 C0 is serially passed to the second full adder as one of it's outputs. The sum/difference S0 is recorded as the least significant bit of the sum/difference. A1, A2, A3 are direct inputs to the second, third and fourth full adders. Then the third input is the B1, B2, B3 EX-ORed with K to the second, third and fourth full adder respectively. The carry C1, C2 are serially passed to the successive full adder as one of the inputs. C3 becomes the total carry to the sum/difference. S1, S2, S3 are recorded to form the result with S0.

Full adder



Truth Table

Inputs			Outputs	
A	B	C_{in}	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

4 bit Full adder/Subtractor:

K=0 Adder

K=1 Subtractor

Circuit Diagram:

Note:* Click here *to See Circuit Diagram

Simulation Profile:

Type of analysis: TIME DOMAIN (TRANSIENT)

Run to time: 16ms

Step size: 1ms

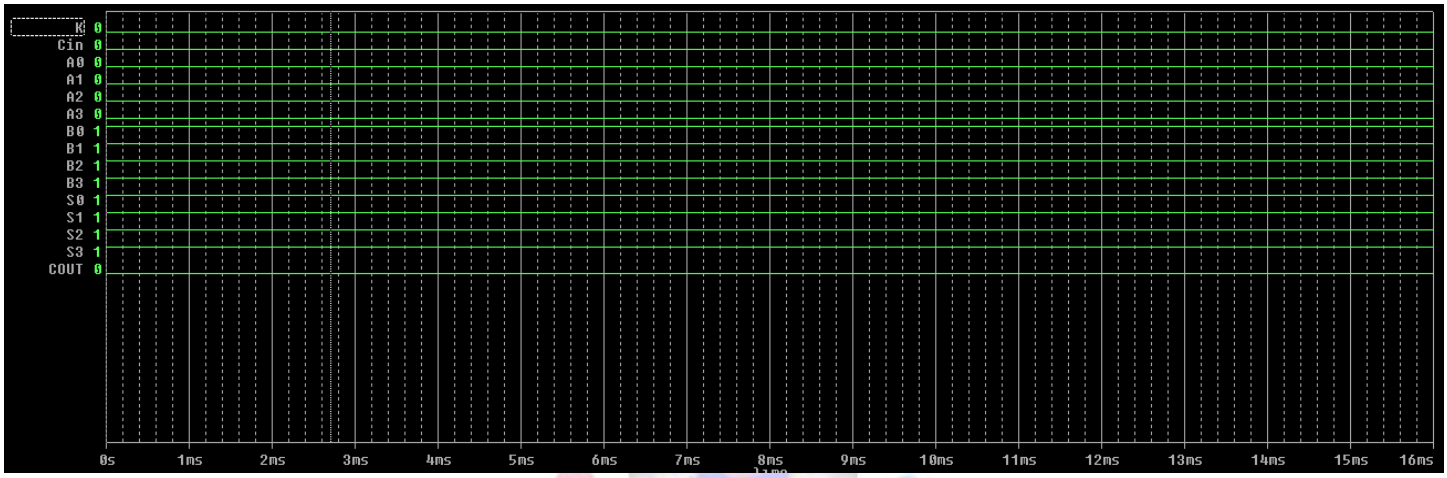
4 Bit Full Adder and Subtractor Truth Table

CONTROL		INPUT												
		4 Bit Full Adder												
Cin	K	A3	A2	A1	A0	B3	B2	B2	B1	S3	S2	S1	S0	Cout
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	1	0	0	1	0	0
0	0	0	0	1	0	0	0	1	0	0	1	0	0	0
0	0	0	0	1	1	0	0	1	1	0	1	1	0	0
0	0	0	1	0	0	0	1	0	0	1	0	0	0	0
0	0	0	1	0	1	0	1	0	1	1	0	1	0	0
0	0	0	1	1	0	0	1	1	0	1	1	0	0	0
0	0	0	1	1	1	0	1	1	1	1	1	1	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	1
0	0	1	0	0	1	1	0	0	1	0	0	1	0	1
0	0	1	0	1	0	1	0	1	0	0	1	0	0	1
0	0	1	0	1	1	1	0	1	1	0	1	1	0	1
0	0	1	1	0	0	1	1	0	0	1	0	0	0	1
0	0	1	1	0	1	1	1	0	1	1	0	1	0	1
0	0	1	1	1	0	1	1	1	0	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	1	0	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
4 Bit Full Subtractor														
Cin	K	A3	A2	A1	A0	B3	B2	B2	B1	S3	S2	S1	S0	Cout
0	1	0	0	0	0	1	1	1	1	0	0	0	0	0
0	1	0	0	0	1	1	1	1	0					
0	1	0	0	1	0	1	1	0	1					
0	1	0	0	1	1	1	1	0	0					
0	1	0	1	0	0	1	0	1	1					
0	1	0	1	0	1	1	0	1	0					
0	1	0	1	1	0	1	0	0	1					
0	1	0	1	1	1	1	0	0	0					
0	1	1	0	0	0	0	1	1	1					
0	1	1	0	0	1	0	1	1	0					

0	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**					
0	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**					
0	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**					
0	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**					
0	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**					
0	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**					
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**

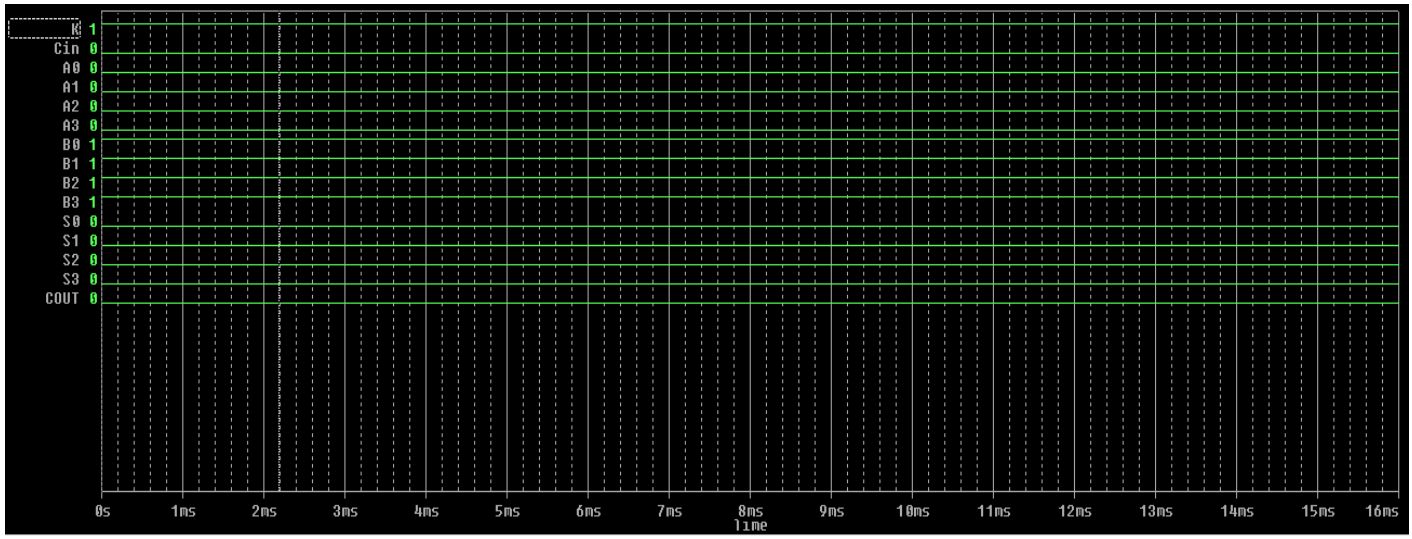
4 bit Full Adder Output Screen:

K=0

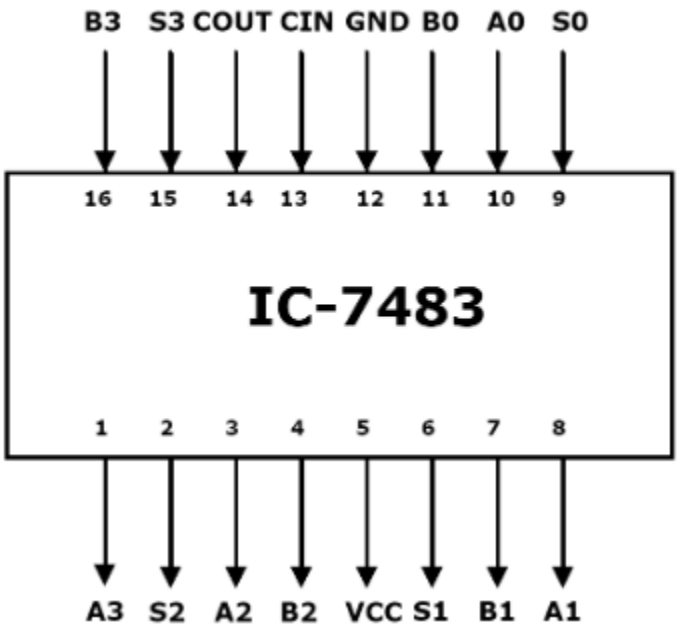


4 bit Full Subtractor Output Screen:

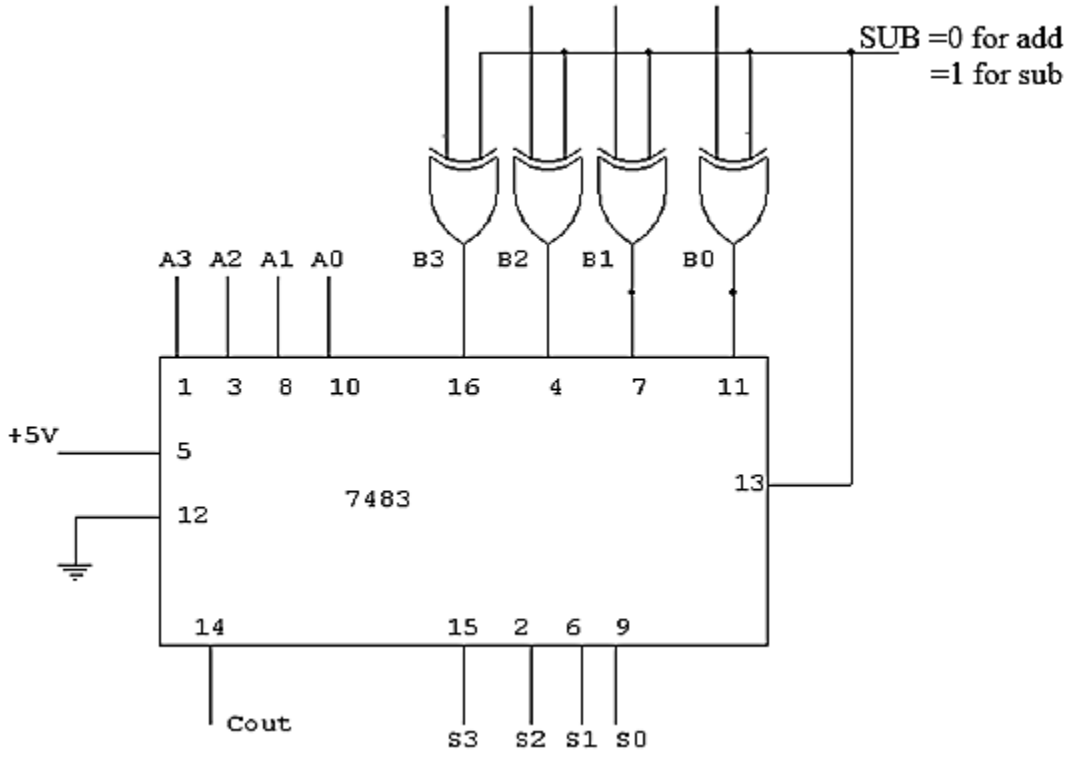
K=1



IC Pin Diagram



Circuit diagram



Result: design a 4 bit full adder and subtractor and simulate the same using basic gates. truth table 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.

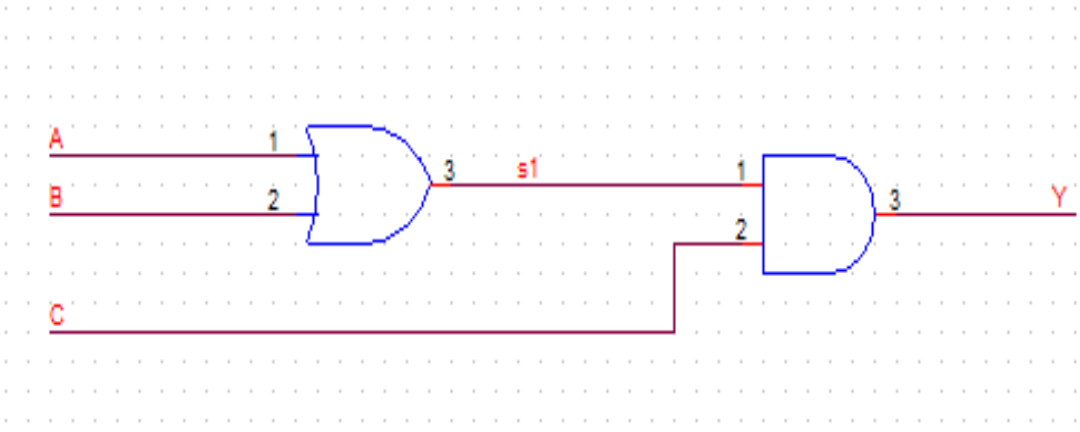
Data Flow:

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			
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

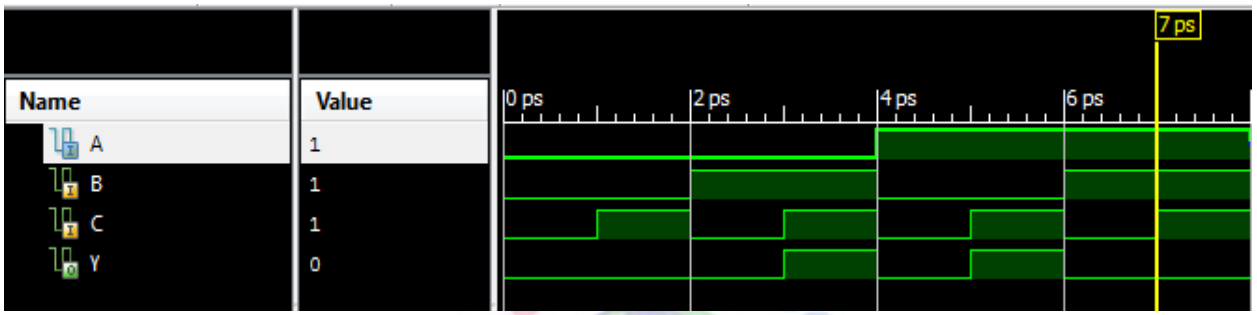
```
module simple_circuit (
input A,
input B,
input C,
output Y,
);
wire s1;
or o1(s1,A,B);
and a1(Y,s1,C)
endmodule
```

- **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,C})
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;

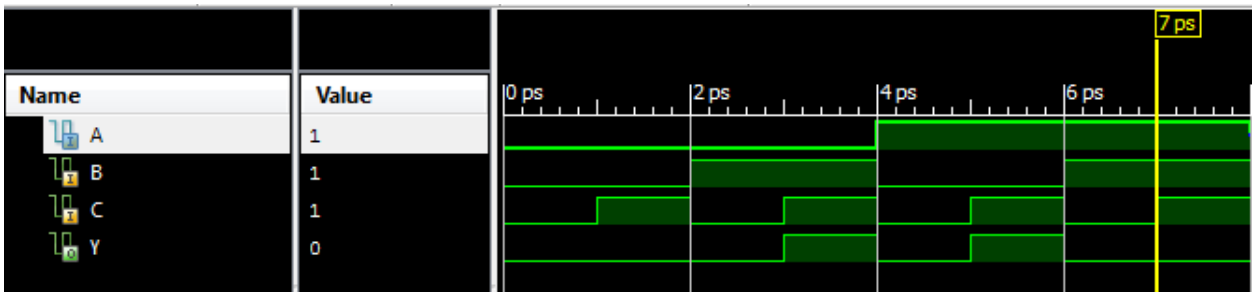
default:Y=0;

endcase

end

endmodule

- **Output Screen**



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

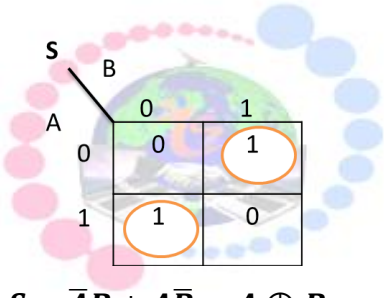
The combinational logic circuit] like half subtractor is used to subtract two single bit digits. It includes three inputs as well as two outputs. The inputs are A, B and Bin whereas the outputs are Borrow & Difference. Please refer to this link to know more about the Full subtractor. Therefore, this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

Half Adder:

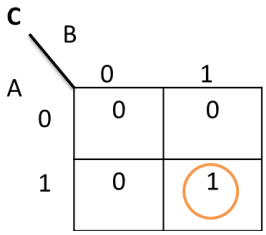
- **Truth table**

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)**

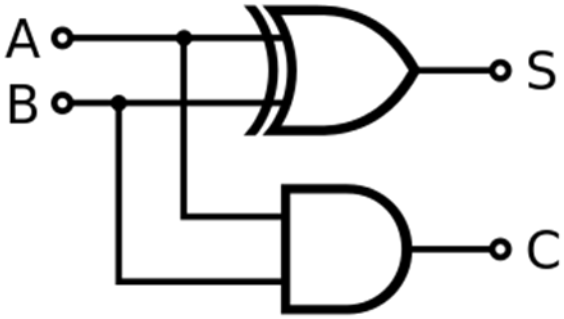


$$S = \bar{A}B + A\bar{B} = A \oplus B$$



$$C = AB$$

- **Circuit diagram**



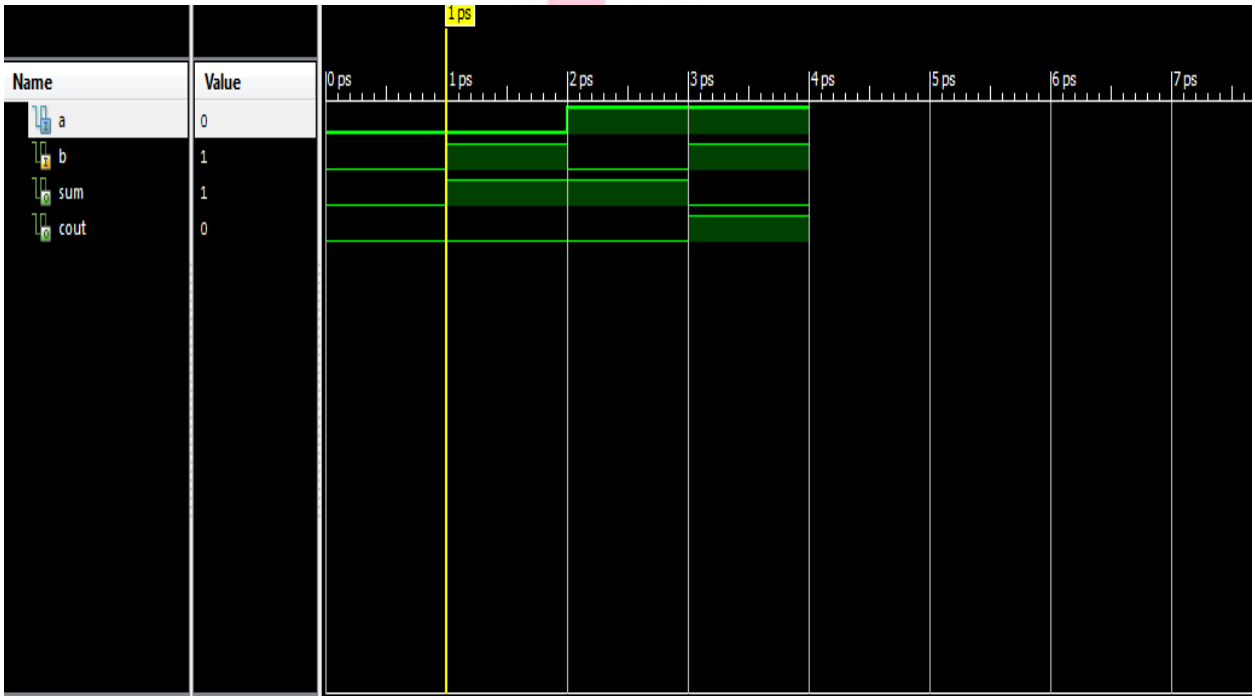
- **Verilog HDL Program**

```
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**

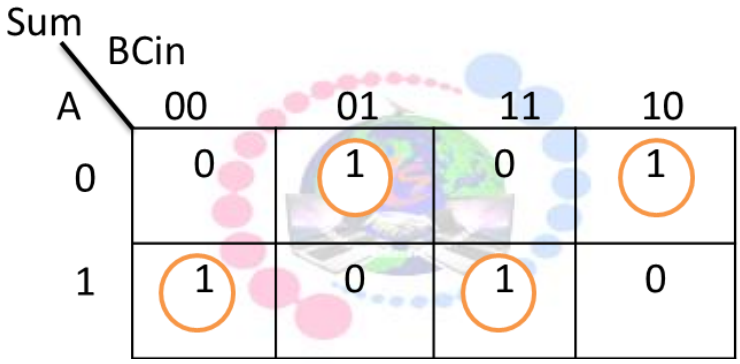


Full Adder

- **Truth table**

Input			Output	
A	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)**

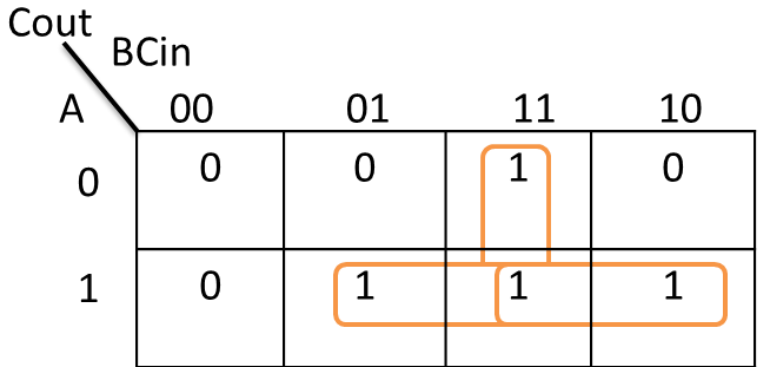


$$Sum = \bar{A}\bar{B}Cin + \bar{A}B\overline{Cin} + A\bar{B}\overline{Cin} + ABCin$$

$$= \bar{A}(\bar{B}Cin + B\overline{Cin}) + A(\bar{B}\overline{Cin} + BCin)$$

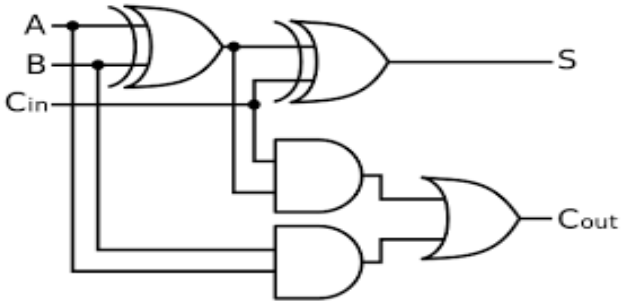
$$= \bar{A}(B \oplus Cin) + A(B \oplus Cin)$$

$$= A \oplus B \oplus Cin$$



$$Cout = AB + BCin + ACin$$

- **Circuit diagram**



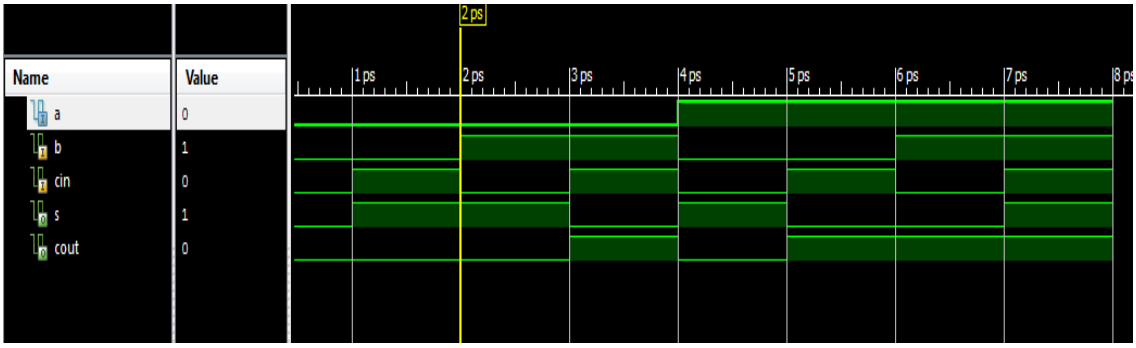
- **Verilog HDL Program**

```
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	Cin
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**

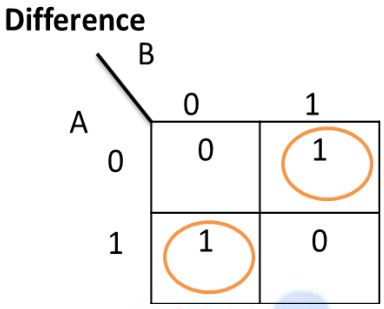


Half Subtractor

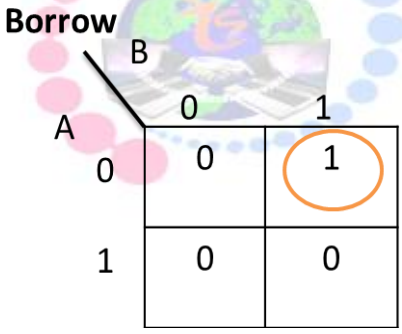
- **Truth table**

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)**

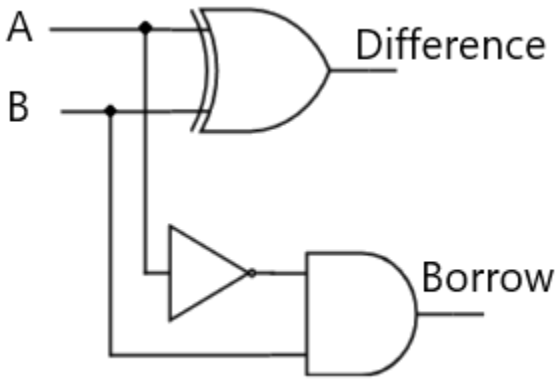


$$\textbf{Difference} = \bar{A}B + A\bar{B} = A \oplus B$$



$$\textbf{Borrow} = \bar{A}B$$

- **Circuit diagram**



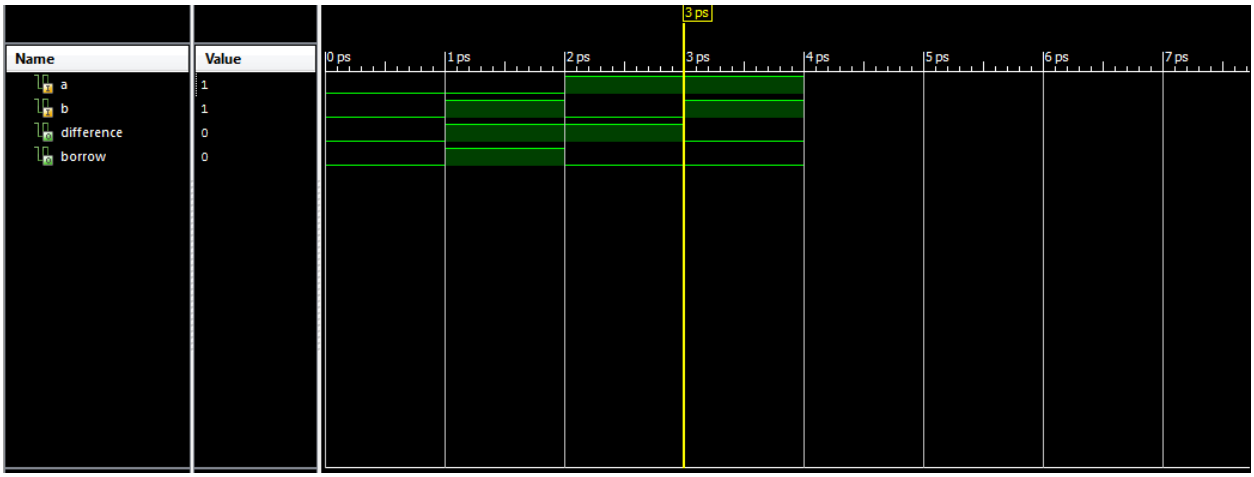
- **Verilog HDL Program**

```
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**

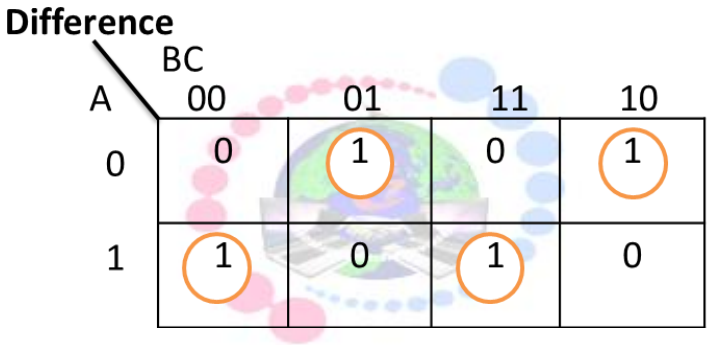


Full Subtractor

- **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)**

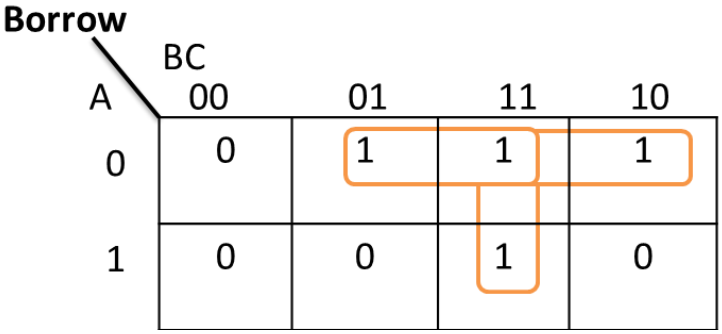


$$\textbf{Difference} = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}C + ABC$$

$$= \bar{A}(\bar{B}C + B\bar{C}) + A(\bar{B}\bar{C} + BC)$$

$$= \bar{A}(B \oplus C) + A(B \oplus C)$$

$$= A \oplus B \oplus C$$



$$\textbf{Borrow} = \bar{A}B + BC + \bar{A}C$$

$$= \bar{A}B + (A \odot B)C$$

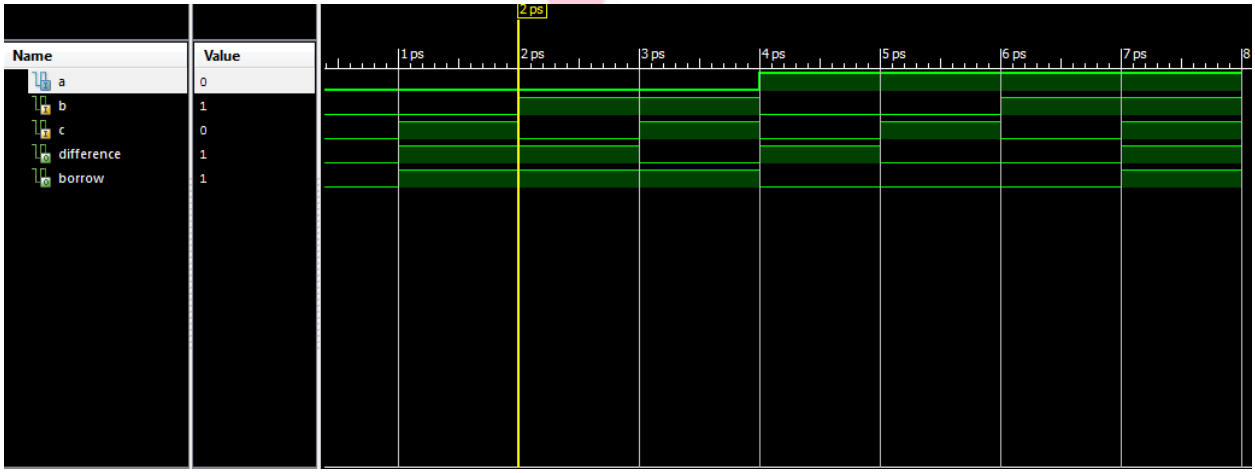
- **Verilog HDL Program**

```
module full_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 0 0 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**

6	**0 1 1 0**	BCD for 6
+ 3	**0 0 1 1**	BCD for 6
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

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 is Invalid

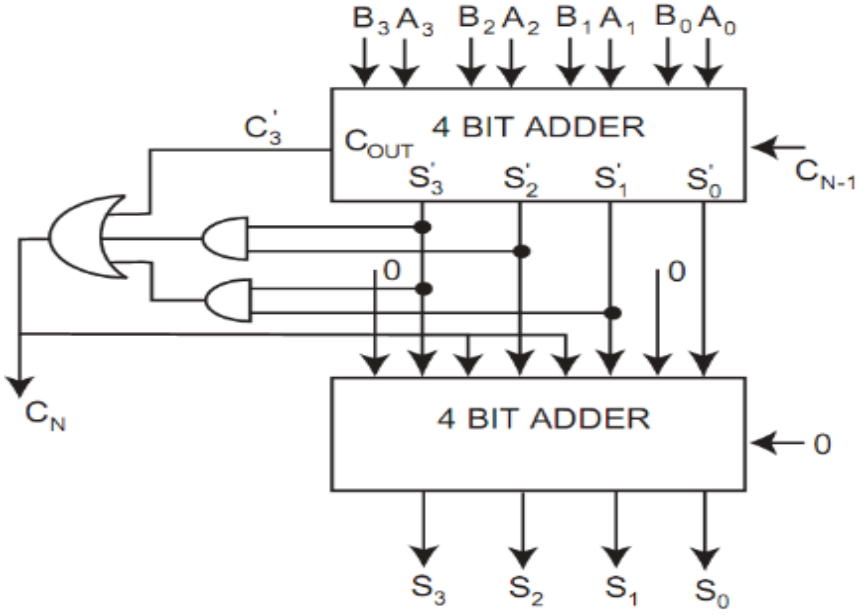
After addition of 6 carry is produced into the second decimal position. Sum equals 9 or less with carry 1

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

- **Truth Table**

Binary Sum						BCD Sum					Decimal
S'_4	S'_3	S'_2	S'_1	S'_0		Carry	S_3	S_2	S_1	S_0	
0	0	0	0	0	SAME CODE	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	NOT SAME CODE	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

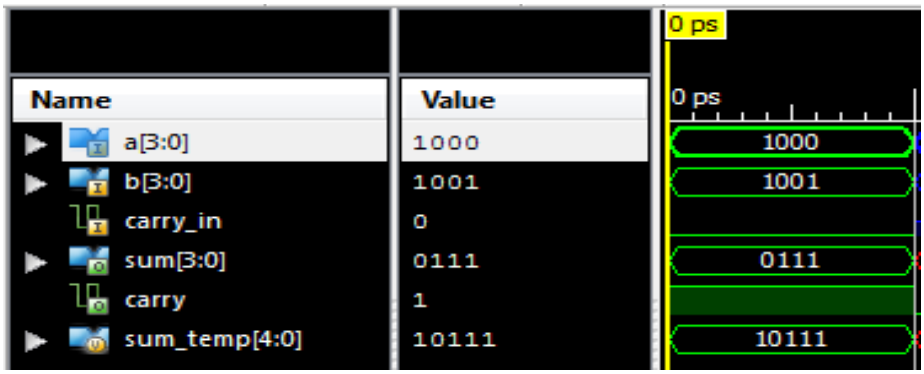
- **Block Diagram of Decimal Adder**



- **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**



Result: Design Verilog HDL to implement Decimal adder 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.

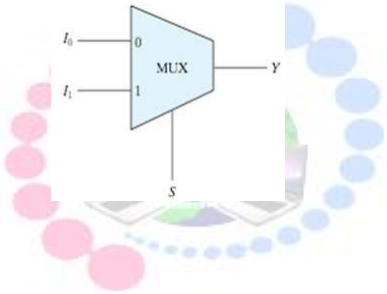
Software requirement: ISE Design Suite 14.7(Xlink), Isim Simulator.

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.

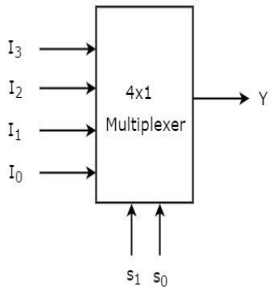
- **2:1 Multiplexer**

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.



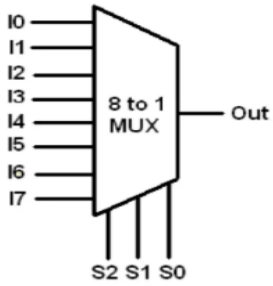
- **4:1 Multiplexer**

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

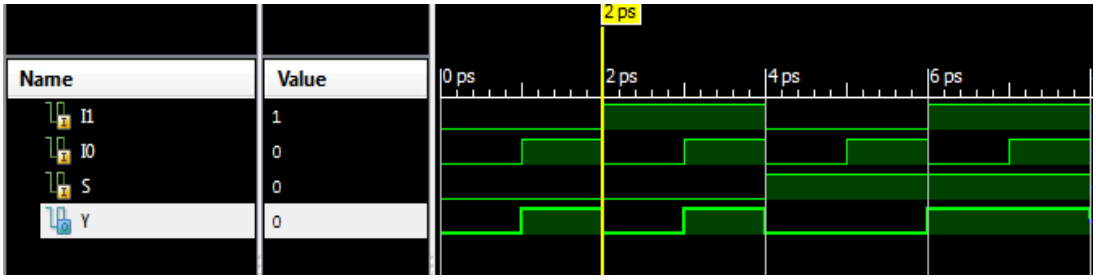
- **Truth Table**

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
```

- **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

- **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
```

- **Output Screen**

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

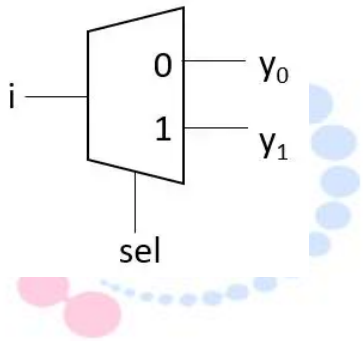
Software requirement: ISE Design Suite 14.7(Xlink), Isim Simulator.

Description:

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

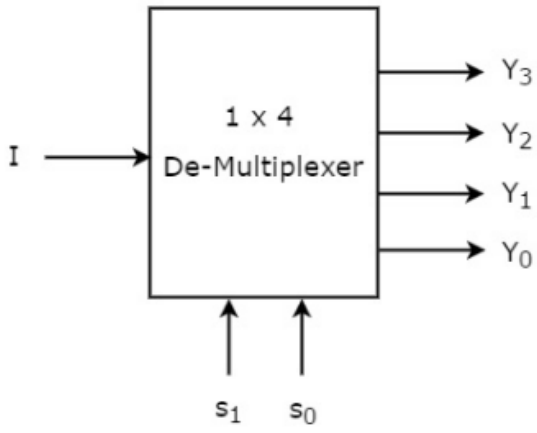
- **1:2 De-multiplexers**

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 (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.



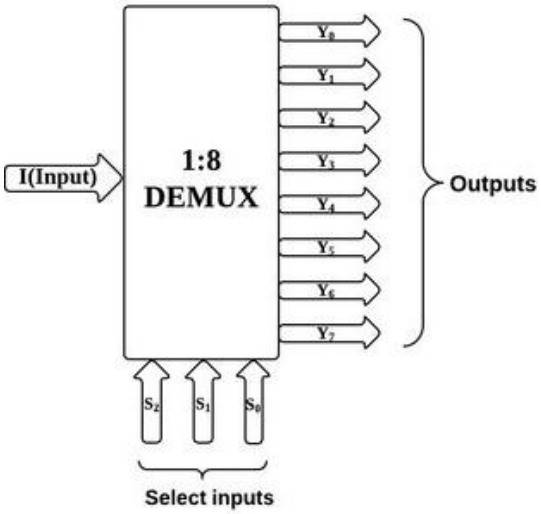
- **1:4 De-multiplexers**

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

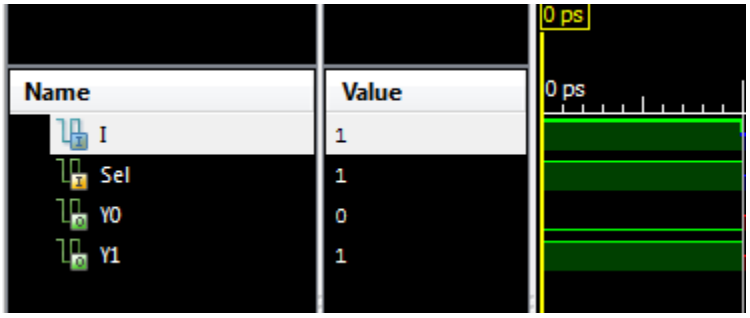
- **Truth Table**

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
```

- **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
```

- **Output Screen**

1:8 De-multiplexers

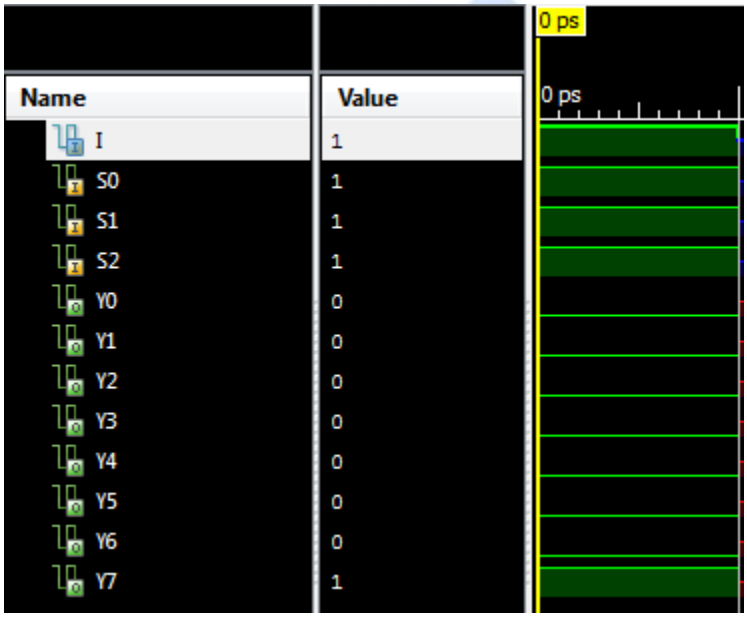
- **Truth Table**

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
```

- **Output Screen**



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:

- **S R 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".

- **J K 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"'.

S R flip flop

- **Truth Table**

Clk	Input		Output		Condition
	S	R	Q	$\bar{Q}$	
↑	X	X	X	X	No Change
↑	0	0	X	X	No Change
↑	0	1	0	1	RESET
↑	1	0	1	0	SET
↑	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
```

- **Verilog HDL Test Bench Program**

```
initial begin
clk=0;
forever  #100 clk = ~clk;
```

end

initial begin

#200; s= 0; r= 0;

#200; s= 0; r= 1;

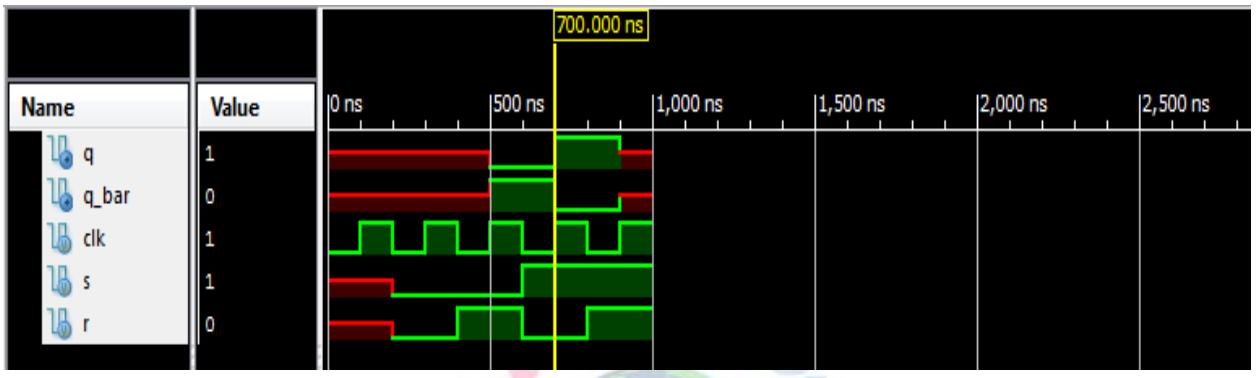
#200; s= 1; r= 0;

#200; s= 1; r=1;

end

endmodule

- **Output Screen**



J K flip flop

- **Truth Table**

Clk	Input		Output		Condition
	J	K	Q	$\bar{Q}$	
↑	X	X	X	X	No Change
↑	0	0	X	X	No Change
↑	0	1	0	1	RESET
↑	1	0	1	0	SET
↑	1	1	0	1	Toggle

- **Verilog HDL Program**

module JK_FF(

input clk,

input j,k,

output reg q,

output q_bar

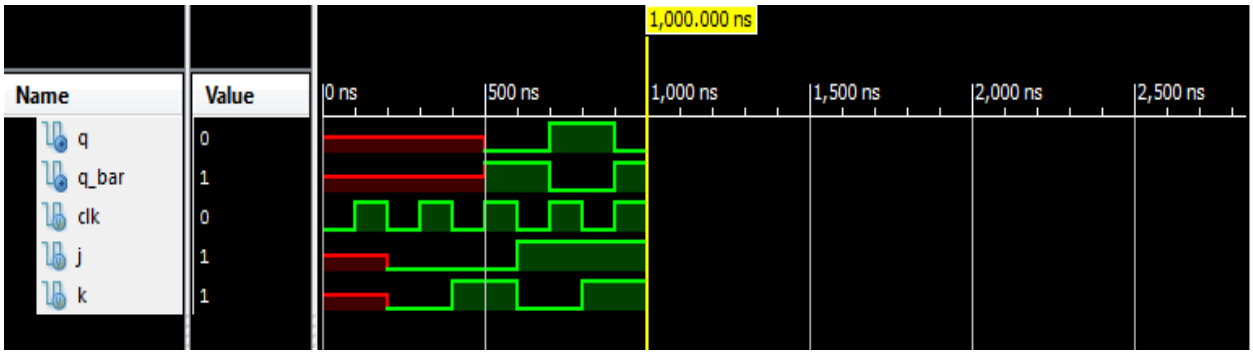
);

```
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
```

- **Verilog HDL Test Bench Program**

```
initial begin
clk=0;
forever  #100 clk = ~clk;
end
initial begin
#200; j= 0; k= 0;
#200; j= 0; k= 1;
#200; j= 1; k= 0;
#200;  j= 1; k=1;
end
endmodule
```

- **Output Screen**



D flip flop

- ***Verilog HDL Program***

```
module D_FF(clk,d,q,qbar);
input clk,d;
output q,qbar;
reg q;
always @ (posedge clk)
begin
q <= d;
end
assign qbar=~q;
endmodule
```

- **Verilog HDL Test Bench Program:**

```
initial begin
clk=0;
forever #100 clk = ~clk;
end
initial begin
#200; D=0;
#200; D=1;
end
endmodule
```

Truth Table:

Clk	Input	Output		Condition
	D	Q	$\bar{Q}$	
↑	X	X	X	No Change
↑	0	0	1	RESET
↑	1	1	0	SET

Output Screen

T flip flop

- ***Verilog HDL Program***

module T_FF(t,q,qbar,clk);

input t,clk;

output q,qbar;

reg q,qbar;

initial q=0;

always@(posedge clk)

begin

if (t==1)

begin

q=~q;

end

else

begin

q=q;

end

qbar=~q;

end

endmodule

- **Verilog HDL Test Bench Program:**

initial begin

clk=0;

forever #100 clk = ~clk;

end

initial begin

#100; t=0;

#100; t=1;

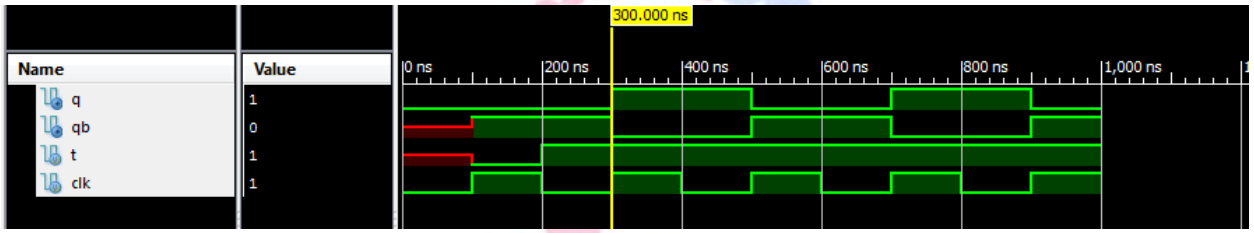
end

endmodule

Truth Table:

Clk	Input	Output		Condition
	T	Q	$\bar{Q}$	
↑	0	0	1	No Changes
↑	1	1	0	Toggle
↑	1	0	1	Toggle

Output Screen:



Result: Design Verilog program for implementing various types of Flip-Flops such as SR, JK D and T and truth table is verified using Verilog HDL Simulator Software.