DAYANANDA SAGAR COLLEGE OF ENGINEERING

(AN AUTONOMOUS INSTITUTE AFFILIATED TO VTU, APPROVED BY AICTE & UGC, ACCREDITED BY NAAC WITH 'A'GRADE.)

DEPARTMENT OF MEDICAL ELECTRONICS

(ACCREDITED BY NBA)

DIGITAL SYSTEM DESIGN
LAB MANUAL

Semester: THIRD

Sub code: 18ML3DLDSD

Name of the Student :

Semester /Section :

USN :

Batch :

VISION AND MISSION OF THE INSTITUTE

VISION

To impart quality technical education with a focus on research and innovation, emphasizing on development of sustainable and inclusive technology for the benefit of society

MISSION

1. To provide an environment that enhances creativity and innovation in pursuit of excellence
2. To nurture teamwork in order to transform individuals as responsible leaders and entrepreneurs
3. To train the students to the changing technical scenario and make them to understand the importance of sustainable and inclusive technologies

VISION AND MISSION OF THE DEPARTMENT

VISION

To develop an excellent centre of progressive quality learning, applied & translational research through inventive collaborations and sustainable solutions to address healthcare related societal challenges

MISSION

1. By creating a conducive atmosphere for continuous learning through increased participation of students and faculty in various academic activities
2. By achieving needful and relevant healthcare solutions through quality education and research
3. By imparting education in the path of ethical and social responsibilities, to work effectively with diverse groups for the benefit of the society

PROGRAM EDUCATIONAL OBJECTIVES

Graduates will be able to

PEO1: Apply knowledge of Medical Electronics to excel in their profession

PEO2: Practice engineering to analyse problems and find solutions in allied domains using multidisciplinary approach

PEO3: Disseminate professional skills for ethical and societal responsibilities

PEO4: Engage in lifelong learning and contribute in the field of engineering research

PROGRAM SPECIFIC OUTCOMES (PSO)

Graduates will be able to:

PSO1: Apply the knowledge of electronics for solving diverse problems of Medical Instrumentation

PSO2: Analyse and implement different techniques in Medical Image and Signal Processing domains catering to Design, Research and Development

PSO3: Provide sustainable solutions in health care and its allied fields by imbibing managerial and techno-social values

DIGITAL SYSTEM DESIGN LAB

Course code : 18ML3DLDSD **Semester: III**
L: P: T: S : 0: 2: 1: 0 **Credits: 02**
Exam Hours: 03 **CIE Marks: 50**
Total Hours : 42 **SEE Marks: 50**
Course Coordinator: Dr. Murigendrayya M. Hiremath
Prof. Padma C R

Course Objectives:

This laboratory course enables students to get

1. Practical experience in design, realization and verification of various number systems and its application in digital design
2. The ability to understand, analyze and design various combinational and sequential circuits. Ability to identify basic requirements for a design application
3. To develop skill to build, and troubleshoot digital circuits

Course Outcomes:

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

CO1	Design and evaluate combinational circuits like Adders, Subtractors, Decoders, Multiplexers and Comparators.
CO2	Design and evaluate sequential circuits like counters using Gates and ICs
CO3	Apply the knowledge of shift registers into applications like Ring counter, Johnson counter and Sequence generator.
CO4	Realize and Implement the Digital Logic using FPGA/CPLD
CO5	Develop and prototype the custom hardware for medical and other digital systems
CO6	Interface the input and output peripherals to the digital system

Mapping of Course outcomes to Program outcomes:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	-	3	2	1	1	-	-	1	1	-	-	2	-	1
CO2	3	3	3	3	3	1	-	-	1	1	-	1	1	1	-
CO3	3	2	1	1	3	1	-	-	1	1	-	1	1	1	-
CO4	3	3	3	3	3	1	-	-	1	-	-	1	1	1	-
CO5	3	3	3	3	3	1	-	-	1	-	-	1	1	1	-
CO6	3	3	3	3	3	1	-	-	1	-	-	1	1	1	1

Course Content:

Exp	Contents of the Experiment	H	CO
1	Realization of Half/Full adder and Half/Full Subtractors using logic gates.	03	CO1
2	Code conversion (i) BCD to Excess-3 code conversion and viceversa. (ii) Realization of Binary to Gray code conversion and vice versa	03	CO1
3	Realization of MUX and DEMUX using basic gates and IC's 74139 and 74153	03	CO1
4	Realization of One/Two bit comparator and study of 7485 magnitude comparator.	03	CO1
5	Truth table verification of Flip-Flops: (i) JK Master slave (ii) T type and (iii) D type.	03	CO2
6	Design and Realization of 3 bit counters as a sequential circuit, Ring counter and Johnson counter	03	CO3
7	Shift left, Shift right, SIPO, SISO, PISO, PIPO operations using 7495	03	CO3
8	Write a HDL code to describe the functions of a Adders / Subtractors using three modeling styles: Behavioral, structural and data flow	03	CO4
9	Design and implement ALU using the Verilog	03	CO5
10	Develop the HDL code for the following flip-flops - SR,JK, D and T.	03	CO6
11	Write HDL code to display messages on the given seven segment display, LCD and accepting Hex key padinput data.	03	CO6
12	Write HDL code to generate different waveforms (Sine, Square, Triangle, Ramp etc.,) using DAC (with change in frequency and amplitude)	03	CO6

ASSESSMENT PATTERN:

CIE –Continuous Internal Evaluation (50 Marks)

SEE –Semester End Examination (50 Marks)

DAYANANDA SAGAR COLLEGE OF ENGINEERING

DEPARTMENT OF MEDICAL ELECTRONICS

DO's & DONT's

DO's

- **Adhere and follow timings, proper dress code with appropriate foot wear.**
- **Bags and other personal items must be stored in designated place.**
- **Come prepare with the viva, procedure, and other details of the experiment.**
- **Secure long hair, Avoid-loose clothing , Deep neck and sleeveless dresses**
- **Do check for the correct ranges/rating and carry one meter/instrument at a time**
- **Inspect all equipment/meters for damage prior to use**
- **Conduct the experiments accurately as directed by the teacher.**
- **Immediately report any sparks/ accidents/ injuries/ any other troublesome incident to the faculty /instructor.**
- **Handle the apparatus/meters/computers gently and with care**
- **In case of an emergency or accident, follow the safety procedure.**
- **Switch OFF the power supply after completion of experiment**

DONT's

- **The use of mobile/ any other personal electronic gadgets is prohibited in the laboratory.**
- **Do not make noise in the Laboratory & do not sit on experiment table.**
- **Do not make loose connections and avoid overlapping of wires**
- **Don't switch on power supply without prior permission from the concerned staff.**
- **Never point/touch the CRO/Monitor screen with the tip of the open pen/pencil/any other sharp object.**
- **Never leave the experiments while in progress.**
- **Do not insert/use pen drive/any other storage devices into the CPU?**
- **Do not leave the Laboratory without the signature of the concerned staff in observation book**

Table of Contents

INTRODUCTION

Logic gate is the basic building block of all digital circuit. It consists of one or more input and one output. It finds application in combinational as well as Sequential circuit. There are basically two types of Logic gate; Basic and Universal Logic gate. The basic gates are AND (Product), OR (Add) and NOT (Inverter). The Universal gates are NAND & NOR gates. These basic logic gates are implemented as small-scale integrated circuits (SSICs) or as part of more complex medium scale (MSI) or very large-scale (VLSI) integrated circuits. Digital IC gates are classified not only by their logic operation, but also the specific logic-circuit family to which they belong. Each logic family has its own basic electronic circuit upon which more complex digital circuits and functions are developed. The following logic families are the most frequently used.

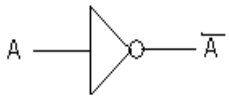
Logic gates

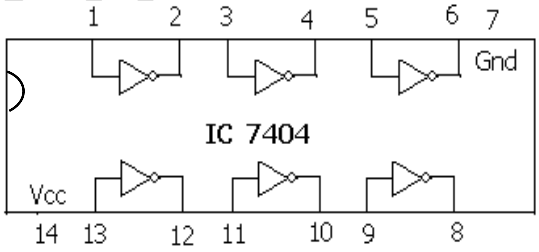
NOT GATE

Truth Table

I/P (A)	O/P $\overline{(A)}$
0	1
1	0

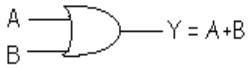
Symbol

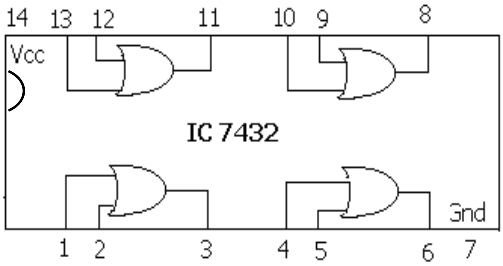




OR GATE

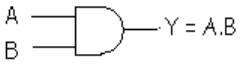
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

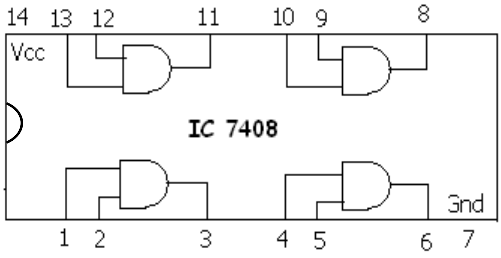




AND GATE

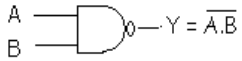
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

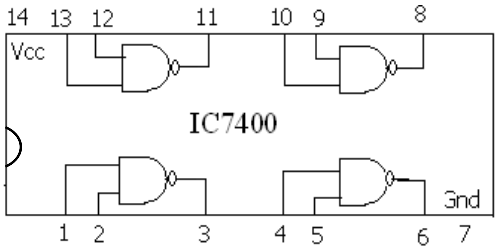




NAND GATE

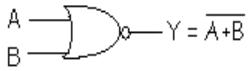
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

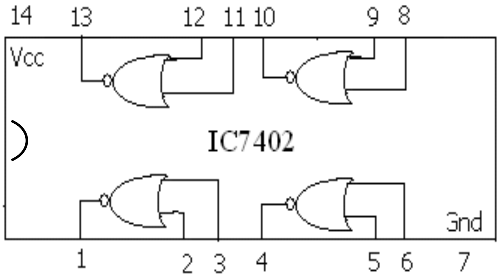




NOR GATE

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

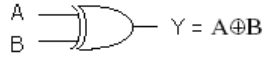


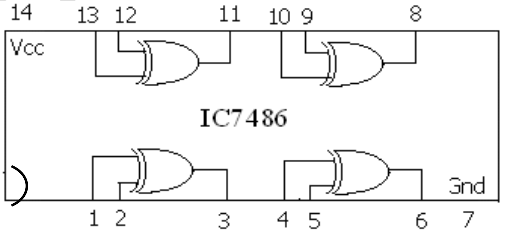


EX-OR GATE

Truth Table Symbol IC7486

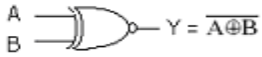
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

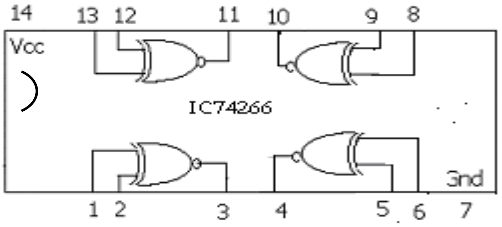




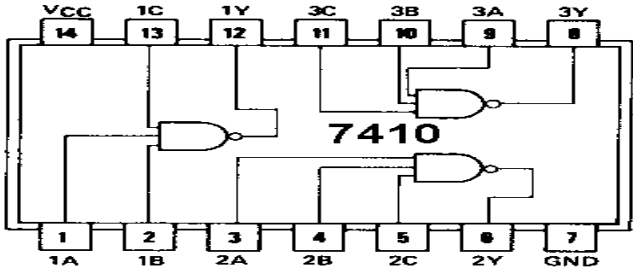
EX-NOR GATE (IC74266)

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

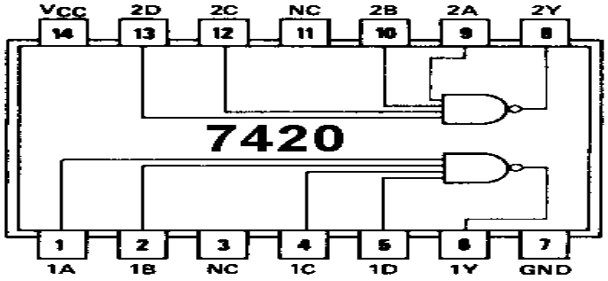




THREE INPUT NAND GATE



FOUR INPUT NAND GATE



Canonical Forms (Normal Forms): Any Boolean function can be written in disjunctive normal form (sum of min-terms) or conjunctive normal form (product of max-terms). To minimize a Boolean expression we can employ any one of the following techniques: Boolean algebra and Karnaugh maps. There are two forms of Boolean expression available.

1) Sum of product (SOP): Is one which consists of sum of all minterms (AND terms) and each minterm is equal to row in a Truth Table where output is 1.Ex: ABC+ABC1+AB1C

2) Product of Sum (POS): Is one which consists of product of all maxterms (OR terms) and each minterm is equal to row in a Truth Table where output is 0.

Ex (A+B+C)(A+B+C^1)+(A+B^1+C).

Karnaugh Map is a graphical representation for solving Boolean functions. It consists of Squares= 2^n , where n is number of variables used in Karnaugh. There are 2, 3, 4, 5, 6 etc variable Karnaugh Map is available. Normally while using Karnaugh Map, the Boolean expression is converted into Canonical form. This method provides simplified form of Boolean Expression thereby reducing the usage of number of gates required to construct circuit.

SIMPLIFICATION & REALIZATION OF BOOLEAN EXPRESSION:-

For Ex. Simplify the Expression 1) $y = \overline{A}BC + AB + \overline{B}C$

2) $y = F(A,B,C) = \sum m(0,1,4,5,7)$ (SOP form)

3 $y = F(A,B,C) = \prod M(2,3,6)$ (POS form)

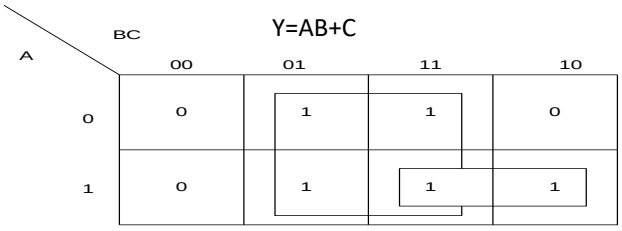
Method 1 using Boolean Theorems for expression 1:

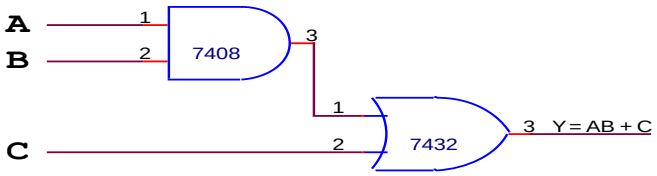
Expand the expression by multiplying the 2nd term with $(C + \overline{C})$ and 3rd term with $(A + \overline{A})$ that is convert the expression into Canonical form as shown below

$$y = \overline{A}BC + AB(C + \overline{C}) + \overline{B}C(A + \overline{A})$$
$$= \overline{A}BC + ABC + AB\overline{C} + A\overline{B}C + \overline{A}\overline{B}C \text{.....1}$$
$$= \overline{A}C(B + \overline{B}) + AC(B + \overline{B}) + AB\overline{C}$$
$$= \overline{A}C + AC + AB\overline{C}$$
$$= C(A + \overline{A}) + AB\overline{C} = C + AB$$

Method 2 using K-map for expression 1:

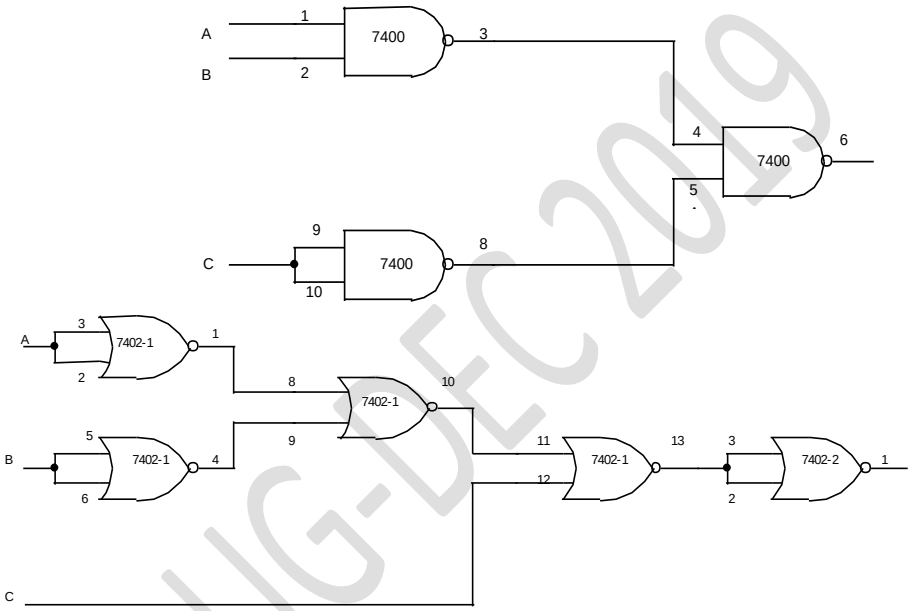
Expression 1 can be mapped on to K-map as shown below.





Truth Table:

INPUTS			OUT PUT
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Why HDL?

Traditional Discrete Integrated circuits (ICs) method is insufficient to realize the complex systems. The Complex systems are usually realized using high code density, programmable chips, such as Application Specific Integrated Circuits (ASICs), Field Programmable Gate Arrays (FPGAs), and require sophisticated CAD tools. HDL is the integral part of CAD tools. HDL offers the designer a very efficient tool for implementing and synthesizing designs on chips. Debugging the design is easy, since HDL packages implement simulators and test benches. Hardware Description Language is to represent digital hardware in a specific language format similar to software languages, such as C. Hence HDL is the right choice for complex digital system designs.

Introduction to Verilog

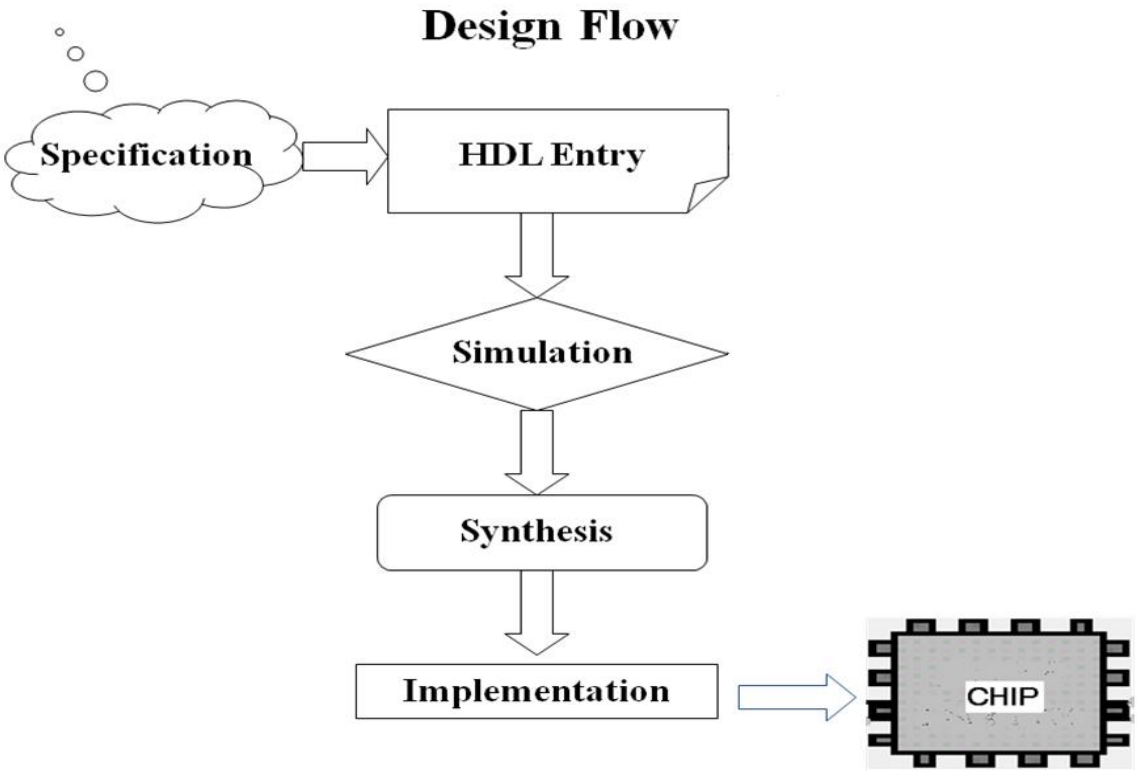
Verilog is a Hardware Description Language was developed in 1984 by Phiip Moorby. It become property of Gateway Design Automation, which was later acquired by Cadence Design System. In 1990 Cadence released Verilog to public. Open Verilog International (OVI) was formed to control the language specifications. In 1993 IEEE accepted OVI Verilog as a standard.

- Verilog HDL is an IEEE standard - 1364.
- Verilog code structure is based on C software language.
- Work is currently underway to define analog extensions to Verilog.
- Specification: It describes the system operation.

Design Flow

- HDL Entry: Entering the functionality of the design using Hardware Description language (HDL) VHDL or VERILOG.

- Synthesis: Converts HDL code in to logical gates or components. These logical gates and components can be downloaded into a chip.
- Simulation: A process of applying stimuli to a design over time and producing, recording and analysing the corresponding responses from the model.
- Implementation: It includes place and routing, and defining constructions.

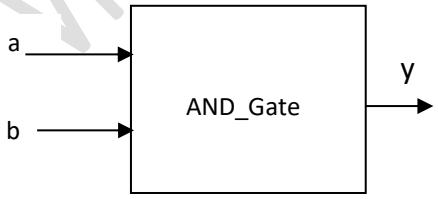


Structure of the Verilog Module

HDL Modules follow the general structure of software languages such as C.

Example:

y = a.b



The verilog module has a declaration and a body. In the declaration, name, inputs, and outputs of the module are listed. The body shows the relationship between the inputs and the outputs.

Example:

```
module AND_Gate (a, b, y);
input a;
input b;
output y;
assign y = a & b; // statement
endmodule
```

The declaration of the module starts with the predefined word module module, followed by a user selected name. The name of the module is user selected AND_Gate. The names of the inputs (a, b) and output (y) are written inside parenthesis and are separated by commas. The semicolon (;) performs function as line separator. For comments // or (/*….*/) can be used.

ISE Quick Start Tutorial

The ISE Quick Start Tutorial provides an understanding of how to create, verify, and implement a design. This tutorial contains the following sections:

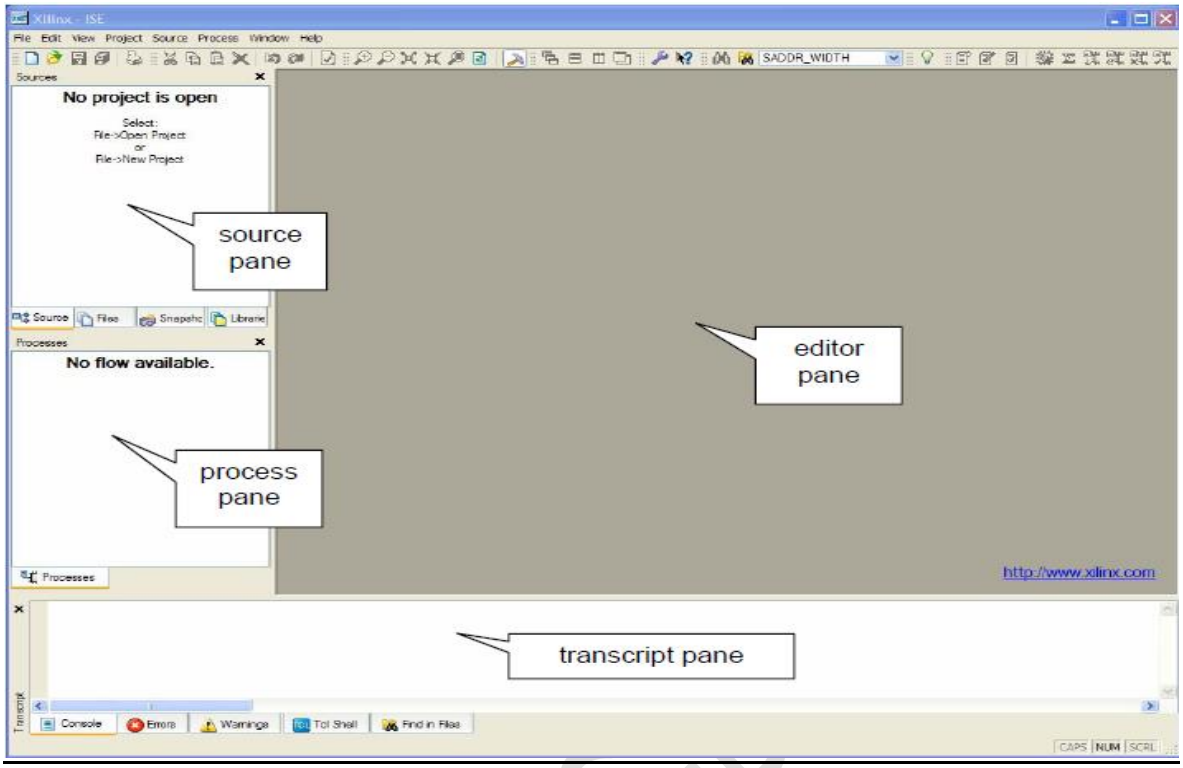
1. Getting Started : Starting the ISE Software
2. Create a New Project
3. Create an HDL Source
4. Design Simulation
5. Syntheses
6. Assigning Pin Location Constraints
7. Implement Design and Verify Constraints
8. Download Design to the Spartan™-3 Board

1. Getting Started:

You start ISE by double-clicking the icon 

on the desktop. This will bring up an empty project window as shown below. The window has four panes:

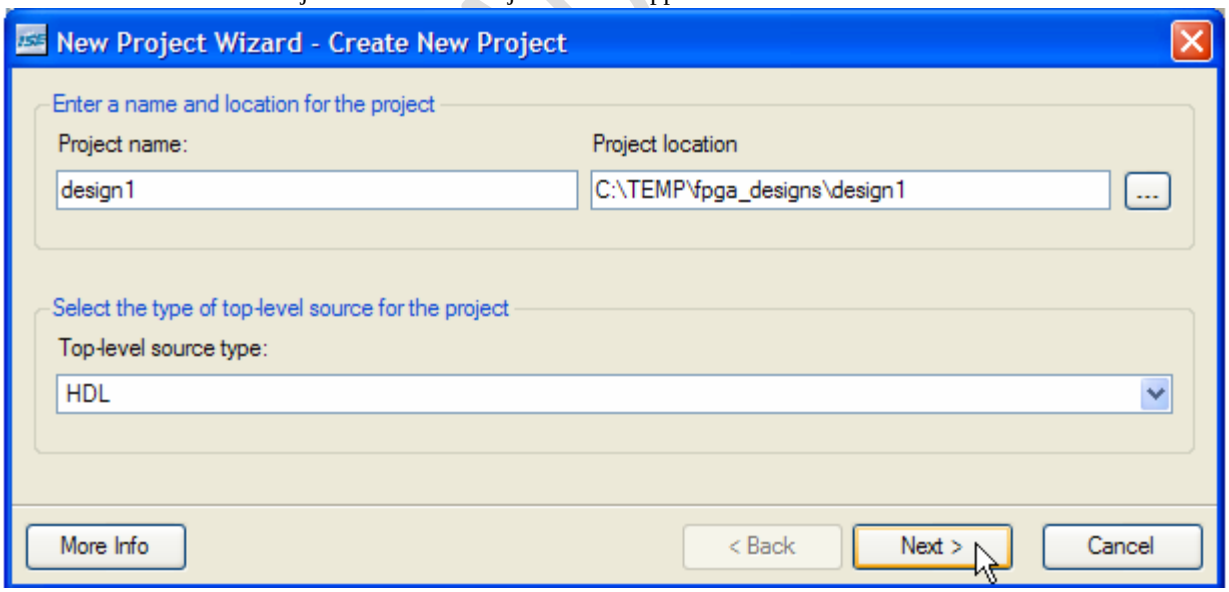
A source pane that shows the organization of the source files that make up your design. There are four tabs so you can view the functional modules, source files, different snapshots (or versions) of your project, or the HDL libraries for your project. **A process pane** that lists the various operations you can perform on a given object in the source pane. **A transcript pane** that displays the various messages from the currently running process. **An editor pane** where you can enter HDL code, schematics, state diagrams, etc.



1. Create a New Project

To start your design, create a new project by referring the following steps:

2.1. Select File ➔ New Project... The New Project Wizard appears.



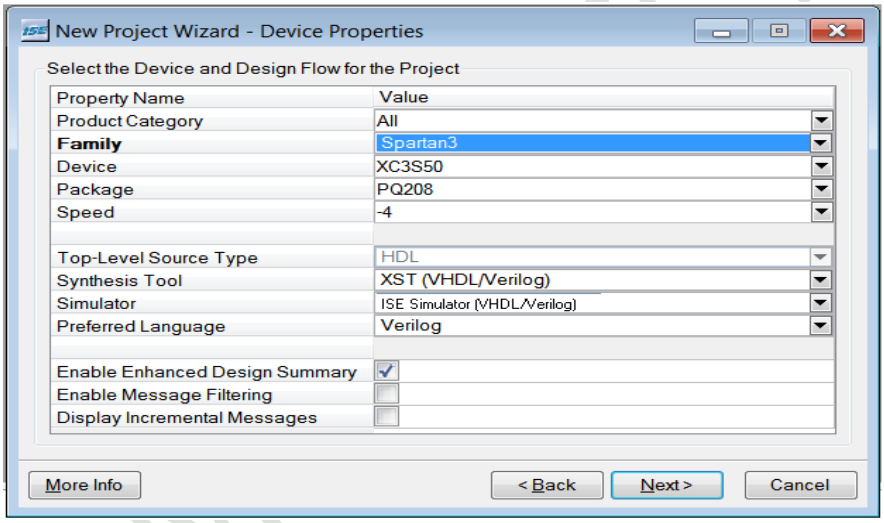
2.2: Type <Project Name> in the Project Name field as *design1* (you may choose differently). Enter or browse to a location (directory path) for the new project. A tutorial subdirectory is created automatically.

2.3: Verify that HDL is selected from the Top-Level Source Type list.

2.4: Click Next to move to the device properties page.

2.5: Now you need to tell ISE what FPGA or CPLD you are going to use for your design. The device family, family member, package and speed grade for the FPGA and CPLD Board are shown below:

For FPGA	For CPLD
• Product Category : *All* • Family : *Spartan3* • Device : *XC3S50* • Package : *PQ208* • Speed Grade : -4 • Top-Level Source Type *: HDL* • Synthesis Tool : *XST (VHDL/Verilog)* • Simulator: *ISE Simulator (VHDL/Verilog)* • Preferred Language : *Verilog (or VHDL)* • Verify that Enable Enhanced Design Summary is selected. Leave the default values in the remaining fields.	• Product Category : *All* • Family : *XC9500CPLDs* • Device : *XC9572* • Package : *PC84* • Speed Grade : *-15* • Top-Level Source Type : *HDL* • Synthesis Tool : *XST (VHDL/Verilog)* • Simulator*: ISE Simulator (VHDL/Verilog)* • Preferred Language : *Verilog (or VHDL)* • Verify that Enable Enhanced Design Summary is selected. Leave the default values in the remaining fields.

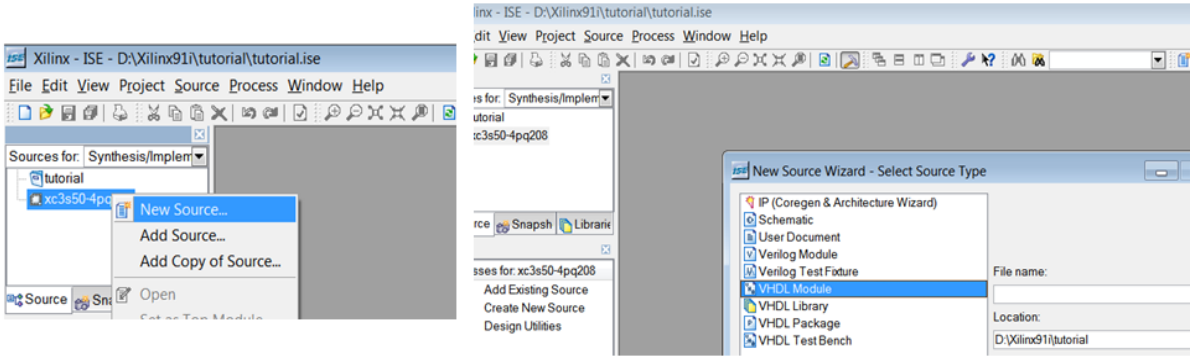


2.6. Click Next to proceed to the Create New Source window in the New Project Wizard. At the end of the next section, your new project will be complete.

2. Create an HDL Source

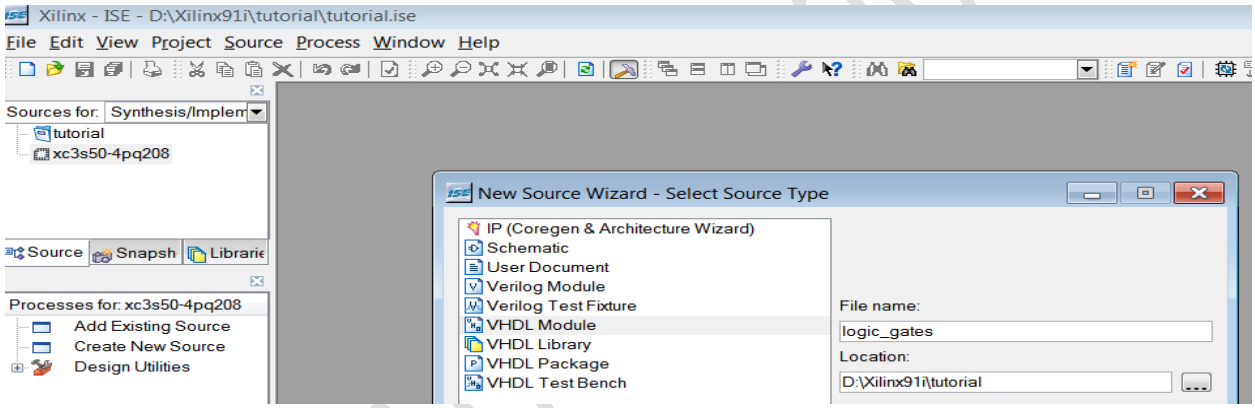
In this section, you will create the top-level HDL file for your design. Determine the language that you wish to use for the tutorial. Then, continue either to the "Creating a VHDL Source" section below, or to the "Creating a Verilog Source" section. Create a VHDL source file for the project as follows:

3.1 Click the New Source button in the New Project Wizard.

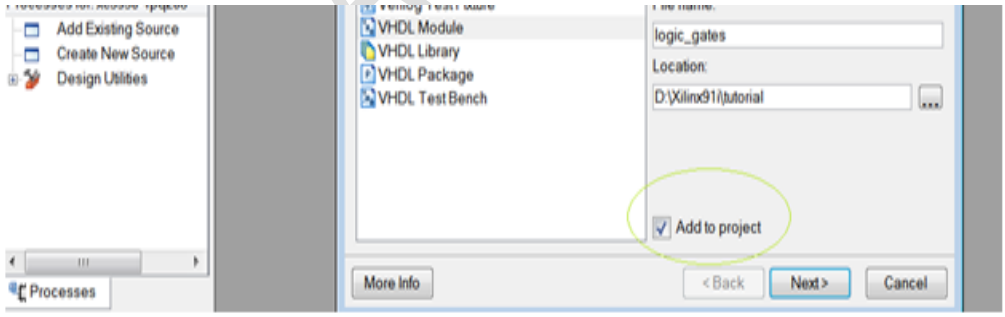


3.2 Select VHDL Module as the source type.

3.3 Type in the file name (for example as logic_gates).

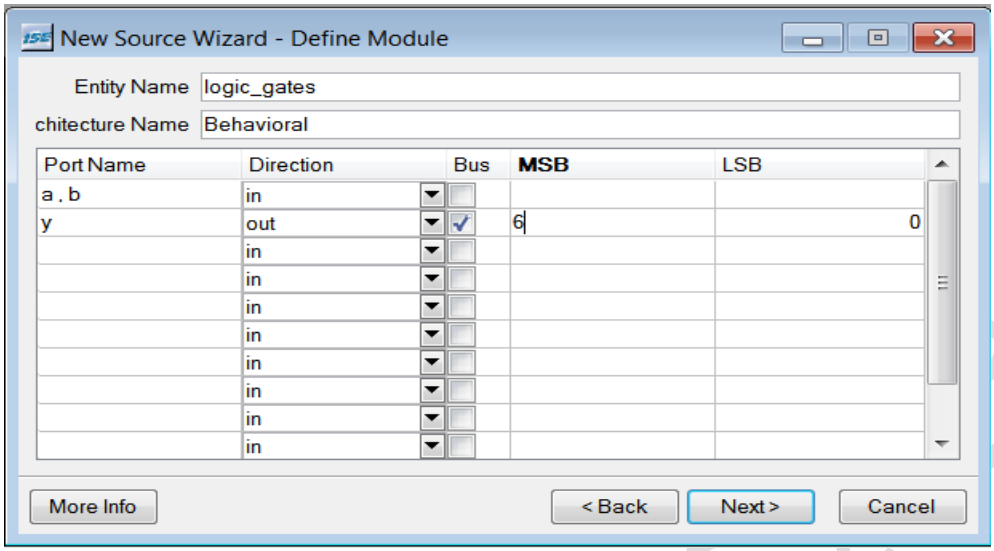


3.4 Verify that the Add to project checkbox is selected.



3.5 Click Next.

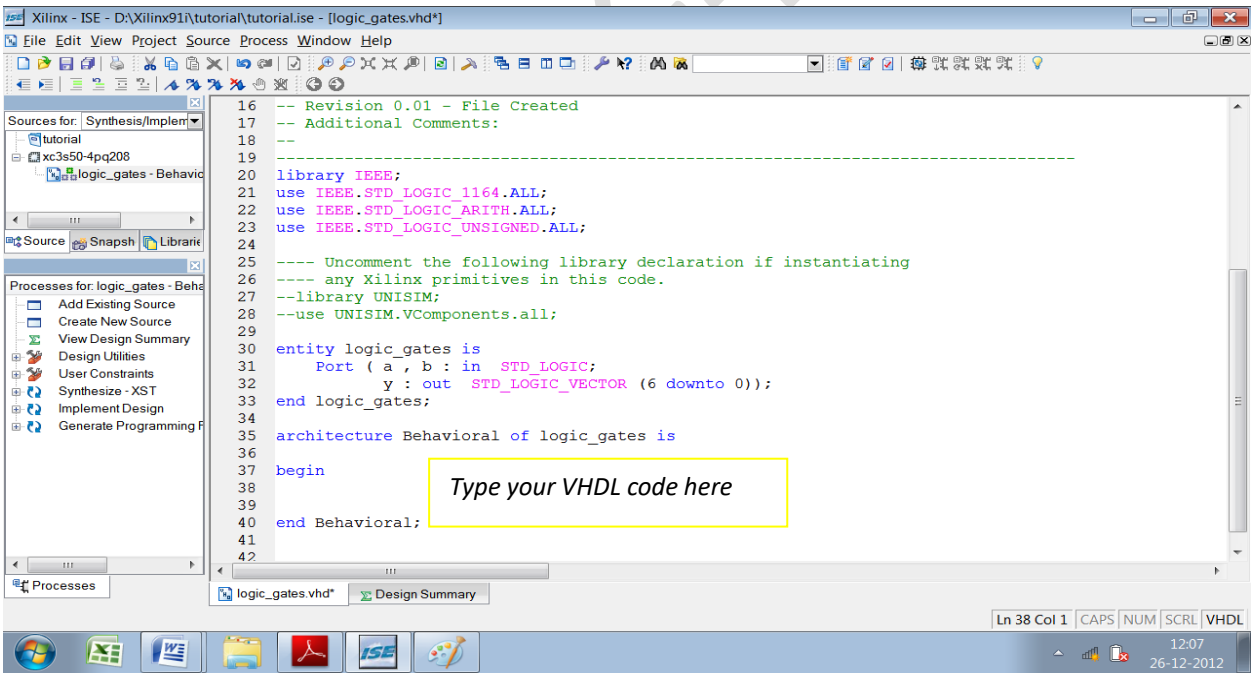
3.6 Declare the ports for your design by filling in the port information as shown below: Refer your VHDL code and block diagram to declare the port details *(example: for full adder designs a, b, and cin are declared as input port and sum and cout are declared as output port).*



3.7 Click next, and then Finish in the New Source Wizard - Summary dialog box to complete the new source file template.

3.8 Click Next, then Next, then Finish.

The source file containing the entity/architecture pair displays in the Workspace, and the logic_gates displays in the Source tab, as shown below:



3.9 Final Editing of the VHDL Source: Type your VHDL code, for example program written for all logic gates as given bellow:

```
y(0)<= NOT a;
y(1)<= a AND b;
y(2)<= a OR b;
y(3)<= a NAND b;
```

```
y(4)<= a NOR b;
y(5)<= a XOR b;
y(6)<= a XNOR b;
```

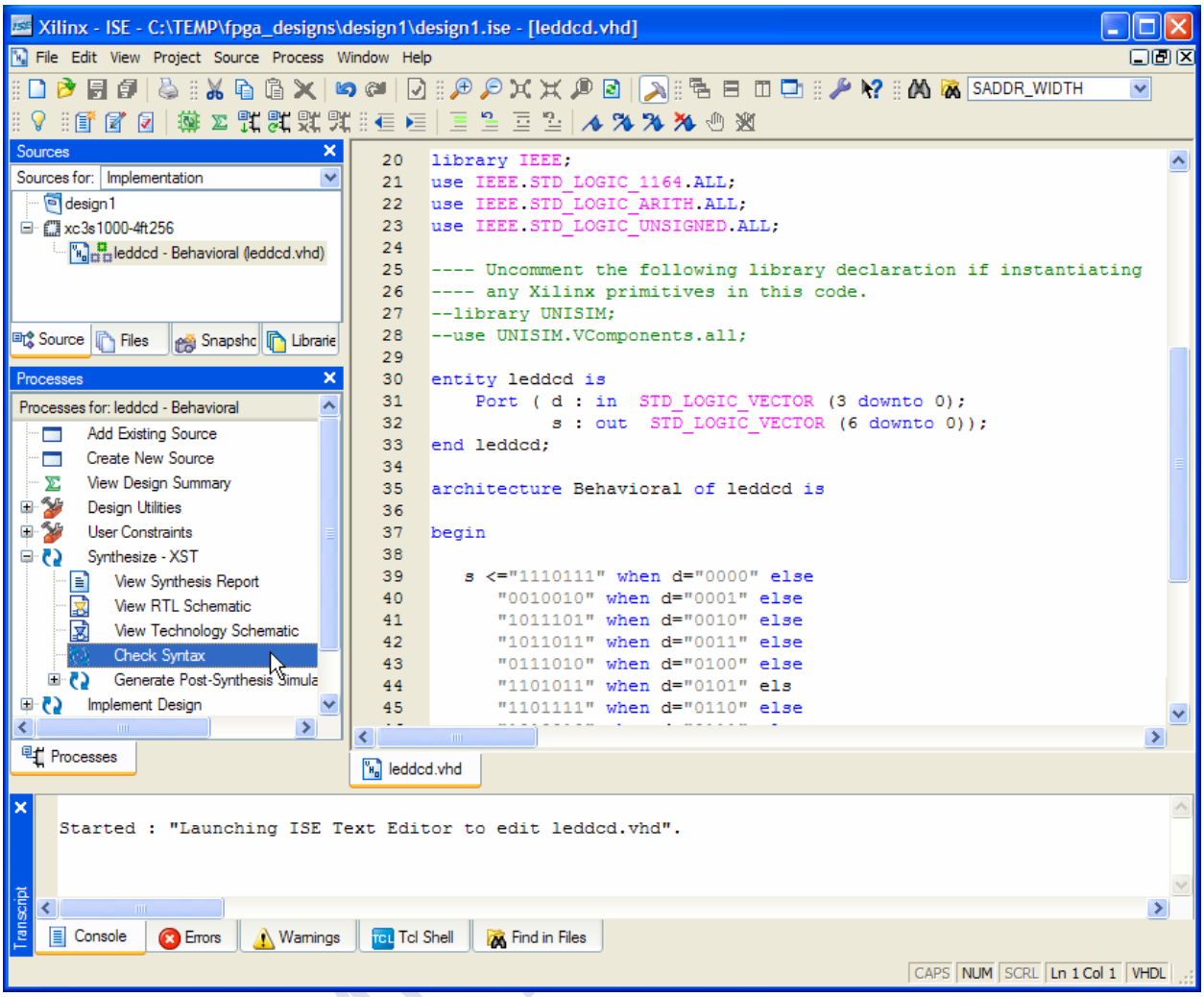
3.10 Save the file by selecting File ->Save. When you are finished, the logic_gates source file will look like the following:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity logic_gates is
   Port (a, b: in STD_LOGIC;
         y: out  STD_LOGIC_VECTOR (6 downto 0));
end logic_gates;

architecture Behavioral of logic_gates is

begin
         y(0)<= NOT a;
         y(1)<= a AND b;
         y(2)<= a OR b;
         y(3)<= a NAND b;
         y(4)<= a NOR b;
         y(5)<= a XOR b;
         y(6)<= a XNOR b;
end Behavioral;
```

3.11 Checking the Syntax

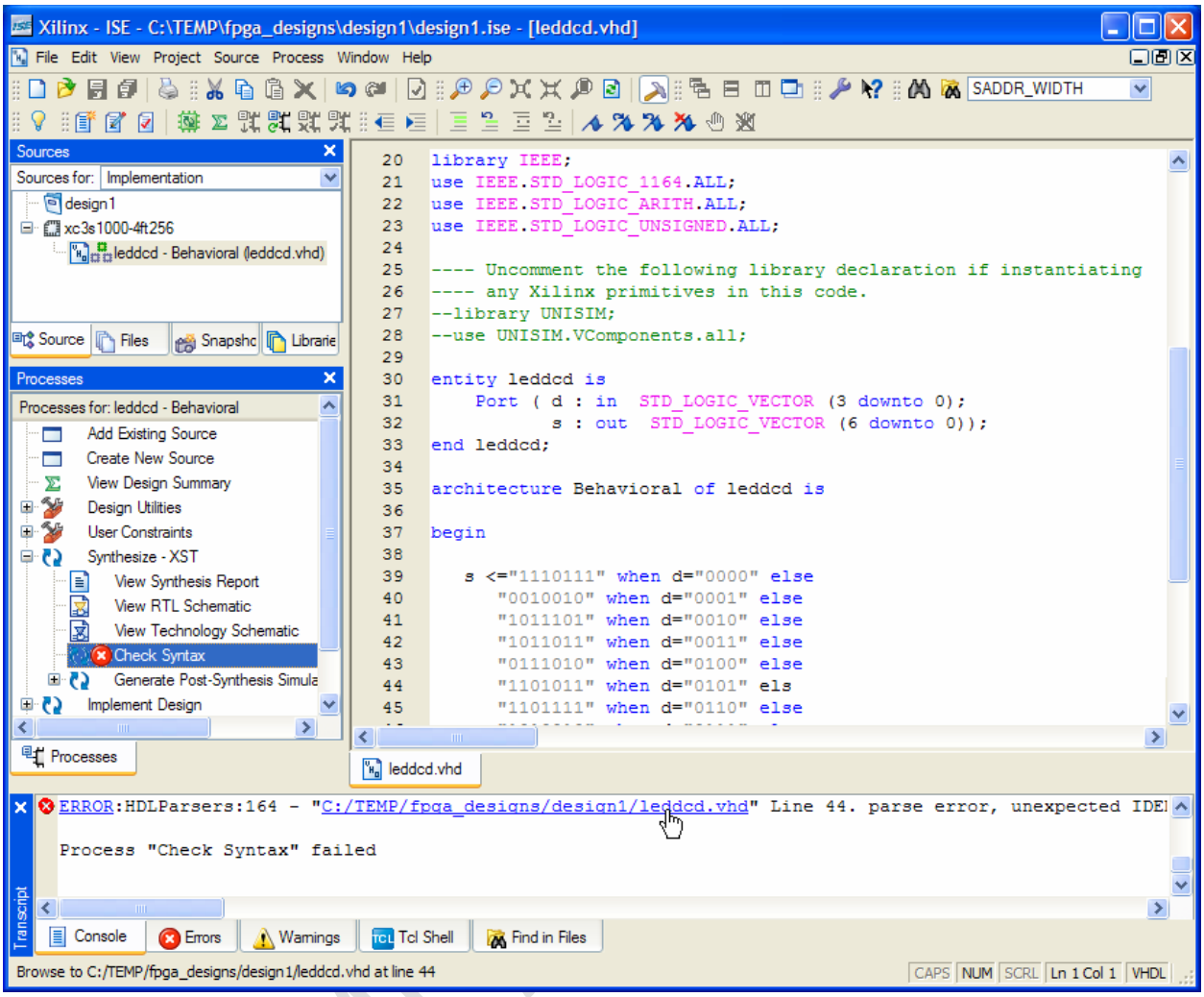
You can check for errors in our HDL code by highlighting the <hdalcode> object in the Sources pane and then double-clicking on Check Syntax in the Process pane as shown below.

The syntax checking tool grinds away and then displays the result in the process pane. If error is found then the error is indicated as the  next to the Check Syntax process. But what is the error and where is it?



3.12 Fixing Errors:

Scroll manually the transcript pane at the bottom of the window, and then find the first error and then go through the explanation to fix the error then click on error tab to find the location of the error.



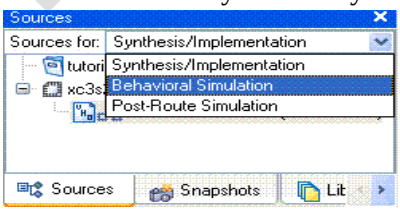
After correcting first error and saving the file, double-click the on Check Syntax in the Process pane to re-check the HDL code. Repeat the same step till process runs without errors and displays a  .

3. Design Simulation

Verifying Functionality using Behavioral Simulation Create a test bench waveform containing input stimulus you can use to verify the functionality of the design.

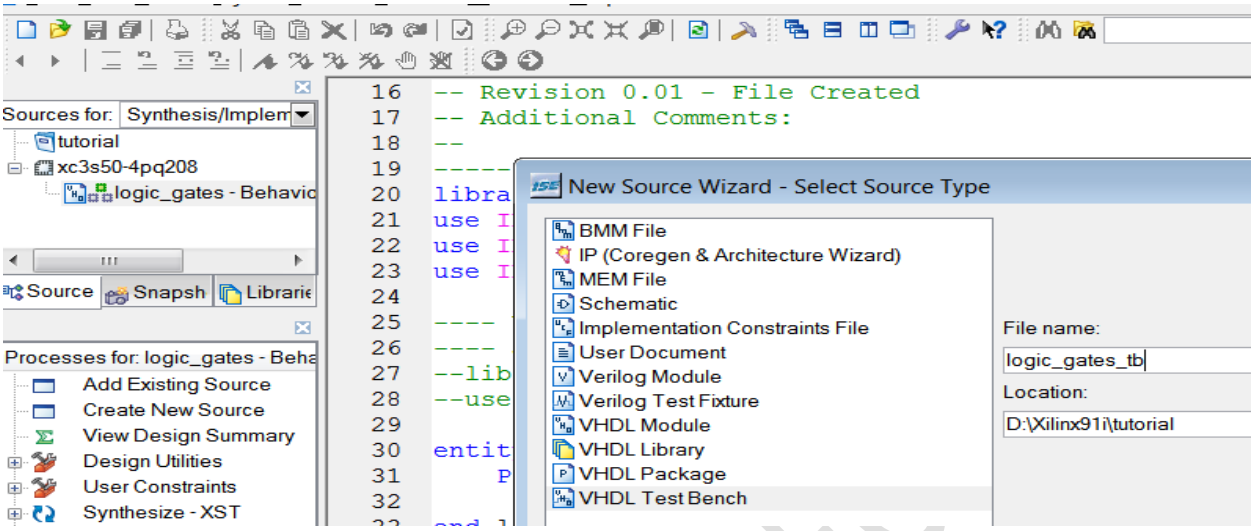
4.1 Create the test bench waveform as follows:

1. Select your HDL file in the Sources window
2. In the Sources window, select the Behavioral Simulation view to see that the test bench waveform file is automatically added to your project.

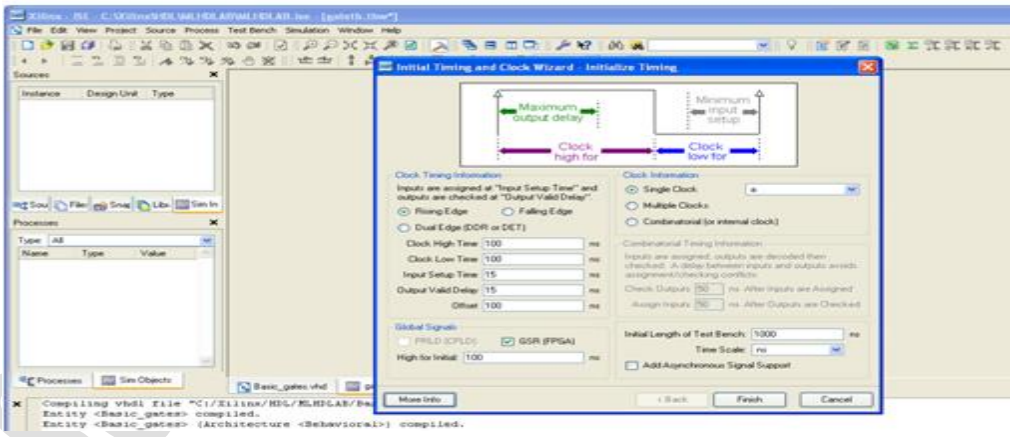


3. Create a new test bench source by selecting Project ➔ New Source.

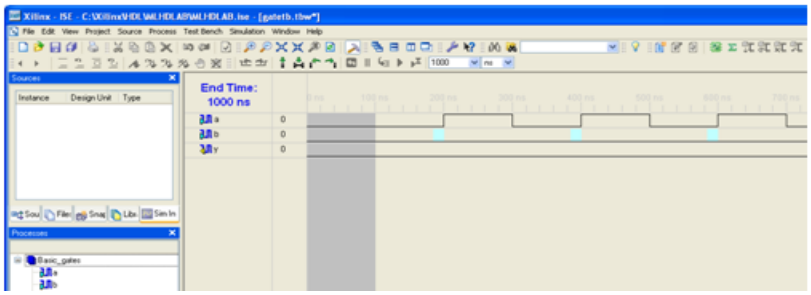
4. In the New Source Wizard, select Test Bench WaveForm as the source type, and type <your test bench file name> in the File Name field.



5. Click Next.
6. The Associated Source page shows that you are associating the test bench waveform with your source file. Click Next.
7. The Summary page shows that the source will be added to the project, and it displays the source directory, type and name. Click Finish.



8. Click Finish to complete the timing initialization.



The test bench waveform is a graphical view of a test bench.

9. Save the waveform.
10. Close the test bench waveform.

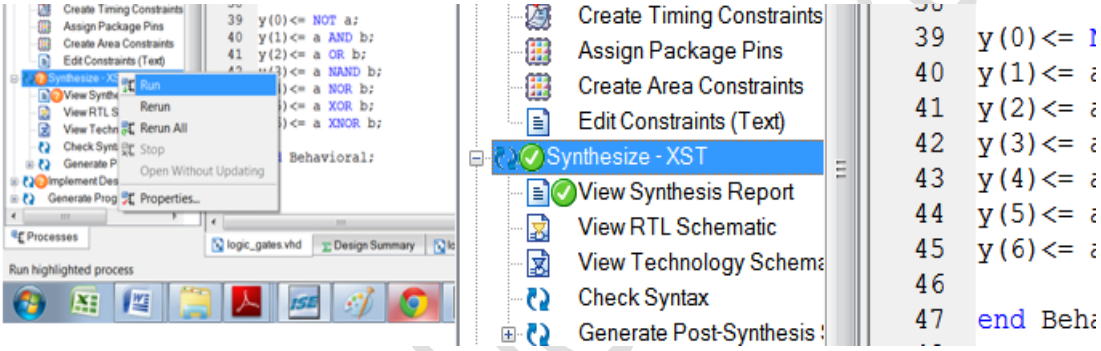
4.2 Simulating Design Functionality

Verify that the design functions as you expect by performing behavior simulation as follows:

1. Verify that Behavioral Simulation and <your test bench waveform file> are selected in the Sources window.
2. In the Processes tab, click the "+" to expand the Xilinx ISE Simulator process and double-click the Simulate Behavioral Model process. The ISE Simulator opens and runs the simulation to the end of the test bench.
3. To view your simulation results, select the Simulation tab and zoom in on the transitions.
4. Verify that the outputs of your design are same as expected (refer your truth table).

5 Syntheses:

Double click on Synthesize – XST tab is located in process window. To view the schematic or netlist of your design, double click on view RTL Schematic tab.

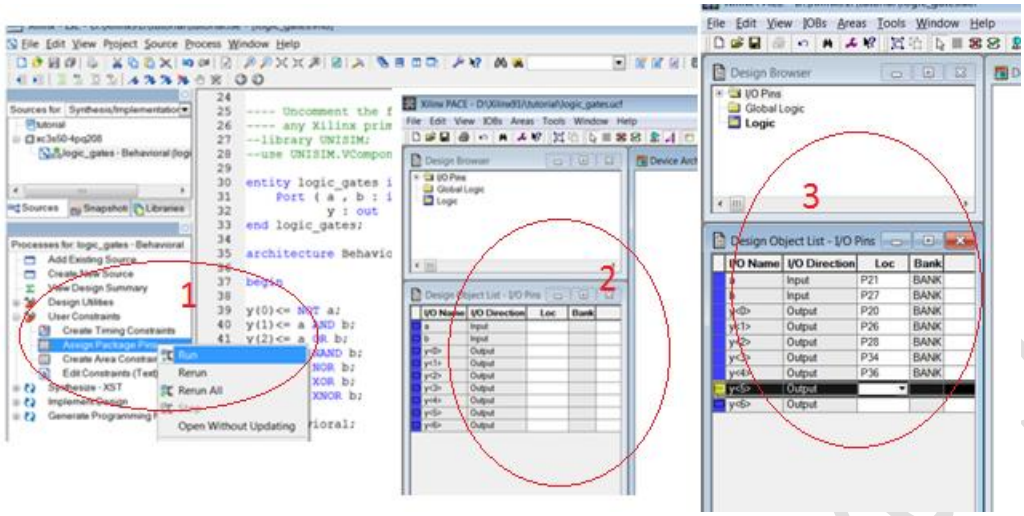


6. Assigning Pin Location Constraints

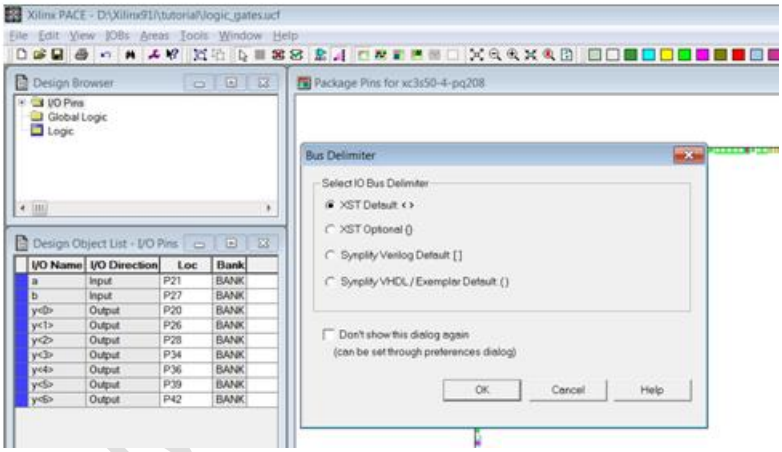
Specify the pin locations for the ports of the design so that they are connected correctly on the Spartan-3 board. To constrain the design ports to package pins, do the following:

1. Verify that <your HDL file> is selected in the Sources window.
2. Double-click the Assign Package Pins process found in the User Constraints process group. The Xilinx Pinout and Area Constraints Editor (PACE) opens.
3. In the Design Object List window, enter a pin location for each pin in the Loc column by referring the pin list table mentioned in manual. Following information explains the example for assigning pins:

- a input port connects to FPGA pin 21 (TS1)
- b input port connects to FPGA pin 27 (TS2)
- y<0> output port connects to FPGA pin 20 (OPLED1 on board)
- y<1> output port connects to FPGA pin 26 (OPLED2 on board)
- y<2> output port connects to FPGA pin 28 (OPLED3 on board)
- y<3> output port connects to FPGA pin 34 (OPLED4 on board)
- y<4> output port connects to FPGA pin 36 (OPLED5 on board)
- y<5> output port connects to FPGA pin 39 (OPLED6 on board)
- y<6> output port connects to FPGA pin 42 (OPLED7 on board)



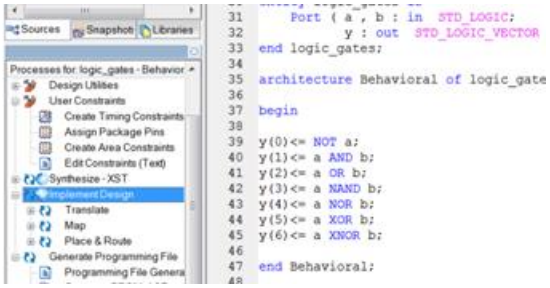
4. Select File ➔ Save. You are prompted to select the bus delimiter type based on the synthesis tool you are using. Select XST Default <> and click OK.



5. Close PACE.

7 Implement Design and Verify Constraints:

7.1 Implement Design Click on implement design

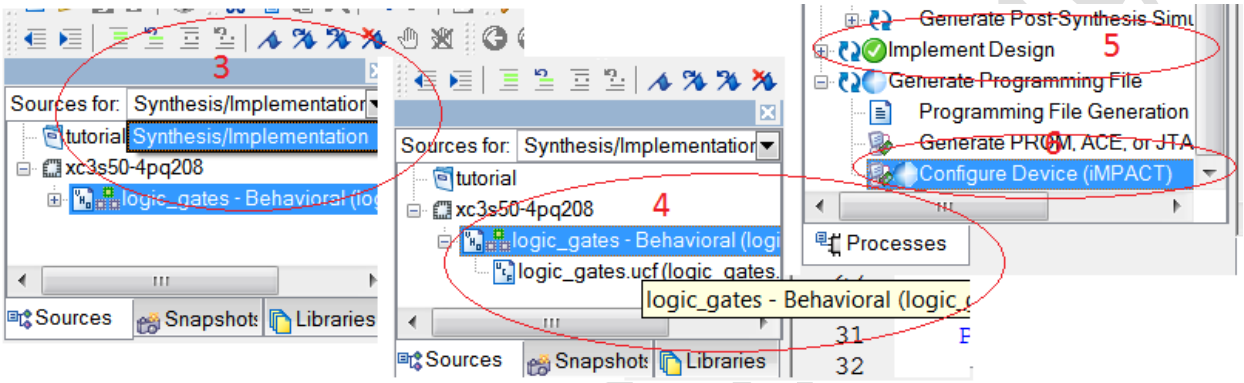


7.2 Verify Constraints: If any constraints errors then you have to verify that the assign package pin are correct. Repeat the step right implementation design.

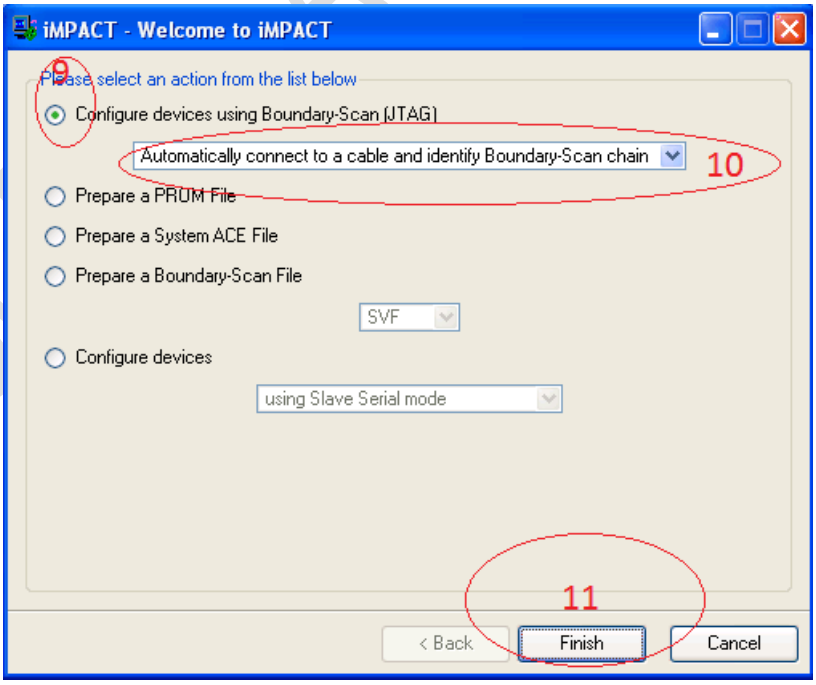
8. Download Design to the Spartan3 Board

This is the last step in the design verification process. This section provides simple instructions for downloading your design to the Spartan-3 board.

1. Connect the 5V DC power cable to the power input on the board.
2. Connect the download cable between the PC and demo board.
3. Select Synthesis/Implementation from the drop-down list in the Sources window.
4. Select <your HDL file> in the Sources window.
5. In the Processes window, click the "+" sign to expand the Generate Programming File processes.
6. Double-click the Configure Device (iMPACT) process.

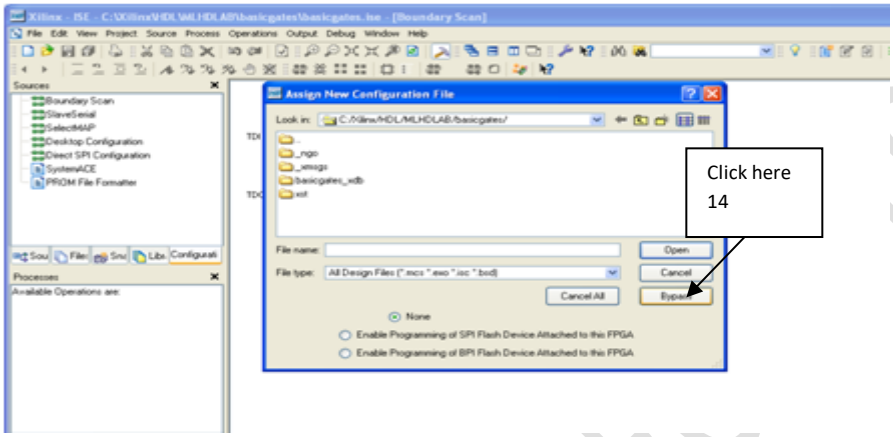


7. The Xilinx WebTalk Dialog box may open during this process. Click Decline.
8. Select Disable the collection of device usage statistics for this project only and click OK. iMPACT opens and the Configure Devices dialog box is displayed.
9. In the Welcome dialog box, select Configure devices using Boundary-Scan (JTAG).
10. Verify that automatically connect to a cable and identify Boundary-Scan chain is selected.

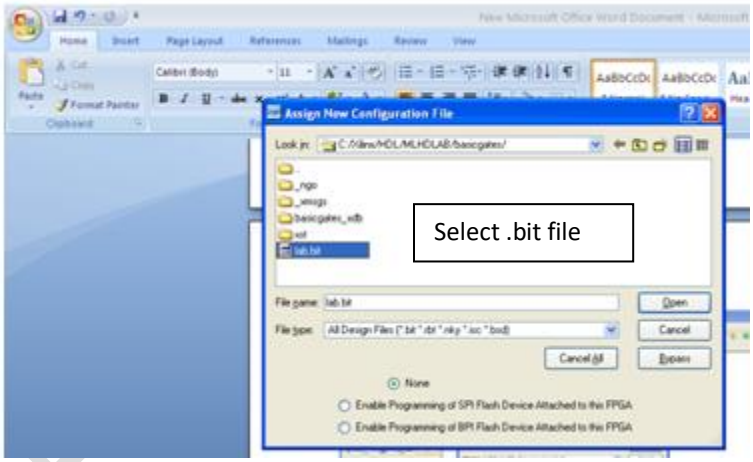


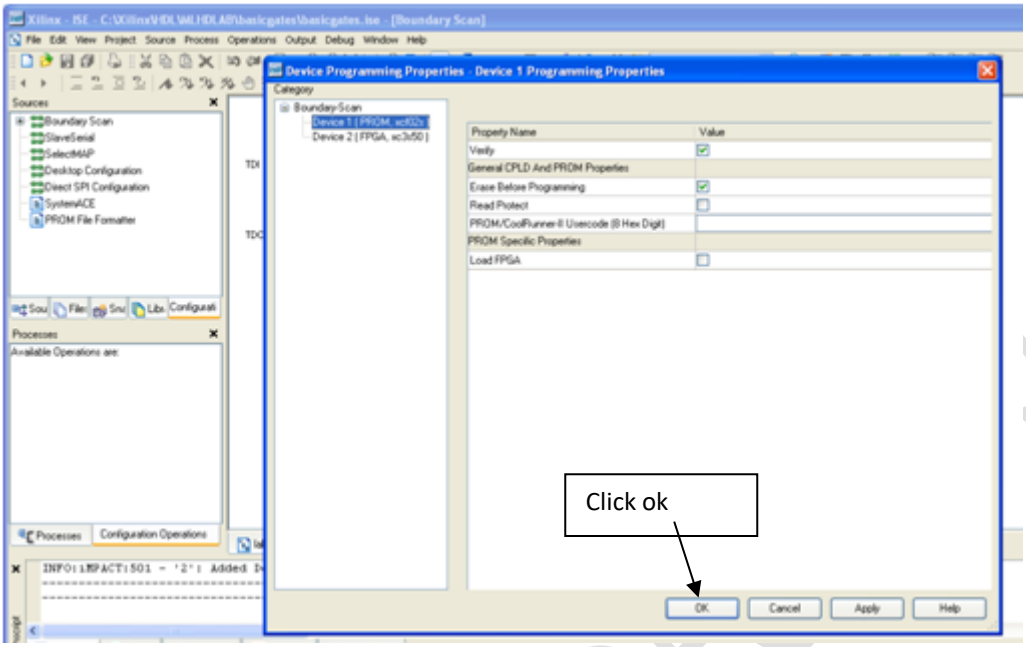
11. Click Finish.

12. If you get a message saying that there are two devices found, click OK to continue. The devices connected to the JTAG chain on the board will be detected and displayed in the iMPACT window.
13. If you get a Warning message, click OK.
14. Select Bypass to skip any remaining devices.

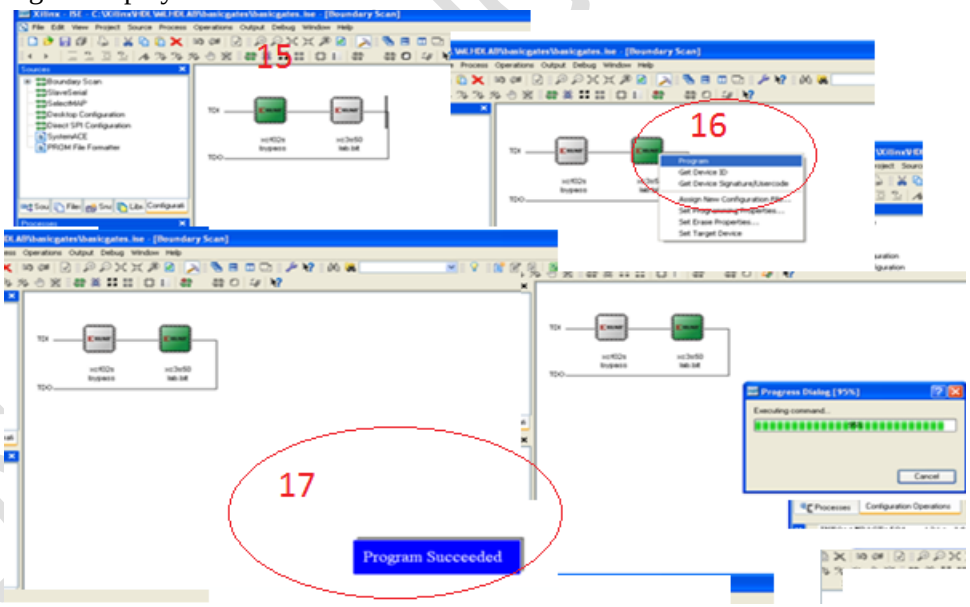


To assign a configuration file to the xc3s200 device in the JTAG chain, select the <your design>.bit file and click Open.





15. The Assign New Configuration File dialog box appears.
16. Right-click on the xc3s50 device image, and select Program... The Programming Properties dialog box opens.
17. Click OK to program the device. When programming is complete, the Program Succeeded message is displayed.



On board output led are on and off as inputs changing. Then test your design comparing with your truth table.

EXPERIMENTS (DIGITAL LOGIC)

1. Realization of Half/Full adder and Half/Full Subtractors using logic gates

i) Half Adder and Full Adder

ii) Half Subtractor and Full Subtractor by using Basic gates and NAND gates

COMPONENTS REQUIRED:

IC 7400, IC 7408, IC 7486, IC 7432, Patch Cords & IC Trainer Kit.

THEORY:

Combinational Circuit is one in which value of the output at any instant of time depend only on present state of the input ie it does not have memory. Adder and Subtractor is a combinational circuit. Adders and Subtractor find application in Arithmetic and Logic Unit, Array Multipliers, tri State buffers etc

Half adder is one which adds two single bit signals at a time. It has two inputs (A&B) and two Output (Sum & Carry) .Full adder is one which adds three one bit signals at a time. It has three inputs (A, B &C) and two Outputs (Sum & Carry). The basic rules of binary addition are

0+0 = 0, 0+1 = 1, 1+0 = 1, 1+1 = Sum 0 with carry 1

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

SUM: $S = A \oplus B \oplus Cin$

CARRY: $Co = AB + Cin\,(A \oplus B)$

Half Subtractor is one which subtracts two single bit signals at a time. It has two inputs (A&B) and two Output (Difference & Borrow) .Full Subtractor is one which subtracts three one bit signals at a time. It has three inputs (A, B&C) and two Outputs (Difference & Borrow). The basic rules of binary Subtraction are

1-1= 0, 0-1 =Difference 1 with Borrow 1, 1-0 = 1, 1-1 = 0.

Full Subtractor: Subtracting two single-bit binary values, B, Bin from a single-bit value A Produces a difference bit D and a borrow out Br bit. This is called full subtraction. The Boolean functions describing the full-Subtractor are:

$$D = A \oplus B \oplus Bin$$

$$Bo = \overline{A}B + Bin(\overline{A \oplus B})$$

PROCEEDURE:

1. Place the IC in the socket of the trainer kit.

2. Make the connections for the gate as shown in the circuit diagram.

3. Verify the Truth Table.

4. Repeat the above steps for other gates in the different IC chips

TO REALIZE

1. HALF ADDER

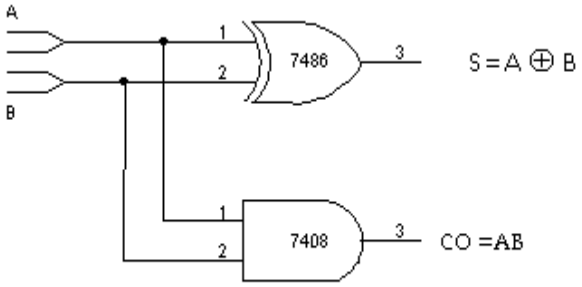
TRUTH TABLE

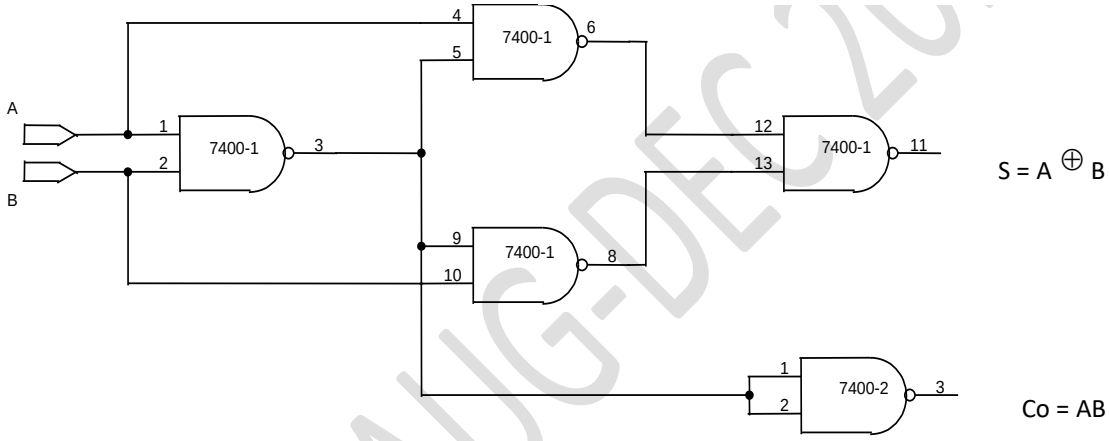
I/P		O/P	
A	B	S	Co
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

BOOLEAN EXPRESSION

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

$Co = AB$





1. FULL ADDER

TRUTH TABLE

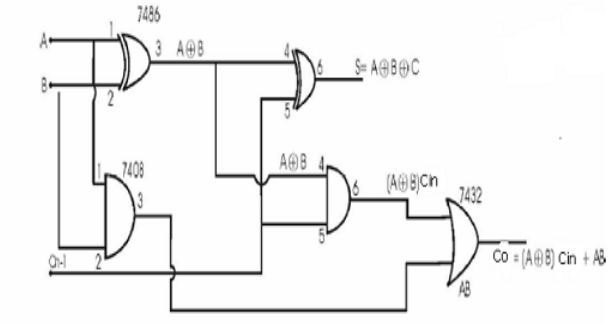
	INPUTS		OUTPUTS	
A	B	Cin	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

BOOLEAN EXPRESSIONS:

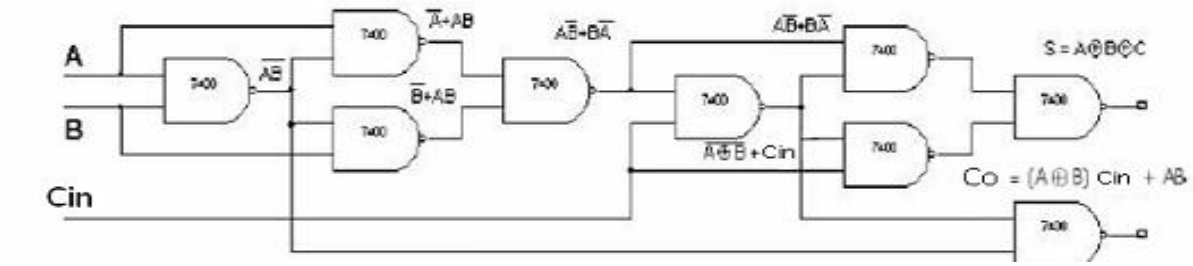
SUM: $S = A \oplus B \oplus Cin$

CARRY: $C_o = AB + Cin\,(A \oplus B)$

Full Adder using basic gates:-



Full Adder using NAND gates only:-



HALF SUBTRACTOR
TRUTH TABLE

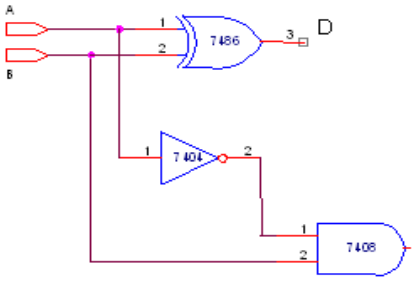
I/P		O/P	
A	B	D	Bo
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

BOOLEAN EXPRESSION

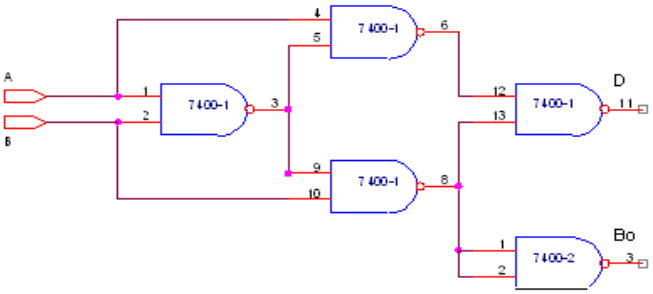
Difference: $D = \overline{A}B + A\overline{B} = A \oplus B$

Borrow: $Bo = \overline{A}B$

i) BASIC GATES:



NAND gates :



FULL SUBTRACTOR:
TRUTH TABLE

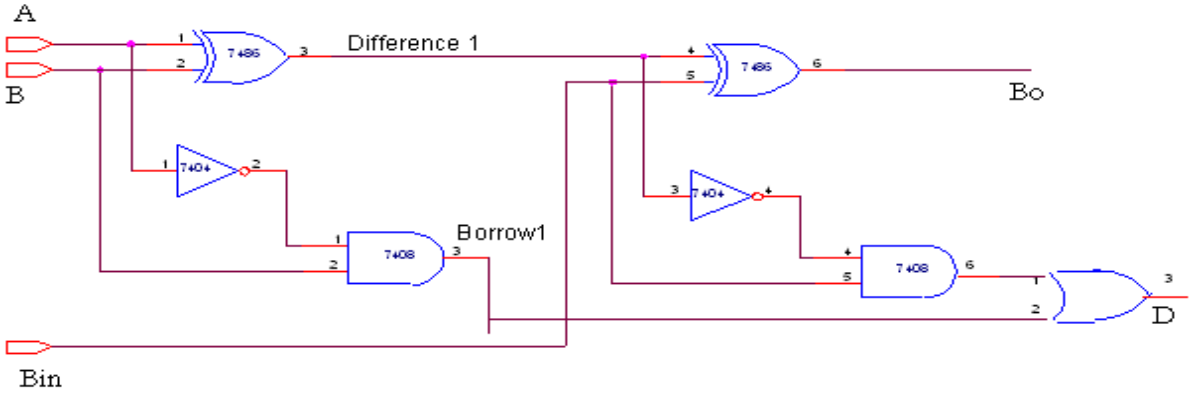
INPUTS			OUTPUTS	
A	B	Bin	D	Bo
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

BOOLEAN EXPRESSION

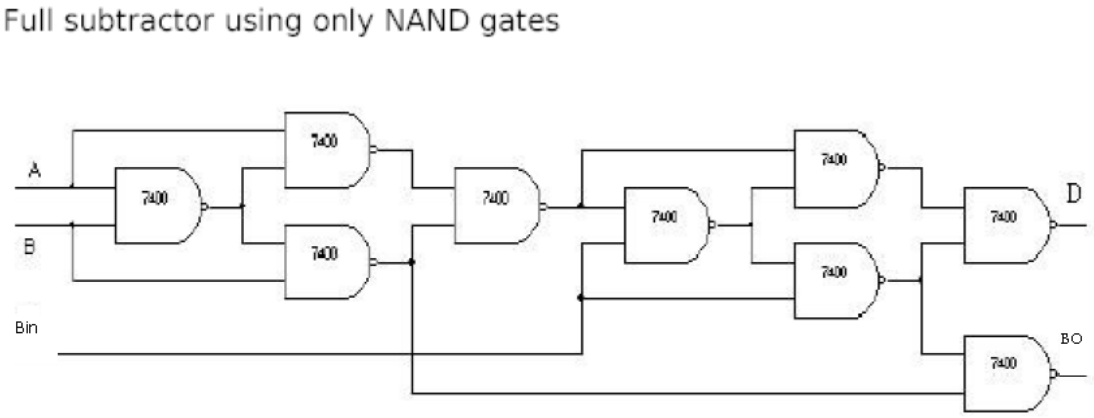
$D = A \oplus B \oplus Bin$

$Bo = \overline{A}B + Bin(\overline{A \oplus B})$

USING BASIC GATES



USING NAND gates



Result: Half adder, Full adder, Half Subtractor, Full Subtractor circuits are realized using logic gates and the truth tables are verified.

Probable Viva Questions:

1) What is a half adder?

2) What is a full adder?

3) What are the applications of adders?

4) What is a half Subtractor?

5) What is a full Subtractor?

6) What are the applications of Subtractor?

7) Obtain the minimal expression for above circuits.

8) Realize a full adder using two half adders

9) Realize a full Subtractor using two half Subtractor

Applications:

Arithmetic Logic Units & Processing units

2. Code conversion

(i) BCD to Excess-3 code conversion and vice versa

(ii) Realization of Binary to Gray code conversion and vice versa

THEORY: This is another binary code which finds application in input/output devices, shaft position encoders and several analog to digital converters. This is an unweighted code. The gray code exhibits only a single bit change from one code number to the next.

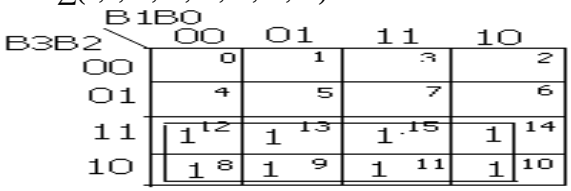
Code converter is a combinational circuit that translates the input code word into a new corresponding word. The excess-3 code digit is obtained by adding three to the corresponding BCD digit. To Construct a BCD-to-excess-3-code converter with a 4-bit adder feed BCD code to the 4-bit adder as the first operand and then feed constant 3 as the second operand .The output is the corresponding excess-3 code. To make it work as a excess-3 to BCD converter, we feed excess-3 code as the first operand and then feed 2's complement of 3 as the second operand. The output is the BCD cod

BINARY TO GRAY CODE CONVERSION VICE &VERSA

Design and set up 4-bit Binary to Gray code converter circuit

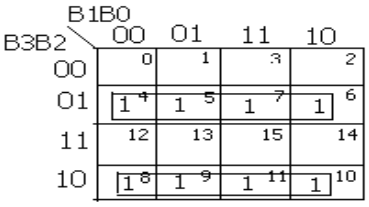
BINARY				GRAY CODE			
B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

G3 = ∑(8,9,10,11,12,13,14,15)



G3=B3

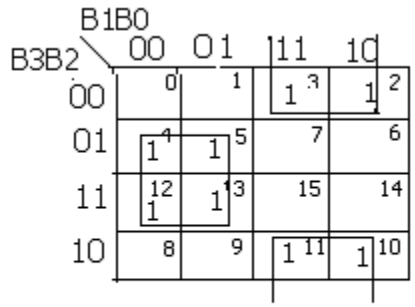
G2 = ∑(4,5,6,7,8,9,10,11)



$\overline{B3}B2 + B3\overline{B2}$

G2= B3⊕B2

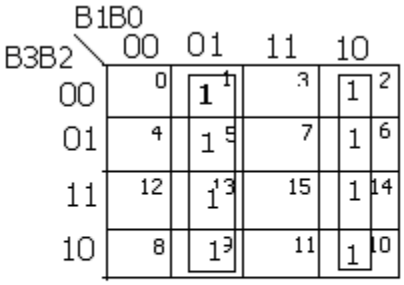
G1= ∑(2,3,4,5,10.,11,12,13)



$G1 = B2\overline{B1} + \overline{B2}B1$

$G1 = B1 \oplus B2$

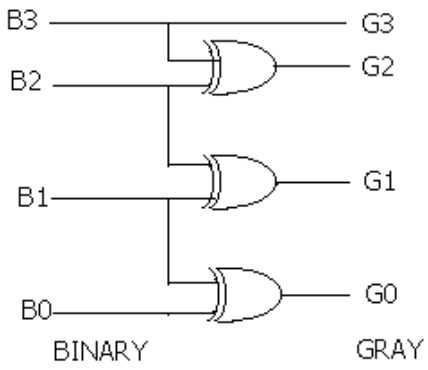
G0 = ∑(1,2,3,5,6,9,10,13,14)



$G0 = \overline{B1}B0 + B1\overline{B0}$

$G0 = B1 \oplus B0$

Binary to Gray Conversion USING EX-OR GATES



2.

$G3 = B3$

$G2 = \overline{B3}B2 + B3\overline{B2}$

$G2 = B3 \oplus B2$

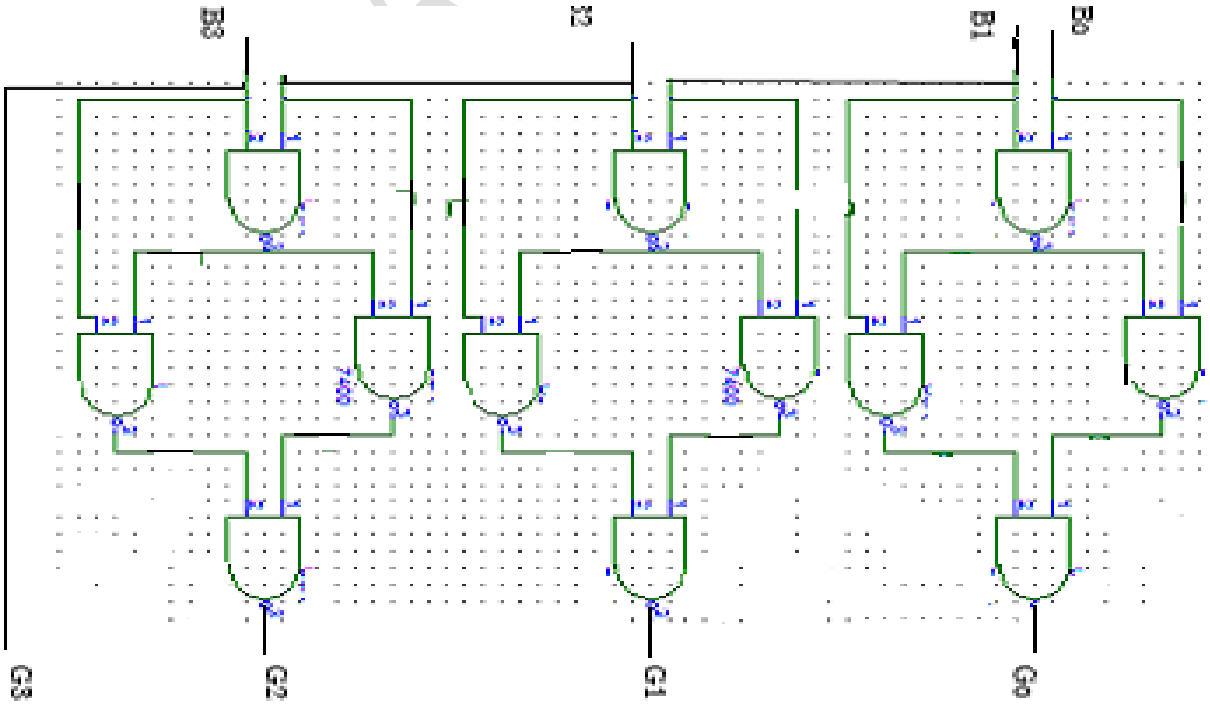
$G1 = B2\overline{B1} + \overline{B2}B1$

$G1 = B1 \oplus B2$

$G0 = \overline{B1}B0 + B1\overline{B0}$

$G0 = B1 \oplus B0$

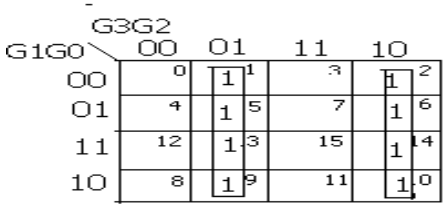
Binary to Gray Conversion USING NAND GATES



Design and set up 4-bit Gray to binary code circuit using gates.

GRAY CODE				BINARY CODE			
G3	G2	G1	G0	B3	B2	B1	B0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

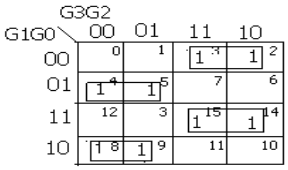
K MAP FOR B2



$B2 = G3G2 + G3\overline{G2}$

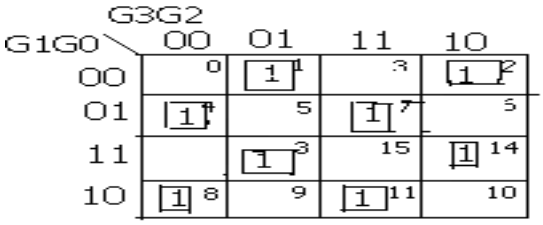
$B2 = G3 \oplus G2$

K MAP FOR B1

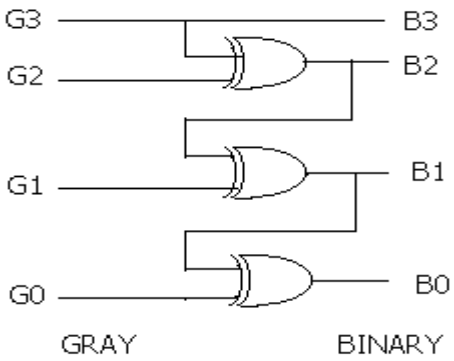


$B1 = \overline{G1}G0\overline{G3} + \overline{G1}\overline{G0}G3 + G1G0G3 + G1\overline{G0}\overline{G3}$
$= \overline{G1}(G0\overline{G3} + \overline{G0}G3) + G1(G0G3 + \overline{G0}\overline{G3})$
$= \overline{G1}(G0 \oplus G3) + G1(\overline{G0 \oplus G3})$
$B1 = G3 \oplus G2 \oplus G1$

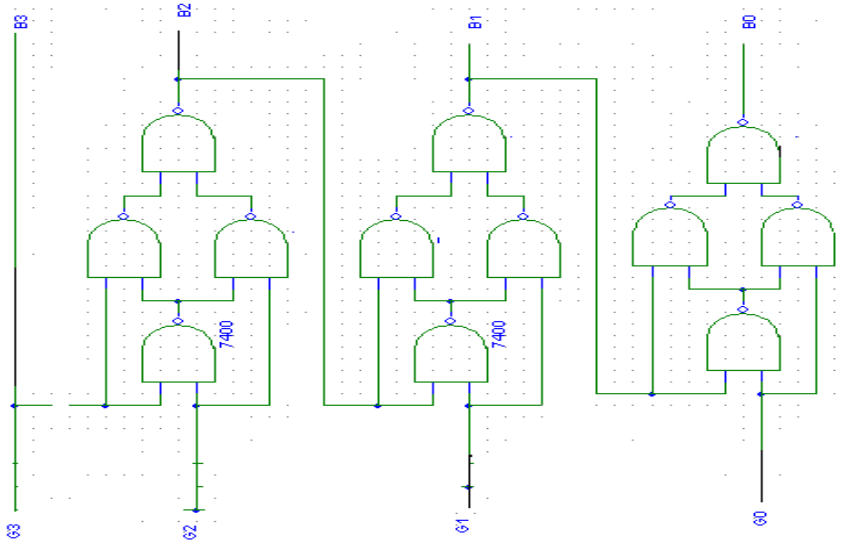
K MAP FOR B0



$B0 = G0 \oplus G1 \oplus G2 \oplus G3$



USING NAND GATES



Result: The digital circuit for binary to gray & gray to binary is built and verified the Conversion Table

Possible viva questions:

1) What are code converters?

2) What is the necessity of code conversions?

3) What is gray code?

4) Realize the Boolean expressions for

a) Binary to gray code conversion

b) Gray to binary code conversion

Applications:

Also referred as Minimum error code, used in error detection.

3. Realization of MUX and DEMUX using basic gates and IC's 74139 and 74153

1) To design and set up a 4:1 Multiplexer (MUX) using only NAND gates.

2) To design and set up a 1:4 De Multiplexer (DE-MUX) using only NAND gates.

3) To verify the various functions of IC 74153(MUX) and IC 74139(DEMUX).

4) To set up a Half/Full Adder and Half/Full Subtractor using IC 74153.

COMPONENTS REQUIRED:

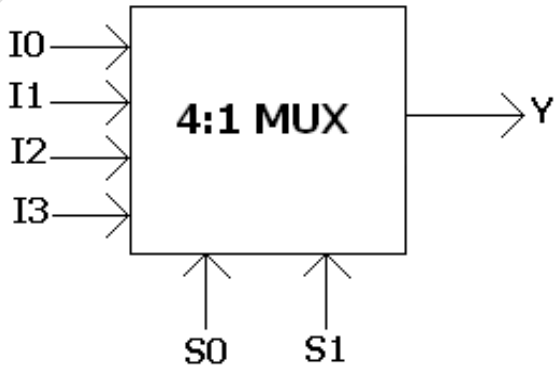
IC 7400, IC 7410, IC 7420, IC 7404, IC 74153, IC 74139, Patch Cords & IC Trainer Kit.

THEORY: Multiplexers are very useful components in digital systems. They transfer a large number of information units over a smaller number of channels, (usually one channel) under the control of selection signals. Multiplexer means many to one. A multiplexer is a circuit with many Inputs but only one output. By using control signals (select lines) we can select any input to the output. Multiplexer is also called as data selector because the output bit depends on the input data bit that is selected. Multiplexer is a digital switch. It allows digital information from several sources to be routed onto a single output line.. There are 2^n input lines and n selection lines whose bit combinations determine which input is selected. Therefore multiplexer is 'many into one' Multiplexer finds application like Parallel to Serial conversion, Data routing, Operation Sequencing and Logic function generation

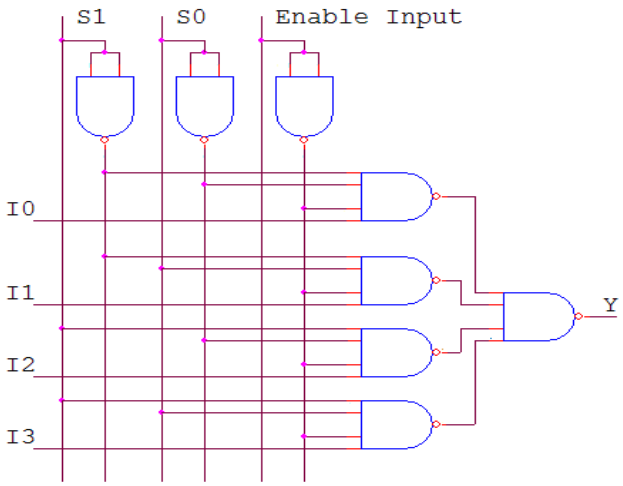
De-multiplexers perform the opposite function of multiplexers. A Demultiplexer is circuits that receives information on a single line and transmits this information on one of are 2^n possible output lines. The selection of specific output line is controlled by the values of n selection lines.

IC 74153: the IC 74153 is a dual 4-i/p mux that can select 2 bits of data from up to eight sources under the control of the common select inputs (S0,S1). The two 4- i/p Mux circuits have individual active low enables (E1,E2) which can be used to strobe the outputs independently outputs (Y1,Y2) are forced low when the corresponding enables (E1,E2) are high.

4:1 MULTIPLEXER

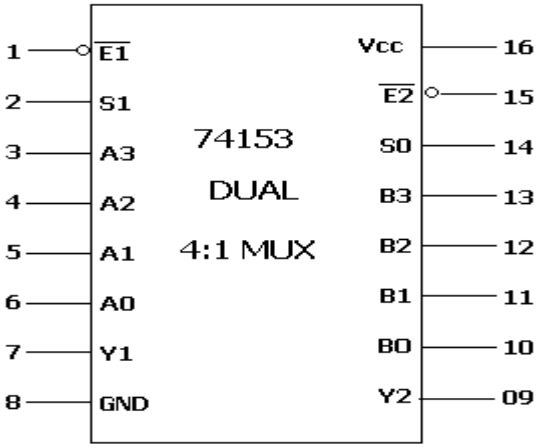


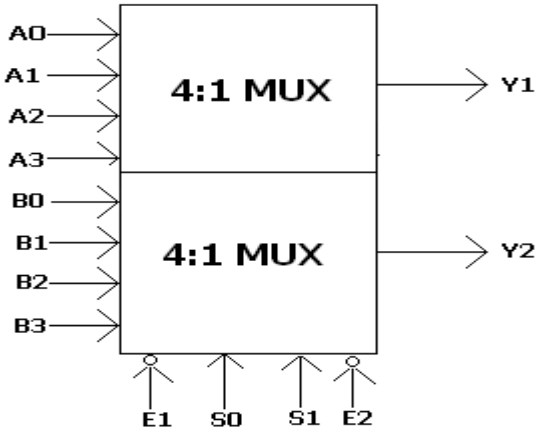
USING NAND GAES



Select inputs		Enable input	Inputs				Out put
S1	S0	E	I0	I1	I2	I3	Y
0	0	0	0	X	X	X	0
		0	1	X	X	X	1
0	1	0	X	0	X	X	0
		0	X	1	X	X	1
1	0	0	X	X	0	X	0
		0	X	X	1	X	1
1	1	0	X	X	X	0	0
		0	X	X	X	1	1

PIN DETAILS OF 74153

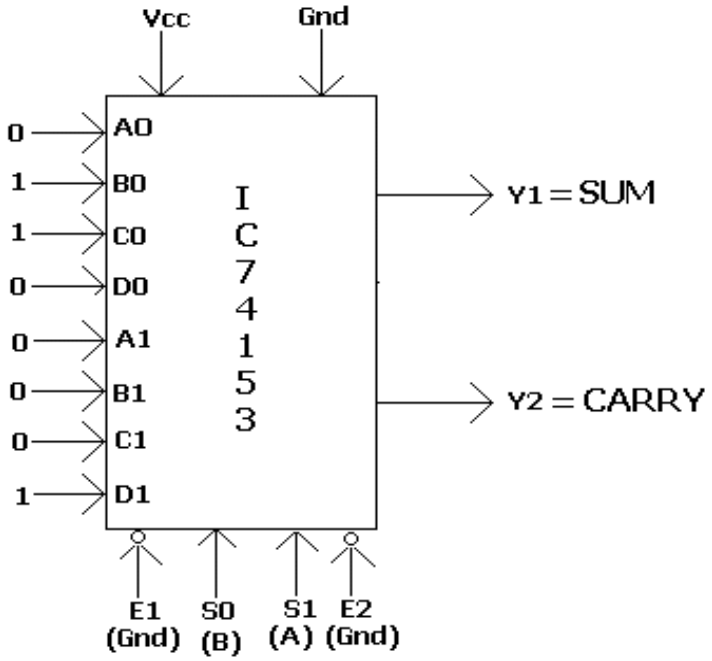




Realization of Half Adder Using IC

TRUTH TABLE **CIRCUIT DIAGRAM**

A	B	SUM	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Realization of Full Adder Using IC

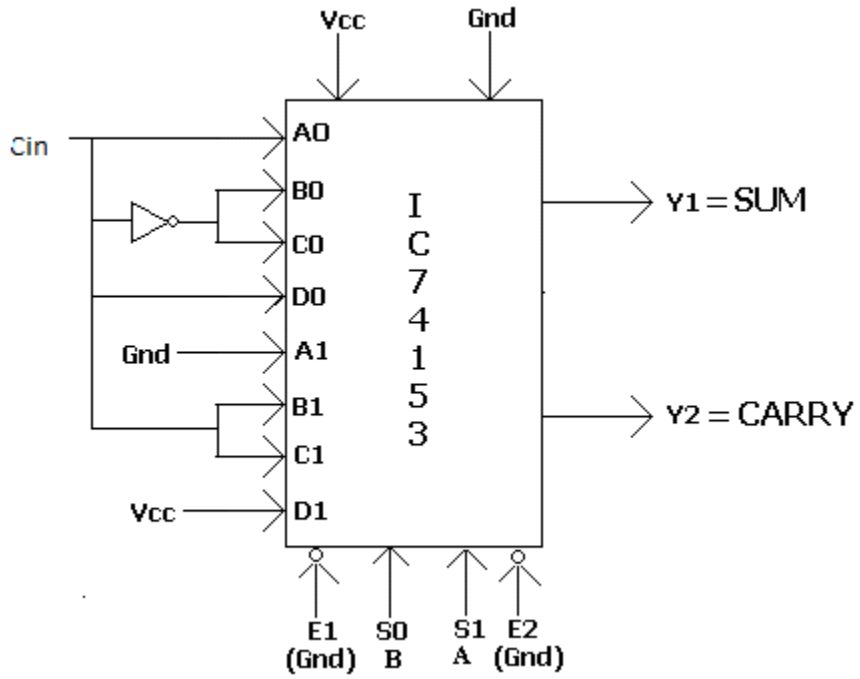
Here the outputs sum and carry out are represented in

Terms of input Cin

A	B	SUM	Cout
0	0	Cin	0
0	1	$\overline{Cin}$	Cin
1	0	$\overline{Cin}$	Cin
1	1	Cin	1

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

FULL ADDER

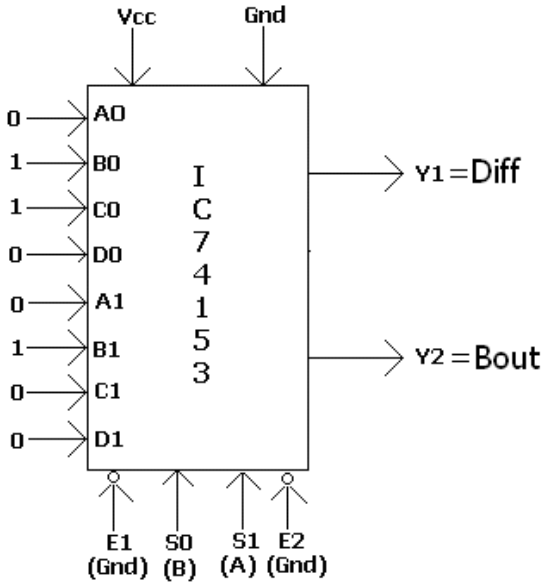


Realization Of Half Subtractor Using IC

TRUTH TABLE

A	B	Diff	Bout
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

CIRCUIT DIAGRAM



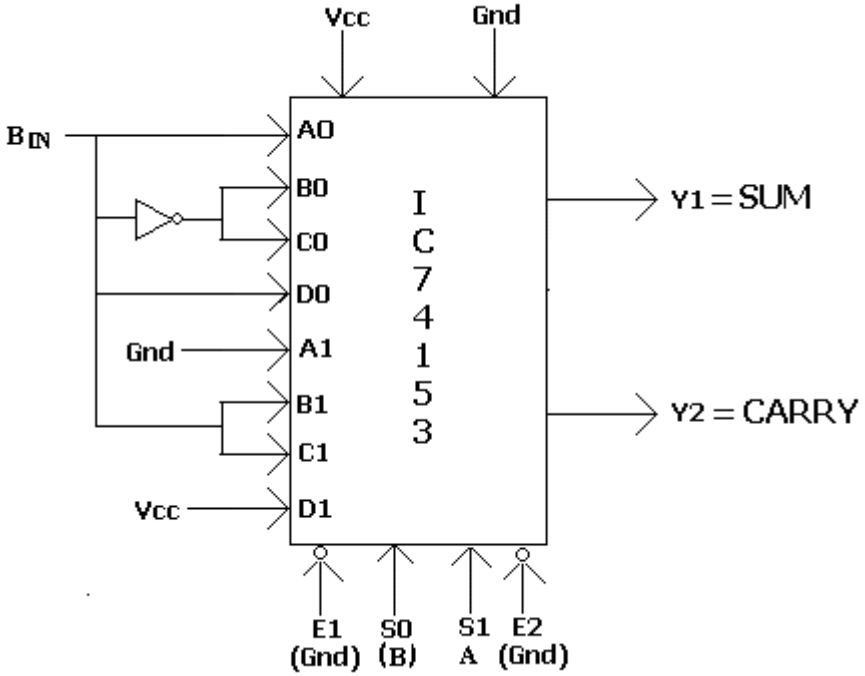
Realization of Full Subtractor Using IC

Truth Table

A	B	Bin	Diff	Bout
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

A	B	Diff	Bout
0	0	Bin	Bin
0	1	$\overline{Bin}$	1
1	0	$\overline{Bin}$	0
1	1	Bin	Bin

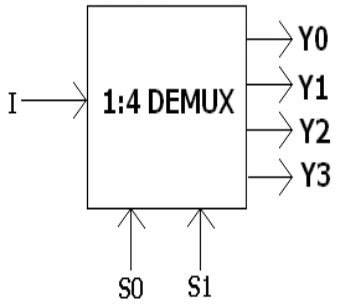
CIRCUIT DIAGRAM



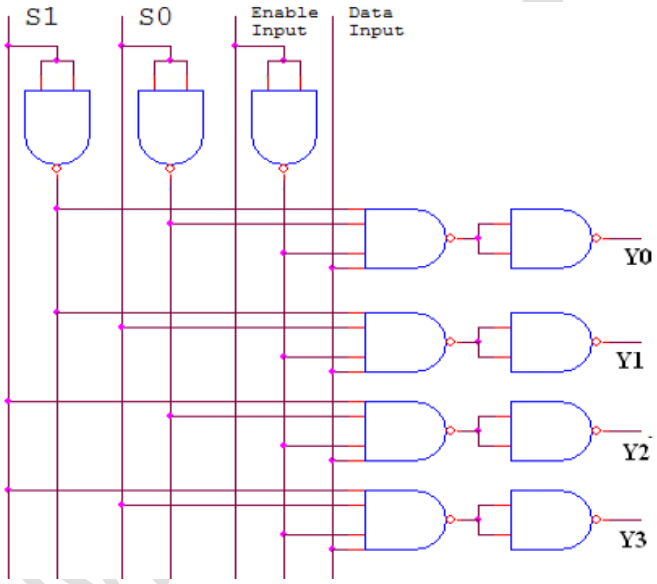
DEMULTIPLEXER

FUNCTION TABLE

SYMBOL



Enable Input	Data Input	Select Inputs		Outputs			
		S1	S0	Y3	Y2	Y1	Y0
0	I	0	0	0	0	0	1
0	I	0	I	0	0	1	0
0	I	1	0	0	1	0	0
0	I	1	1	1	0	0	0

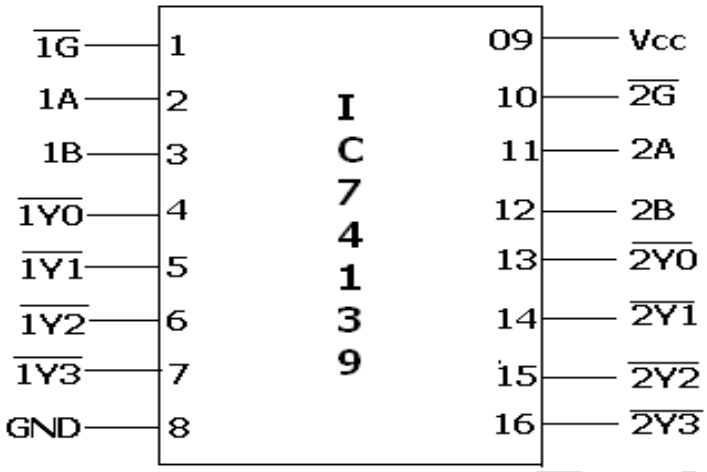


CIRCUIT DIAGRAM USING NAND GATES

ARITHMETIC CIRCUITS USING DEMULTIPLEXER IC74139

IC 74139: The IC 74139 is a high speed dual 1 of 4 decoder/ de multiplexer. This device has two independent decoders each accepting two binary weighted inputs (a, b) and providing four mutually exclusive active low outputs (Y0-Y3).Each decoder has an active low enable (E) when E=1 every o/p is forced high. The enable can be used as the data input for a 1 of 4 DEMUX applications

IC 74139 DEMUX/ DECODER



FUNCTIONAL TABLE

$\overline{1G}$	A(S1)	B(S0)	Y3	Y2	Y1	Y0
1	X	X	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

IC 74139: The IC 74139 is a high speed dual 1 of 4 decoder/ de multiplexer. This device has two independent decoders each accepting two binary weighted inputs (a, b) and providing four mutually exclusive active low outputs (Y0-Y3).Each decoder has an active low enable (E) when E=1 every o/p is forced high. The enable can be used as the data input for a 1 of 4 DEMUX applications

1. HALF ADDER/HALF SUBTRACTOR

A	B	Sum	Carry	Diff	Bout
0	0	0	0	0	0
0	1	1	0	1	1
1	0	1	0	1	0
1	1	0	1	0	0

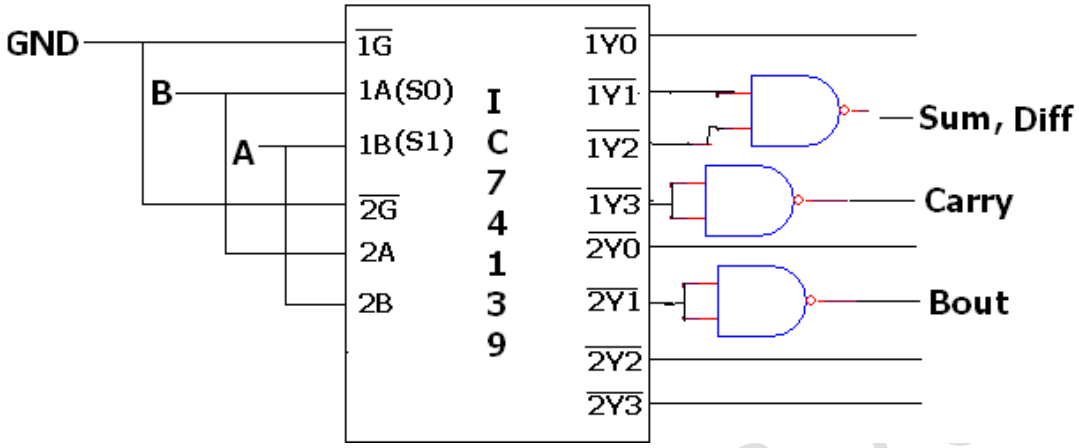
From truth table sum = $\sum 1,2$

Carry = $\sum 3$

Diff = $\sum 1,2$

Bout = $\sum 1$

LOGIC DIAGRAM



FULL ADDER

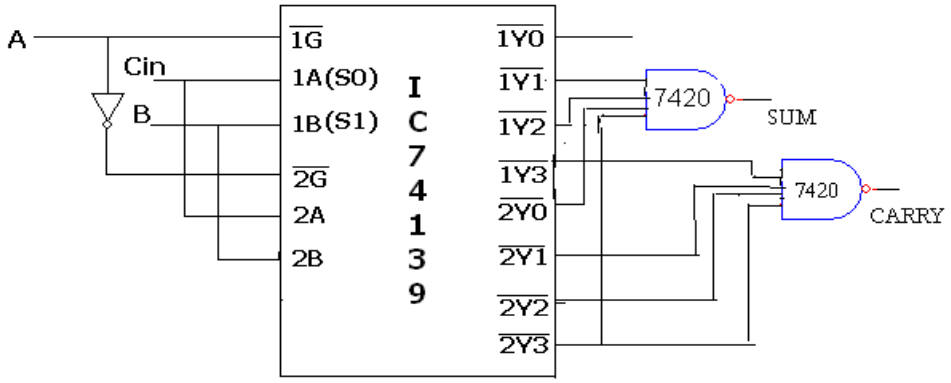
Sum = ∑1,2,4,7

Carry =∑3,5,6,7

TRUTH ABLE

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

FULL ADDER CIRCUIT



FULL SUBTRACTOR

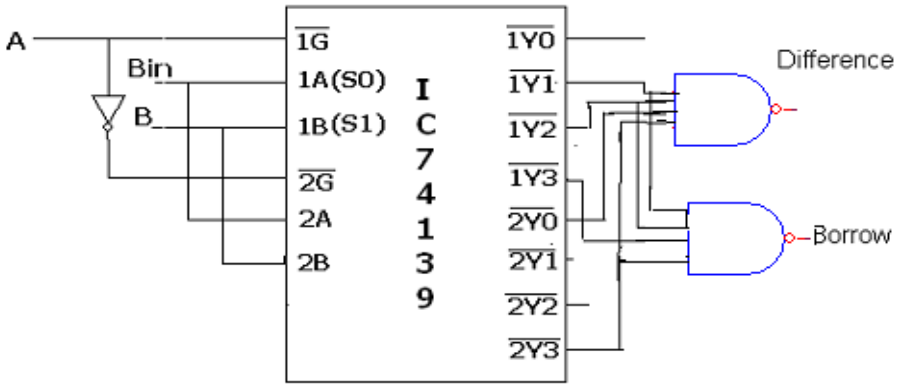
Diff = ∑1,2,4,7

Bout = ∑1,2,3,7

TRUTH ABLE

A	B	Bin	Diff	Bout
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

Full Subtractor LOGIC DIAGRAM



RESULT: Verified MUX/DEMUX adder and subtractor truth table

Possible Viva questions

1) What is a multiplexer?

2) What is a de-multiplexer?

3) What are the applications of multiplexer and de-multiplexer?

4) Derive the Boolean expression for multiplexer and de-multiplexer.

5) How do you realize a given function using multiplexer

6) What is the difference between multiplexer & demultiplexer?

7) In 2^n to 1 multiplexer how many selection lines are there?

8) How to get higher order multiplexers?

9) Implement an 8:1 mux using 4:1 muxes?

Applications:

Parallel to Serial conversion, Data routing, Operation Sequencing and Logic function generation

4. Realization of One/Two bit comparator and study of 7485 magnitude comparator.

(A) To realize 1-bit digital comparator & 2-bit digital comparator

(B) Study of 7485 Magnitude Comparator

COMPONENTS REQUIRED: IC 7400, IC 7410, IC 7420, IC 7432, IC 7486, IC 7402, IC 7408, IC 7404, IC 7485, Patch

Cords & IC Trainer Kit.

THEORY

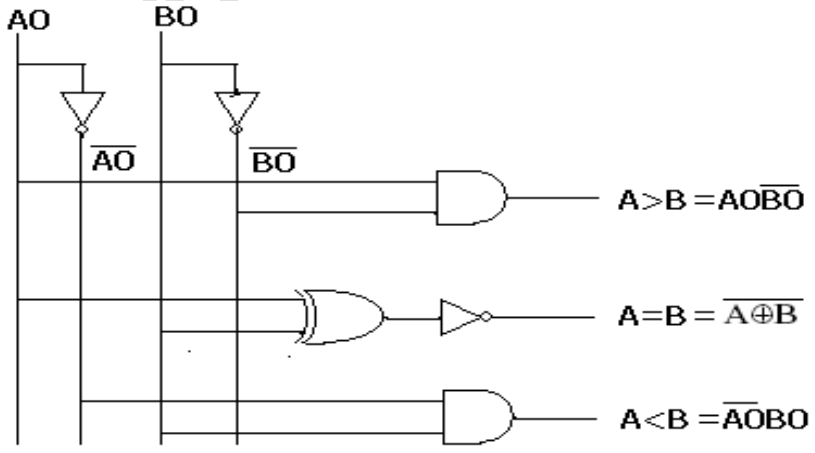
Magnitude Comparator is a logical circuit, which compares two signals A and B and generates three logical outputs, whether A > B, A = B, or A < B. IC 7485 is a high speed 4-bit Magnitude comparator, which compares two 4-bit words. The A = B Input must be held high for proper compare operation. The Magnitude Comparator is a combinational circuit that compares two numbers A & B and determines their relative magnitudes. The outcome of the comparison is specified by three binary variables that indicates whether A>B, A=B, A<B.

1-BIT COMARATOR

TRUTH TABLE

A0	B0	A>B	A=B	A<B
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

BASIC GATES



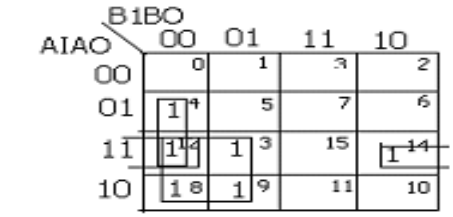
2-BIT COMARATOR

TRUTH TABLE

2-BIT COMARATOR

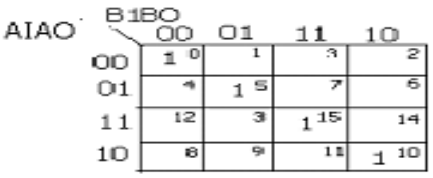
A1	A0	B1	B0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

K –MAP FOR A>B



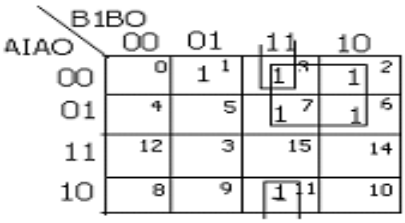
$A>B=A1\overline{B1}+A0\overline{B0}(\overline{B1}+A1)$

K –MAP FOR A=B.



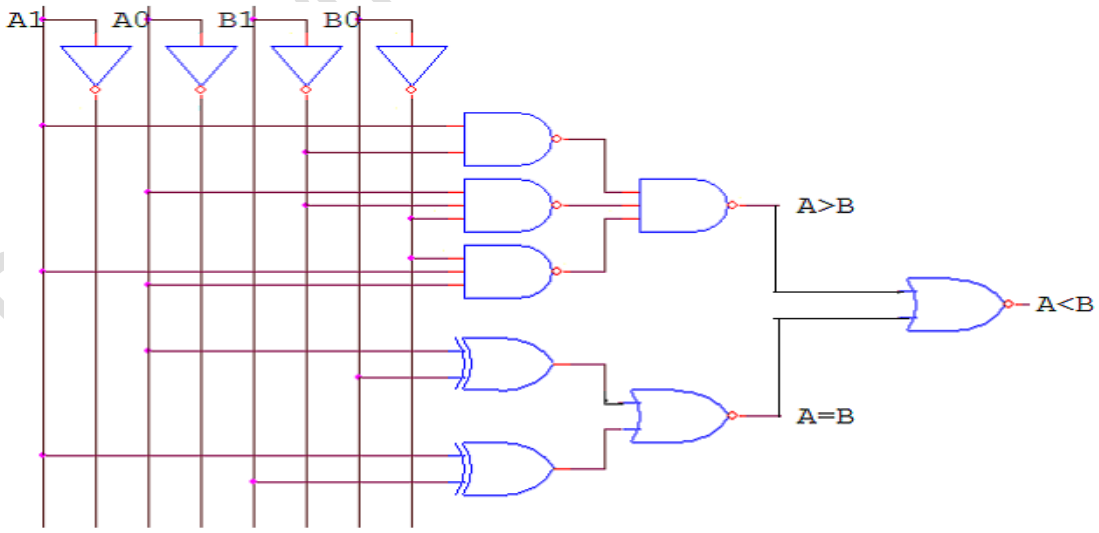
$(A=B)=(\overline{A0 \oplus B0})(\overline{A1 \oplus B1})$

K –MAP FOR A<B



$A<B=A1B1+\overline{A0}B0(B1+\overline{A1})$

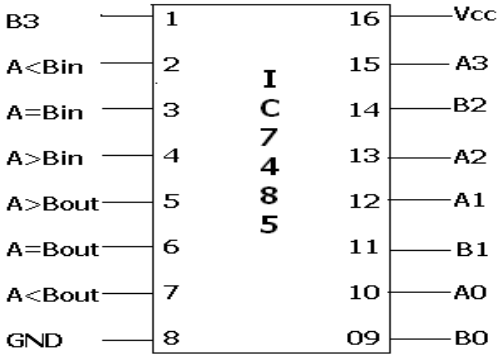
CIRCUIT DIAGRAM



4 Bit Magnitude Comparator

(B) STUDY OF 7485 MAGNITUDE COMPARATOR

IC 7485 PIN DETAILS



Examples

A3	A2	A1	A0	B3	B2	B1	B0	A>B	A=B	A<B
0	0	1	0	1	1	0	1	0	0	1
1	1	1	1	1	1	1	0	1	0	0
1	0	0	1	1	0	0	1	0	1	0
0	1	0	1	0	0	0	0	1	0	0

RESULT: Constructed the 1-bit / 2-bit Comparator and Magnitude comparator and verified the truth Table.

Possible viva questions

1) What is a comparator?

2) What are the applications of comparator?

3) Derive the Boolean expressions of one bit comparator and two bit comparators.

4) How do you realize a higher magnitude comparator using lower bit comparator?

5) Design a 2 bit comparator using a single Logic gates?

6) Design an 8 bit comparator using a two numbers of IC 7485?

Applications:

Arithmetic Logic Units & Processing units.

1-BIT / 2-BIT COMPARATOR AND 7485 MAGNITUDE COMPARATOR

AIM :(A) To realize 1-bit digital comparator & 2-bit digital comparator

(B) Study of 7485 Magnitude Comparator

COMPONENTS REQUIRED: IC 7400, IC 7410, IC 7420, IC 7432, IC 7486, IC 7402, IC 7408, IC 7404, IC 7485, Patch

Cords & IC Trainer Kit.

THEORY

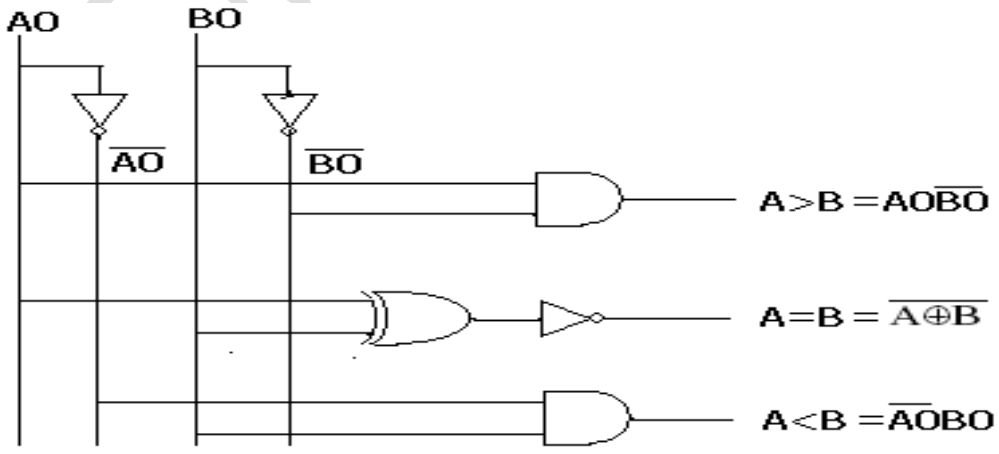
Magnitude Comparator is a logical circuit, which compares two signals A and B and generates three logical outputs, whether A > B, A = B, or A < B. IC 7485 is a high speed 4-bit Magnitude comparator, which compares two 4-bit words. The A = B Input must be held high for proper compare operation. The Magnitude Comparator is a combinational circuit that compares two numbers A & B and determines their relative magnitudes. The outcome of the comparison is specified by three binary variables that indicates whether A>B, A=B, A<B.

1-BIT COMARATOR

TRUTH TABLE

A0	B0	A>B	A=B	A<B
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

BASIC GATES



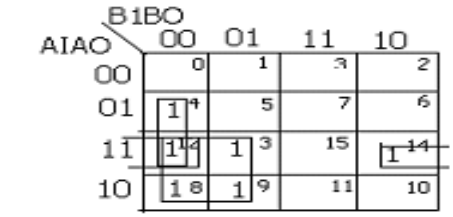
2-BIT COMARATOR

TRUTH TABLE

2-BIT COMARATOR

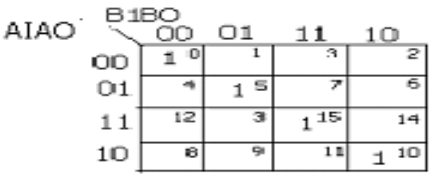
A1	A0	B1	B0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

K –MAP FOR A>B



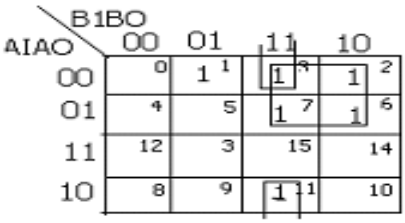
$A>B=A1\overline{B1}+A0\overline{B0}(\overline{B1}+A1)$

K –MAP FOR A=B.



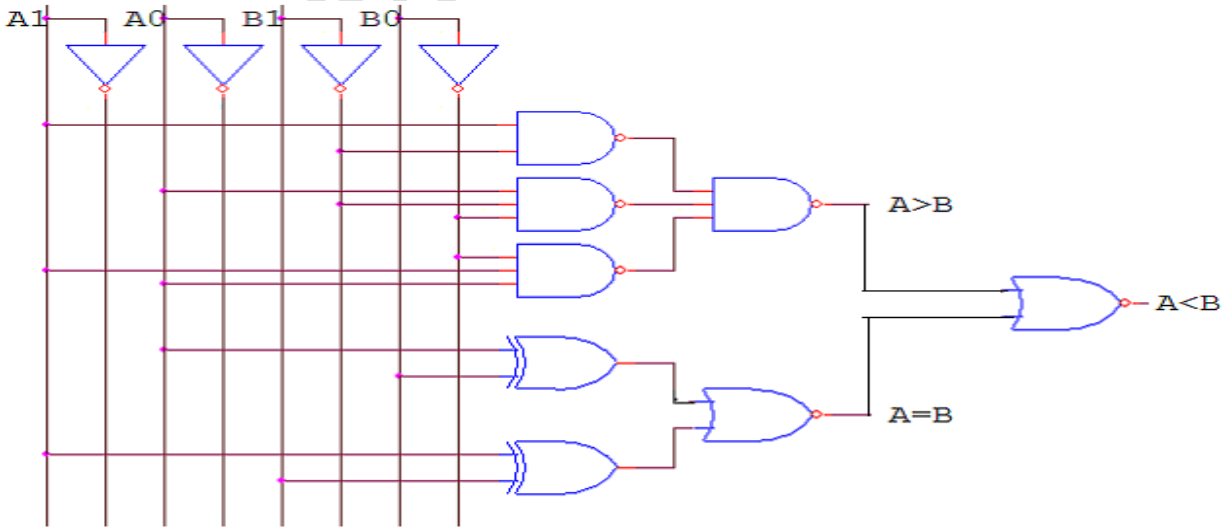
$(A=B)=(\overline{A0 \oplus B0})(\overline{A1 \oplus B1})$

K –MAP FOR A<B



$A<B=A1B1+\overline{A0}B0(B1+\overline{A1})$

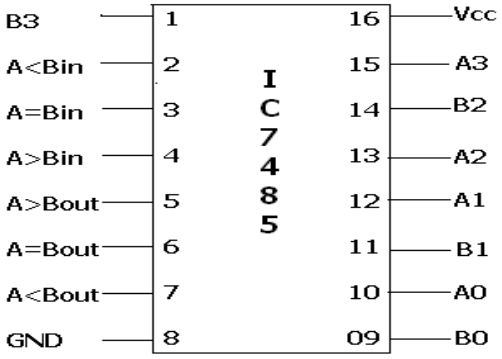
CIRCUIT DIAGRAM



4 Bit Magnitude Comparator

(B) STUDY OF 7485 MAGNITUDE COMPARATOR

IC 7485 PIN DETAILS



Examples

A3	A2	A1	A0	B3	B2	B1	B0	A>B	A=B	A<B
0	0	1	0	1	1	0	1	0	0	1
1	1	1	1	1	1	1	0	1	0	0
1	0	0	1	1	0	0	1	0	1	0
0	1	0	1	0	0	0	0	1	0	0

RESULT: Constructed the 1-bit / 2-bit Comparator and Magnitude comparator and verified the truth Table.

Possible viva questions

1) What is a comparator?

2) What are the applications of comparator?

3) Derive the Boolean expressions of one bit comparator and two bit comparators.

4) How do you realize a higher magnitude comparator using lower bit comparator?

5) Design a 2 bit comparator using a single Logic gates?

6) Design an 8 bit comparator using a two numbers of IC 7485?

Applications:

Arithmetic Logic Units & Processing units.

5. Truth table verification of Flip-Flops: (i) JK Master slave (ii) T type and (iii) D type.

1) JK Master Slave Flip Flop 2)T type Flip Flop.3) D type Flip Flop.

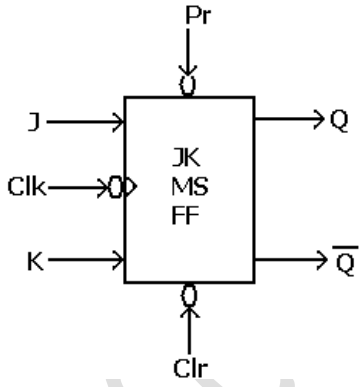
COMPONENTS REQUIRED:-

IC 7408, IC 7404, IC 7402, IC 7400, Patch Cords & IC Trainer Kit.

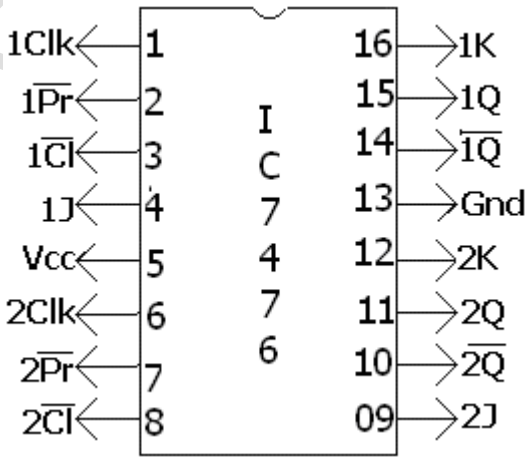
THEORY: Logic circuits that incorporate memory cells are called *sequential logic circuits*; their output depends not only upon the present value of the input but also upon the previous values. Sequential logic circuits often require a timing generator (a clock) for their operation. The latch (flip-flop) is a basic bi-stable memory element widely used in sequential logic circuits. Usually there are two outputs, Q and its complementary value.

The flip-flops can be mode to respond to trailing edge of a pulse by employing two flip-flop circuits, one to hold the output state on the trailing edge and the other to sample the i/p information on the leading edge. Such a combination is called a master- slave flip-flop. The MS combination can be constructed for any type of FF. In case of JK MS FF the information present at the J and K inputs is transmitted to the master FF on the leading edge of the clk pulse and held there until the trailing edge of clock pulse occurs offer which if is allowed to pass through to the slave FF.

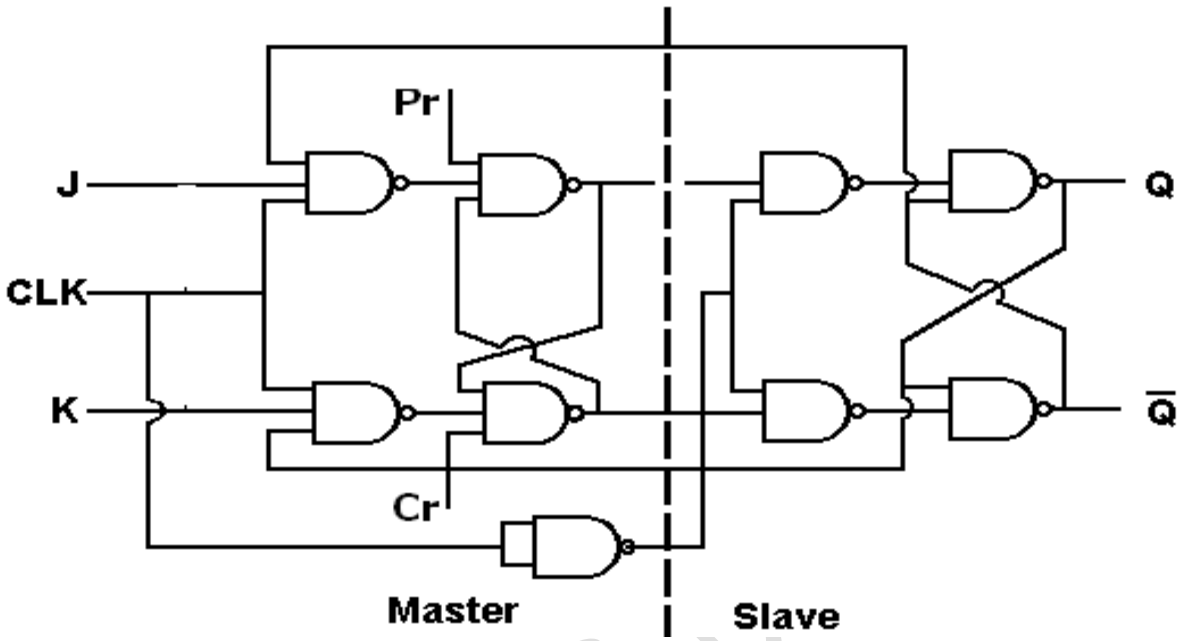
SYMBOL OF MS-JK FF



PIN DIAGRAM



Master Slave JK Flip flop
(Note: PR = Cr =1)

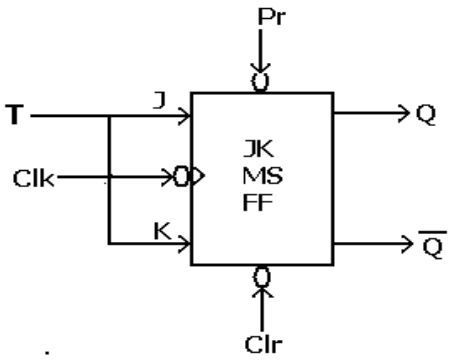


TRUTH TABLE of Master Slave JK Flip Flop

Inputs			Output	COMMENTS
Clk	J	K	Qn+1	
↓	X	X	Qn	No Change
↓	0	0	Qn	No change
↓	0	1	0	Reset
↓	1	0	Qn	Set
↓	1	1	$\overline{Qn}$	F.F. Preset(Set)

T- Type FF using MS JK FF

SYMBOL

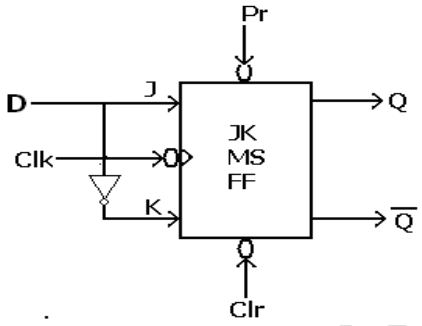


Truth Table of T FF

Inputs CLK	Tn	Output Qn+1	Comments
0	X	Qn	No change
↓	0	Qn	No change
↓	1	$\overline{Qn}$	Toggles

1) D - Type FF using MS JK FF

SYMBOL



Truth Table of D FF

Inputs CLK	Dn	Output Qn+1	Comments
0	X	Qn	No change
↓	0	0	Data tarnsfered
↓	1	1	Data tarnsfered

RESULT: Verified the truth table of JK Master Slave and D and T type of Flip Flop

Possible viva questions:

1. What is the difference between Flip-Flop & latch?
2. Give examples for synchronous & asynchronous inputs?
3. What are the applications of different Flip-Flops?
4. What is the advantage of Edge triggering over level triggering?
5. What is the relation between propagation delay & clock frequency of flip-flop?
6. What is race around in flip-flop & how to over come it?
7. Convert the J K Flip-Flop into D flip-flop and T flip-flop?
8. List the functions of asynchronous inputs?

Applications:

Memory elements, Timers , Counters and Delay units.

6. Design and Realization of 3 bit counters as a sequential circuit, Ring counter and Johnson counter

ASYNCHRONOUS COUNTER

COMPONENTS REQUIRED: - IC 7476, Patch Cords & IC Trainer Kit

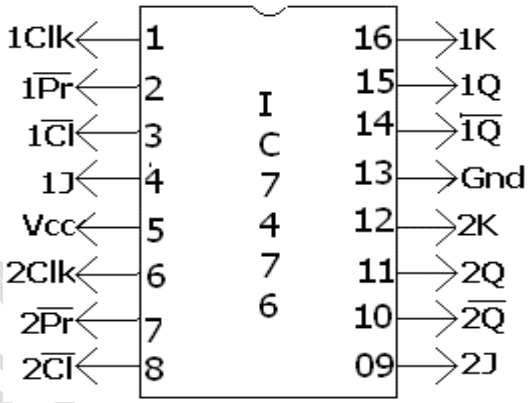
THEORY: A sequential circuit that gives through a prescribed sequence of states upon the application of input pluses is called counters. The straight binary sequence counter is the simple and most straight forward. An n-bit binary counter has n flip-flops and can count in binary from 0 to 2^n -1. A counter in which each flip-flop is triggered by the output goes to previous flip-flop.

As all the flip-flops do not change state simultaneously spike occur at the output. To avoid this, strobe pulse is required. Because of the propagation delay the operating speed of asynchronous counter is low. Asynchronous counter are easy and simple to construct

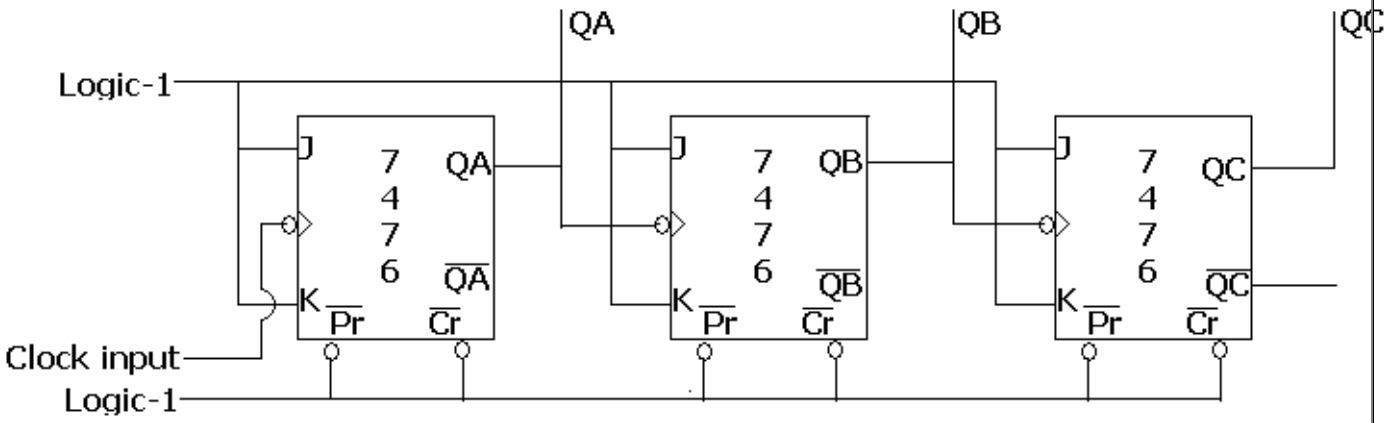
ASYNCRONOUS COUNTERS A binary ripple (Asynchronous) counter consists of series connections of T- flip-flops without any logic gates. Each FF is triggered by the output of its preceding FF goes from 1 to 0.

Realization of 3-bit binary counters using IC7476

PIN DIAGRAM



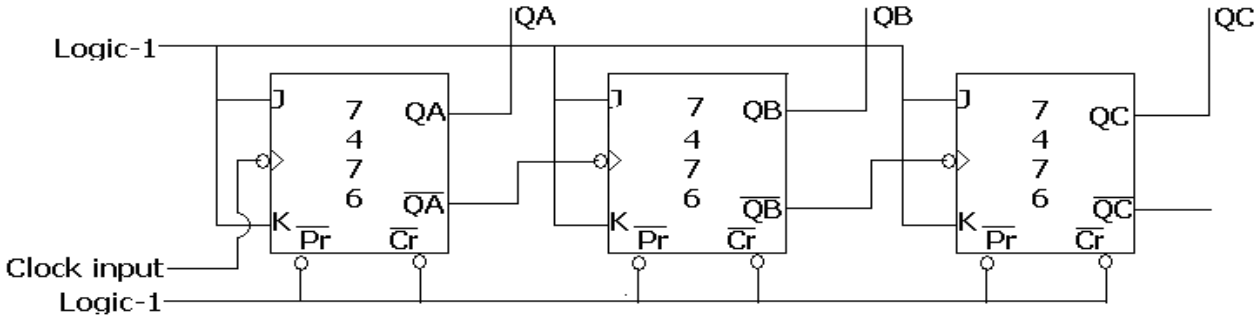
Asynchronous UP Counter (MOD-8)



TRUTH TABLE

Number of clock pulses	Flip Flop outputs		
	Qc	Qb	Qa
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0

Asynchronous DOWN Counter (MOD-8)



TRUTH TABLE

Number of clock pulses	Flip Flop outputs		
	Qc	Qb	Qa
0	1	1	1
1	1	1	0
2	1	0	1
3	1	0	0
4	0	1	1
5	0	1	0
6	0	0	1
7	0	0	0
8	1	1	1

Possible Viva Questions

1. What is an asynchronous counter?
2. How is it different from a synchronous counter?
3. Realize asynchronous counter using T flip flop.

Applications:

Frequency dividers , Delay units, Timers, Real time counters

SYNCHRONOUS COUNTER

THEORY:

A counter in which each flip-flop is triggered by the output goes to previous flip-flop. As all the flip-flops do not change states simultaneously in asynchronous counter, spike occur at the output. To avoid this, strobe pulse is required. Because of the propagation delay the operating speed of asynchronous counter is low. This problem can be solved by triggering all the flip-flops in synchronous with the clock signal and such counters are called synchronous counters.

UP COUNTER

Design and Realization of 3 Bit Synchronous Counter Using IC7476

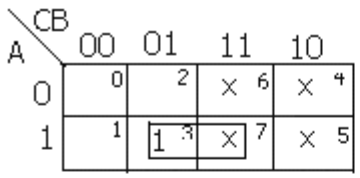
Excitation table

Present state output	**Next State output**	**J**	**K**
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

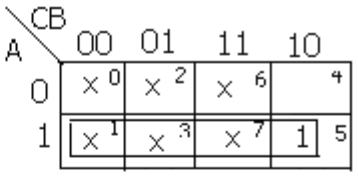
MOD-6 Synchronous UP COUNTER

In MOD-6 counter invalid state is 110

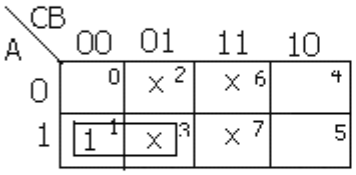
PRESENT STATE			NEXT STATE			EXCITATION					
Qc	Qb	Qa	Qc	Qb	Qa	JC	KC	JB	KB	JA	KA
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	0	0	0	X	1	0	X	X	1



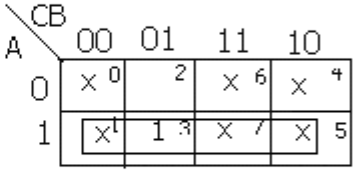
Jc=QbQa



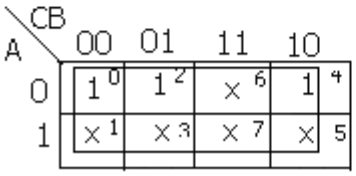
Kc=Qa



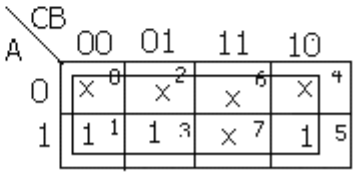
Jb=$\overline{Q}$cQa



Kb=Qa

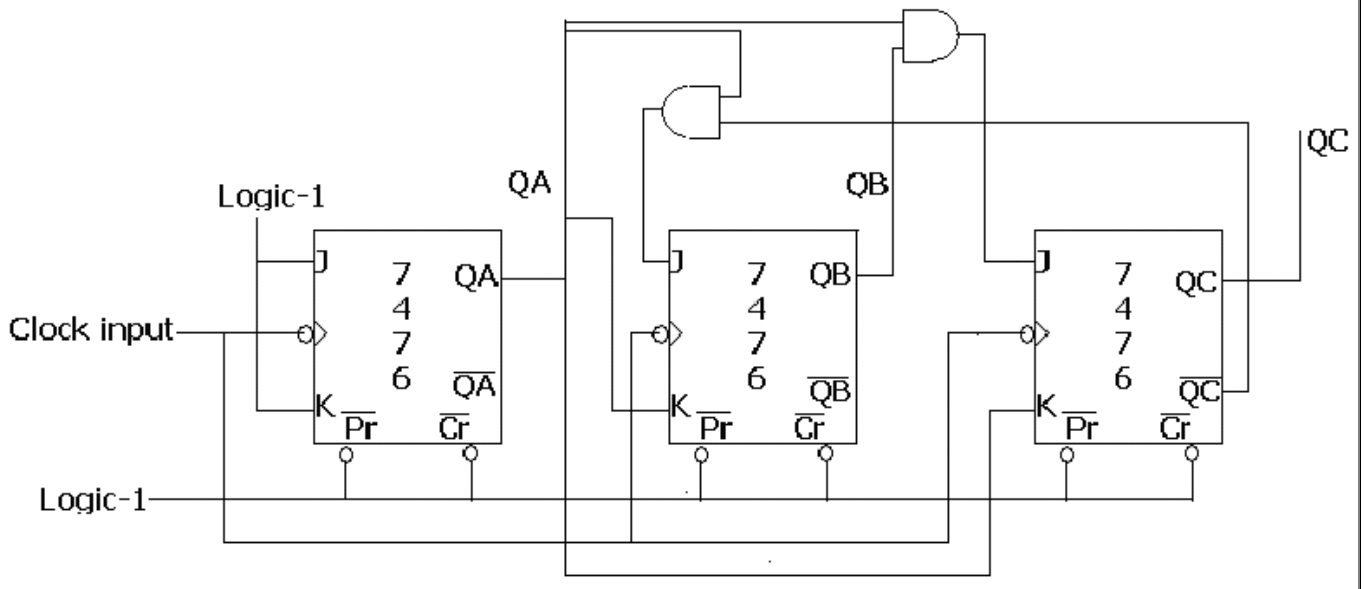


Ja=1



Ka=1

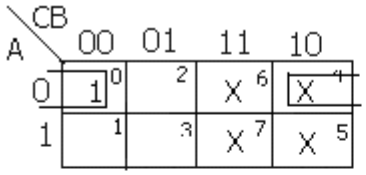
Circuit Diagram



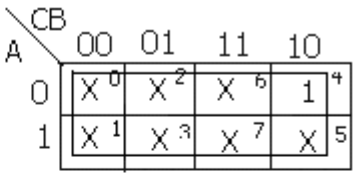
Synchronous Down Counter

PRESENT STATE			NEXT STATE			FLIP-FLOPS					
Qc	Qb	Qa	Qc	Qb	Qa	Jc	Kc	Jb	Kb	Ja	Ka
1	0	0	0	1	1	X	1	1	X	1	X
0	1	1	0	1	0	0	X	X	0	X	1
0	1	0	0	0	1	0	X	X	1	1	X
0	0	1	0	0	0	0	X	0	X	X	1
0	0	0	1	0	0	1	X	0	X	0	X

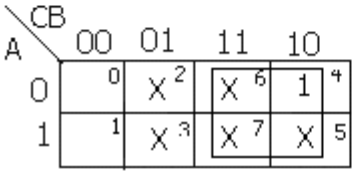
Simplifications



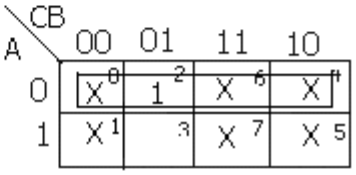
$Jc = \overline{Qa}\overline{Qb}$



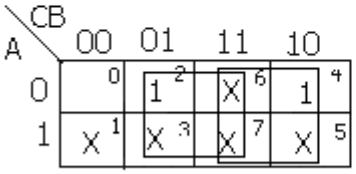
Kc = 1



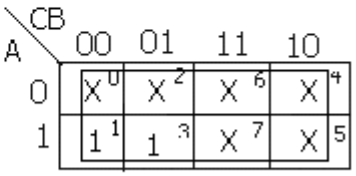
Jb = Qc



Kb = Qa

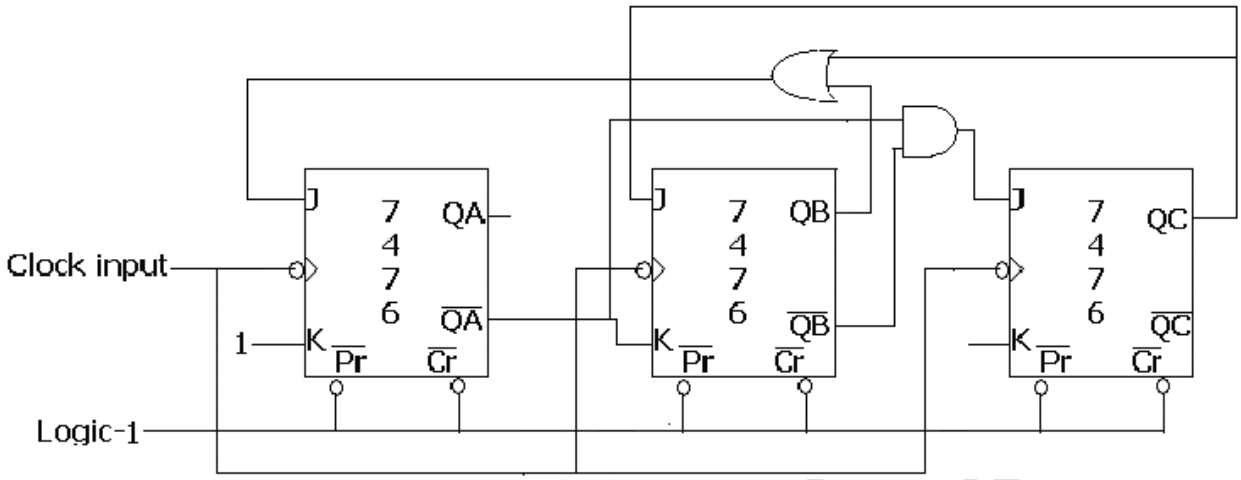


Ja = Qb+Qc



Ka = 1

Circuit Diagram



RESULT: The working of Synchronous counters is verified.

RING COUNTER AND JOHNSON COUNTER

To Realize and study Ring counter/Johnson counter

COMPONENTS REQUIRED: - IC 7495,Patch cords, Trainer Kit

THEORY:

Ring counter is a basic register with direct feedback such that the contents of the register simply circulate around the register when the clock is running. Here the last output that is Q_D in a shift register is connected back to the serial input.

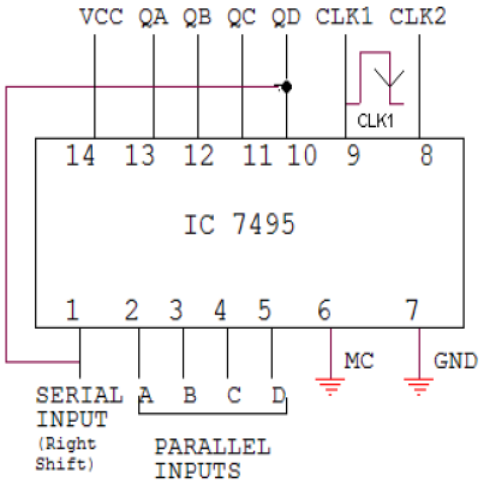
A basic ring counter can be slightly modified to produce another type of shift register counter called Johnson counter. Here complement of last output is connected back to the not gate input and not gate output is connected back to serial input. A four bit Johnson counter gives 8 state output.

PROCEDURE:

1. Connect the circuit as in the figure
2. Load the parallel input data in to the shift register (ie to make Qa Qb Qc Qd = parallel input data)
3. Connect mode control pin no 6 to '1' state the parallel inputs A B C D to be loaded in to shift register are given to pin no 2,3,4,5 respectively. Clock is applied to clk1(pin no. 9)
4. Clk 2 (pin no 8) is pulsed once now A B C D parallel inputs appeares in the respective out puts.
5. To circulate this parallel input data connect mode control (pin no 6) to '0' state ,apply clk pulse at clock '1' pin no 9 and observe the outputs.

CIRCUIT DIAGRAM:

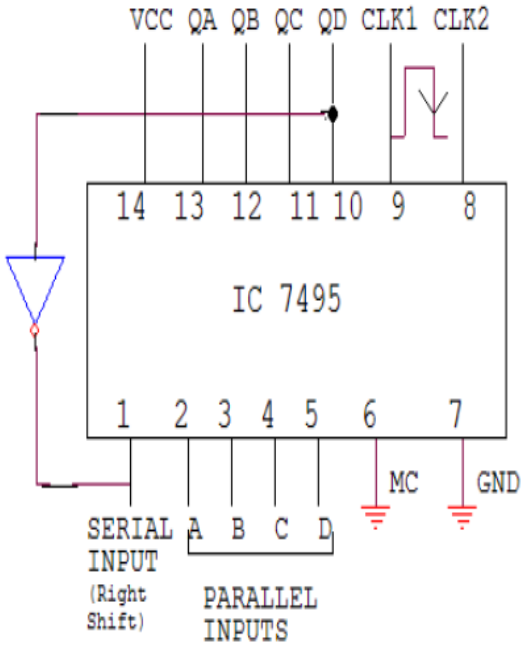
1) RING COUNTER



TRUTH TABLE

Clock pulses	Q_A	Q_B	Q_C	Q_D
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0
5	0	1	0	0
6	0	0	1	0
7	0	0	0	1
8	1	0	0	0

2) JOHNSON COUNTER



TRUTH TABLE

Clock pulses	Q_A	Q_B	Q_C	Q_D
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0

RESULT: The truth table & working of Ring and Johnson counters is verified

Applications:

Ring counter:

Ring counters are used to count the data in a continuous loop.

They are used to detect the various numbers values or various patterns within a set of information, by connecting AND & OR logic gates to the ring counter circuits.

They are also used in frequency divider circuits.

Johnson counter:

Johnson counter is used as multiphase square wave generator

Probable Viva Questions:

1.What are synchronous counters?

2.What are the advantages of synchronous counters?

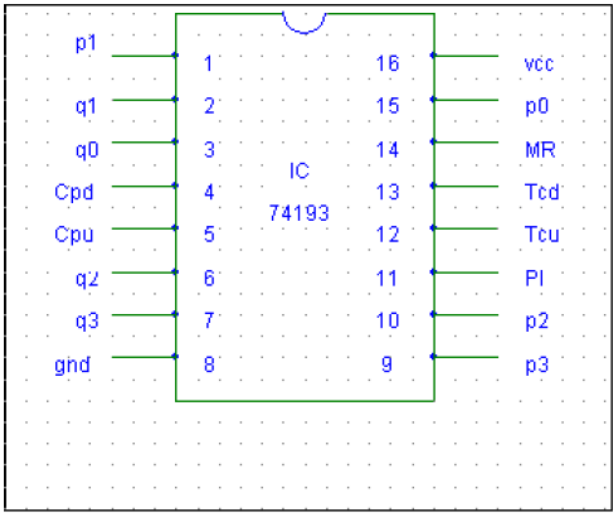
3.What is an excitation table?

4.Write the excitation table for D,T FF.

5.Design mod-5 synchronous counter using TFF

To realize a MOD-N counter using IC-74193

PIN DETAILS OF IC 74193



1. P1,P2,P3 and P0 are parallel data inputs
2. Q0,Q1,Q2 and Q3 are flip-flop outputs
3. MR: Asynchronous master reset
4. PL: Asynchronous parallel load(active low) input
5. TCd : Terminal count down output
6. TCu : Terminal count up output

Up counter

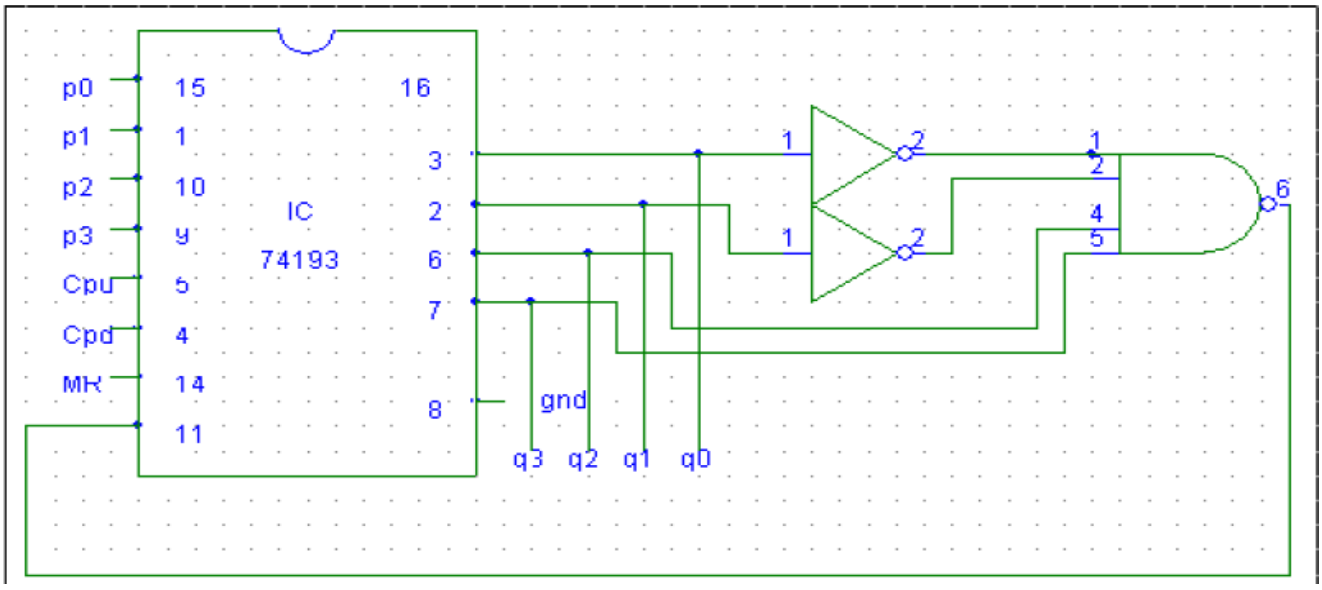
CLK	Q_D	Q_C	Q_B	Q_A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

Down counter

CLK	Q_D	Q_C	Q_B	Q_A
0	1	1	1	1
1	1	1	1	0
2	1	1	0	1
3	1	1	0	0
4	1	0	1	1
5	1	0	1	0
6	1	0	0	1
7	1	0	0	0
8	0	1	1	1
9	0	1	1	0
10	0	1	0	1
11	0	1	0	0
12	0	0	1	1
13	0	0	1	0
14	0	0	0	1
15	0	0	0	0
16	1	1	1	1

b) Design up counter for preset value 0010 and N=10

CIRCUIT DIAGRAM

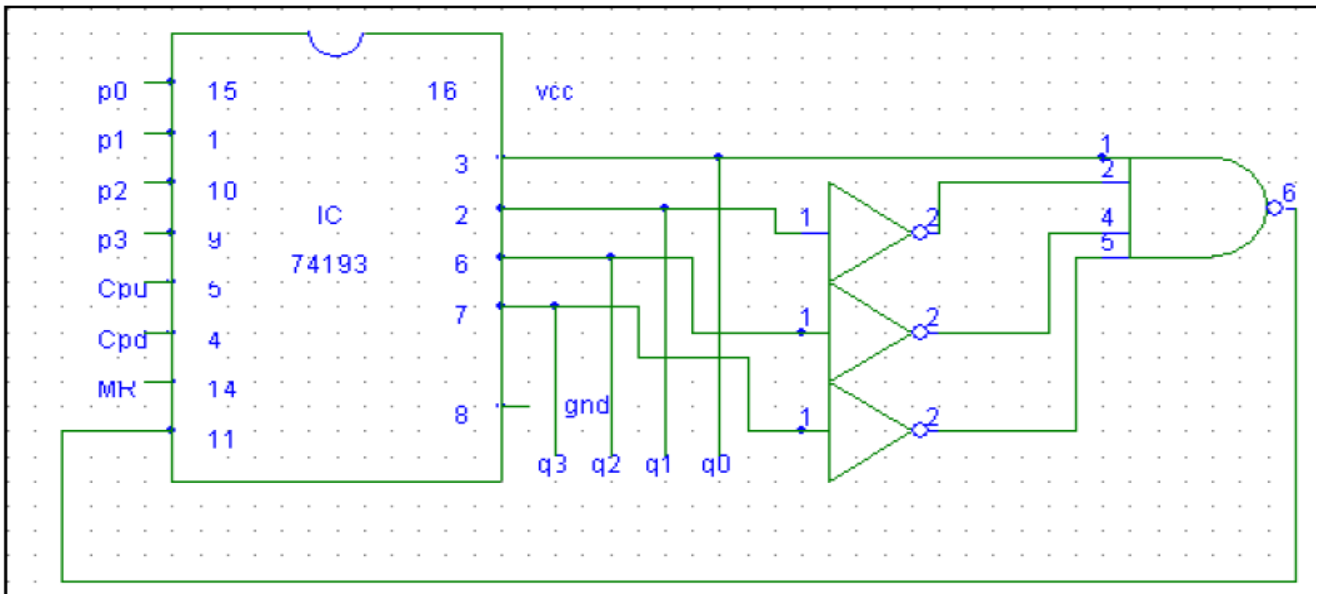


TRUTH TABLE

CLK	Q_D	Q_C	Q_B	Q_A
1	0	0	1	0
2	0	0	1	1
3	0	1	0	0
4	0	1	0	1
5	0	1	1	0
6	0	1	1	1
7	1	0	0	0
8	1	0	0	1
9	1	0	1	0
10	1	0	1	1
11	1	1	0	0
12	0	0	1	0

c) Design of down counter for preset value 1011 and N=10

CIRCUIT DIAGRAM



TRUTH TABLE

CLK	Q_D	Q_C	Q_B	Q_A
1	1	0	1	1
2	1	0	1	0
3	1	0	0	1
4	1	0	0	0
5	0	1	1	1
6	0	1	1	0
7	0	1	0	1
8	0	1	0	0
9	0	0	1	1
10	0	0	1	0
11	0	0	0	1
12	1	0	1	1

Result: The working of IC 74193 as an up/down presettable counter is verified.

Probable Viva Questions:

1.What is a presettable counter?

2.What are the applications of presettable counters?

3.Expalin the working of IC 74193.

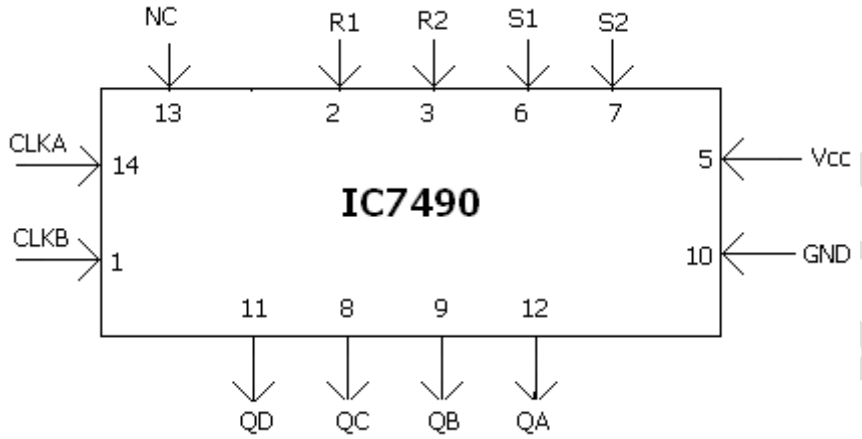
4.Write the circuit for preset value of 0100 and N=5(up counter)

Applications:

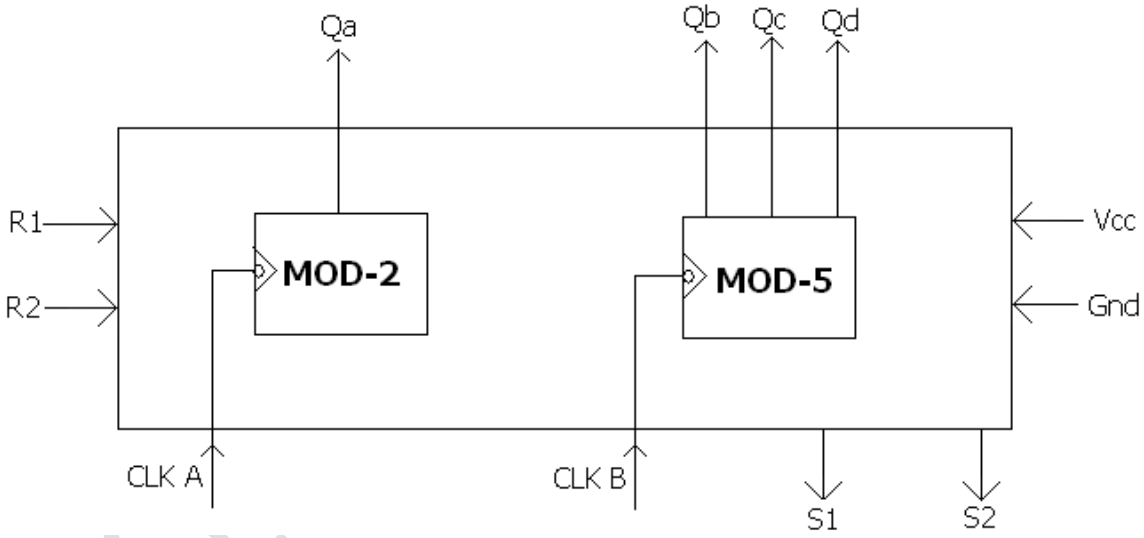
Frequency dividers,Delay units, Timers, Real time counters

Realize a MOD-N counter using IC-7490

Pin Diagram



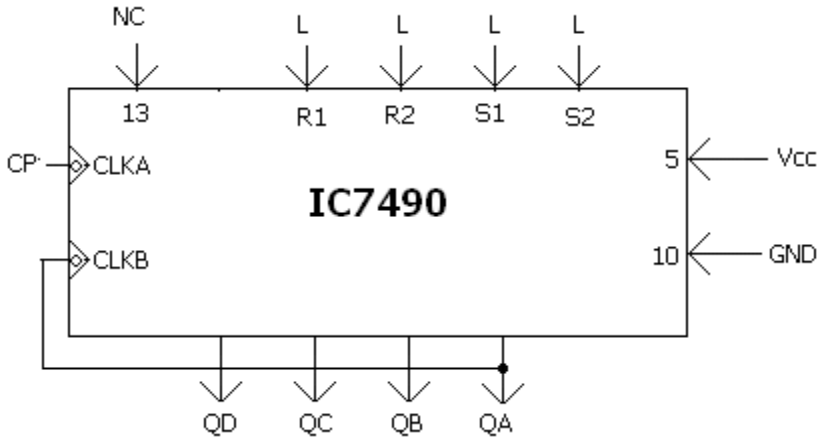
Internal Diagram



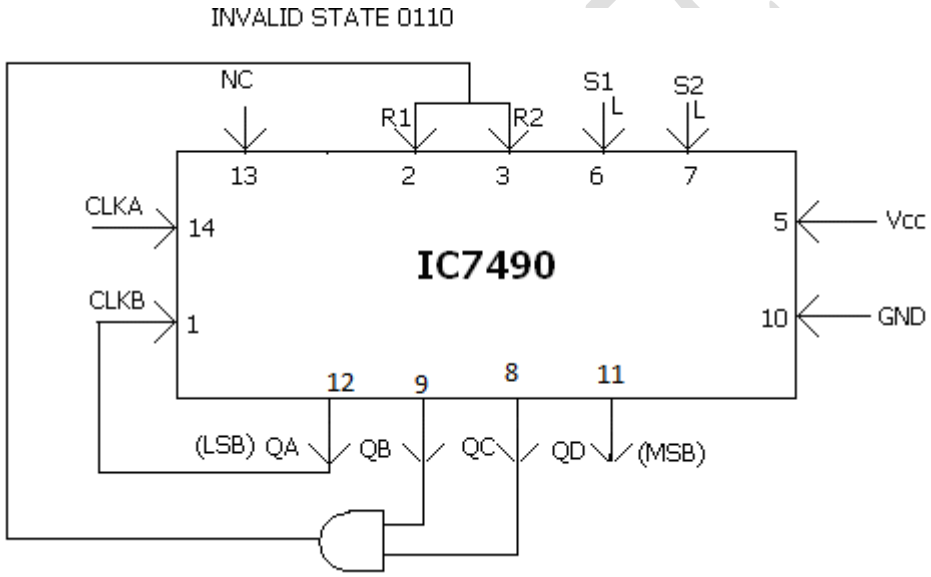
Conditional Table

R1	R2	S1	S2	Qa	Qb	Qc	Qd
H	H	L	X	L	L	L	L
H	H	X	L	L	L	L	L
X	L	H	H	1	0	•	1
L	X	L	X	MOD-2 COUNTER			
X	L	X	L	MOD-5 COUNTER			

7490 As Mod-10 Counter



7490 As Mod-6 Counter



RESULT: The working of IC 7490 is verified.

Probable Viva Questions:

1.What is a decade counter?

2.What do you mean by a ripple counter?

3.Explain the design of Modulo-N counter(N<=9) using IC 7490

Applications:

Frequency dividers,Delay units, Timers, Real time counters

7. Shift left, Shift right, SIPO, SISO, PISO, PIPO operations using 7495

To realize and study of Shift Register.

1) SISO (Serial in Serial out)

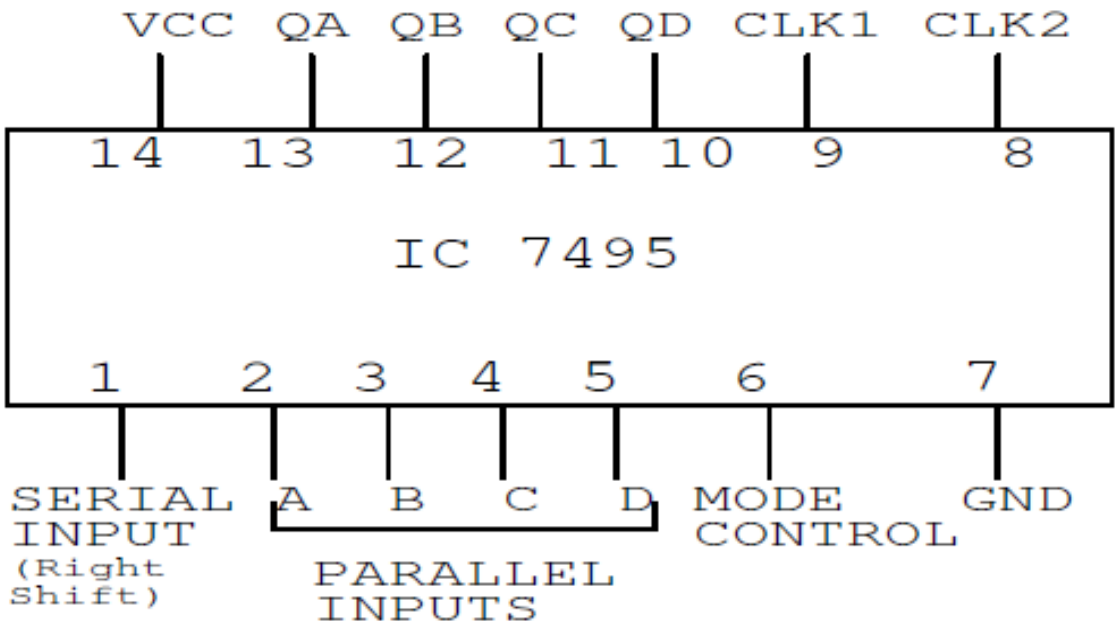
2) SIPO (Serial in Parallel out)

3) PIPO (Parallel in Parallel out)

4) PISO (Parallel in Serial out)

COMPONENTS REQUIRED: IC 7495, Patch Cords & IC Trainer Kit.

PIN DIAGRAM OF IC 7495



M=1 for parallel operation

M=0 for serial operation

Procedure:

SERIAL IN SERIAL OUT / SERIAL IN PARALLEL OUT (SISO) / (SIPO)

1. connect mode control(pin no 6) to '0' and apply serial data at serial Input at pin no 1 terminal storing from LSB
2. Apply clock pulses at clock one in (pin no 9) ,terminal after each Data bit and observe outputs
3. verify its operation as a right shift register

1) SERIAL IN SERIAL OUT (SISO) (Right Shift)

Serial i/p data	Shift Pulses	Q_A	Q_B	Q_C	Q_D
-	-	X	X	X	X
0	t1	0	X	X	X
1	t2	1	0	X	X
0	t3	0	1	0	X
1	t4	1	0	1	0
X	t5	X	1	0	1
X	t6	X	X	1	0
X	t7	X	X	X	1
X	t8	X	X	X	X

2) SERIAL IN PARALLEL OUT (SIPO)

Serial i/p data	Shift Pulses	Q_A	Q_B	Q_C	Q_D
-	-	X	X	X	X
0	t1	0	X	X	X
1	t2	1	0	X	X
0	t3	0	1	0	X
1	t4	**1**	**0**	**1**	**0**

PARALLEL IN SERIAL OUT (PISO)

(i) to load the parallel input data in to the shift register (ie to make Qa Qb Qc Qd = parallell input data)

(ii) connect mode control pin no 6 to '1' state the parallel inputs A B C D to be loaded in to shift register are given to pin no 2,3,4,5 respectively.

(iii) Clck 2 (pin no 8) is pulsed once now A B C D parallel inputs appeares in the respective out puts.

(iv) To convert this parallel input data in to serial out put data on Qd line first connect mode control (pin no 6) to '0' state ,apply clk pulse at clock '1' pin no 9 and observe the outputs.

PARALLEL IN PARALLEL OUT (PIPO)

(i) connect mode control pin no 6 to '1' state; the parallel inputs A B C D to be loaded in to shift register are given to pin no 2,3,4,5 respectively. Clk applied at clk2(pin no. 8)

(ii) Observe the outputs at Qa, Qb, Qc, and Qd.

3) PARALLEL IN PARALLEL OUT (PIPO)

Clock Input Terminal	Shift Pulses	Q_A	Q_B	Q_C	Q_D
-	-	X	X	X	X
CLK_2	t1	A	B	C	D

4) PARALLEL IN SERIAL OUT (PISO)

Clock Input Terminal	Shift Pulses	Q_A	Q_B	Q_C	Q_D
-	-	X	X	X	X
CLK_2	t1	1	0	1	0
CLK_1	t2	X	1	0	1
0	t3	X	X	1	0
1	t4	X	X	X	1
X	t5	X	X	X	X

Result: The various operations of a shift register is verified.

Applications: Memories

EXPERIMENTS (VERILOG HDL)

8. Write a HDL code to describe the functions of a Adders / Subtractors using three modeling styles: Behavioral, structural and data flow

Procedure:

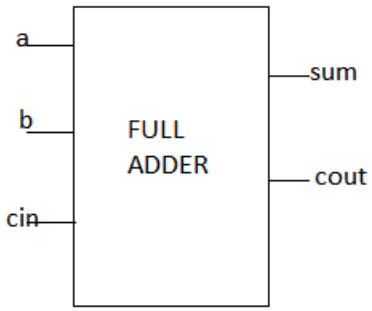
1. Draw logic diagram or block diagram
2. Write the truth table
3. Write Verilog Code
4. HDL entry: Refer the following steps to enter the /Verilog code into the Xilinx tool.
   - ✓ Create new project: refer step 3 **Create an HDL Source** of *ISE Quick Start Tutorial.*
   - ✓ Create new file VHDL source file.
     - Entity name: logic_gates
     - Port Details: a, b Inputs of type std_logic
       - y(0),y(1),y(2),y(3),y(4),y(5),y(6) Outputs of type std_logic
   - ✓ After completion of code, compile the file. If the design is error free, go to next step.
5. Simulation: For more details refer step 3 **Create an HDL Source** of *ISE Quick Start Tutorial.*
   - ✓ Create new test bench file, make the test bench as top level and simulate the design.
   - ✓ Analyze the waveform to check if your design is correct. If outputs are not as expected go back to the source file and make necessary correction.
   - ✓ Trace the waveform
6. Synthesis: Run synthesis, if the design is error free, go to next step. For more details about the steps refer *ISE Quick Start Tutorial.*
7. Implementation:
   - ✓ Create implementation constraint file
     - File name: logic_gates_ucf
   - ✓ Assign Package pins.
8. Programming: Configure the Device, the downloading the bit file to chip (FPGA or CPLD).
9. Test your design by comparing with your truth table.
10. Write conclusion about experiment.

References:

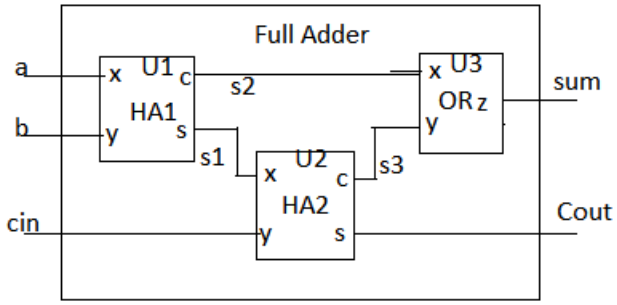
- 1.4 Operators page no 7,8,9,10,11,12,13,14 of text book HDL programming VHDL and Vrilog Nazeih M. Botros.
- Basics of all logic gates from logic design text book
- www.xilinx.com for details about tools and board

STRUCTURAL

Structural descriptions model the system as components or gates. This description is identified by presence of the key word component in the architecture (VHDL) or gate construct, such as and, or, not in the module (Verilog). If the VHDL architecture or Verilog module consists only of components or gates; the style is coined as pure structural.



Block Digram



Strectural Block Digram

DATA FLOW

Verilog Program:

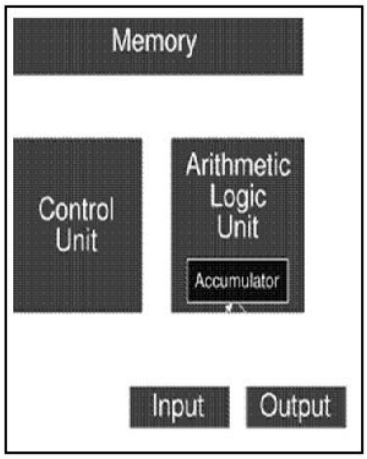
module fulladder (a, b, cin, sum, cout);
input a, b, cin;
output sum, cout;
assign sum = a^b^cin;
assign cout = (a&b)|(b&cin)|(a&cin);
endmodule

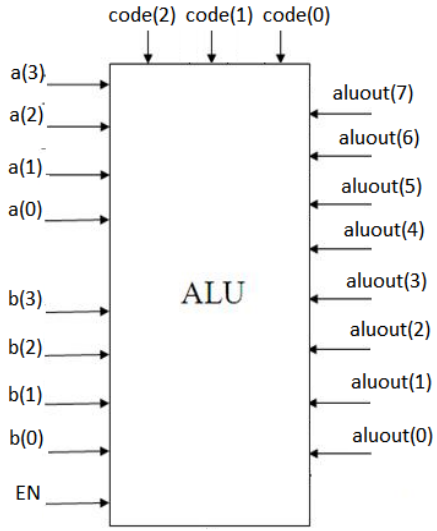
BEHAVIRIOL

Verilog Program:

module fulladder_BH (a, b, cin, sum, cout);
input a, b, cin;
output sum, cout;
reg sum, cout;
wire x;
assign x = {a, b, cin};
always @(x)
begin
if(x==3'B001|x==3'B010|x==3'B100|x==3'B111)
sum = 1'B1; else sum = 1'B0;
if(x==3'B011|x==3'B101|x==3'B110|x==3'B111)
cout = 1'B1; else cout = 1'B0;
end
endmodule

9. Design and implement ALU using the Verilog





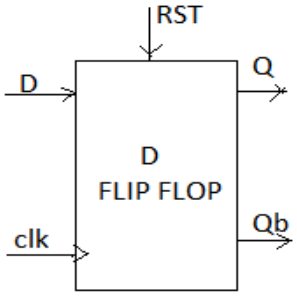
code	aluout
000	a + b
001	a – b
010	~a
011	a * b
100	a&b
101	a|b
110	~(a&b)
111	~(a|b)

Central Processing Unit (CPU)

10.Develop the HDL code for the following flip-flops - SR,JK, D and T.

Clk	RST	D	Q	Qb	Description
↑	1	X	0	1	Memory no change
↑	0	0	0	1	Reset Q » 0
↑	0	1	1	0	Set Q » 1

Note: ↑ indicates direction of clock pulse as it is assumed D flip-flops are edge triggered.

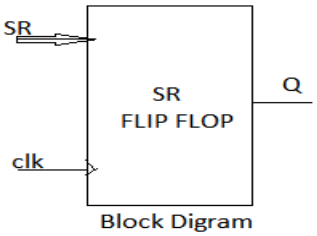


Verilog Program:

```
module dff(d, rst, clk, q, qb);
input d, rst, clk;
output q, qb;
reg q, qb;
always @ (posedge clk)
begin
        if (rst == 1'B1)
        begin
                q = 1'B0; qb = 1'B1;
        end
        else
        begin
                q = d; qb = ~q;
        end
end
endmodule
```

Result:

SR-FF

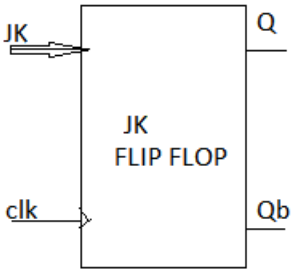


Block Digram

clk	SR	Q
↑	00	Q
↑	01	0
↑	10	1
↑	11	invalid

Transition Table

JK-FF



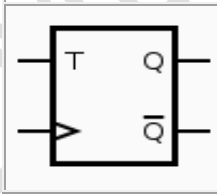
Block Digram

clk	JK	Q
↑	00	Q
↑	01	0
↑	10	1
↑	11	toggle

Transition Table

Verilog
module jkff(jk, clk, q, qb);
input clk;
input [1:0]jk;
output q,qb;
reg q,qb;
always @(posedge clk)
begin
case(jk)
2'b00 : q = q ;
2'b01 : q = 1'b0;
2'b10 : q = 1'b1 ;
2'b11 : q = ~q ;
default:q=1'BZ;
endcase
end
endmodule

T-FF

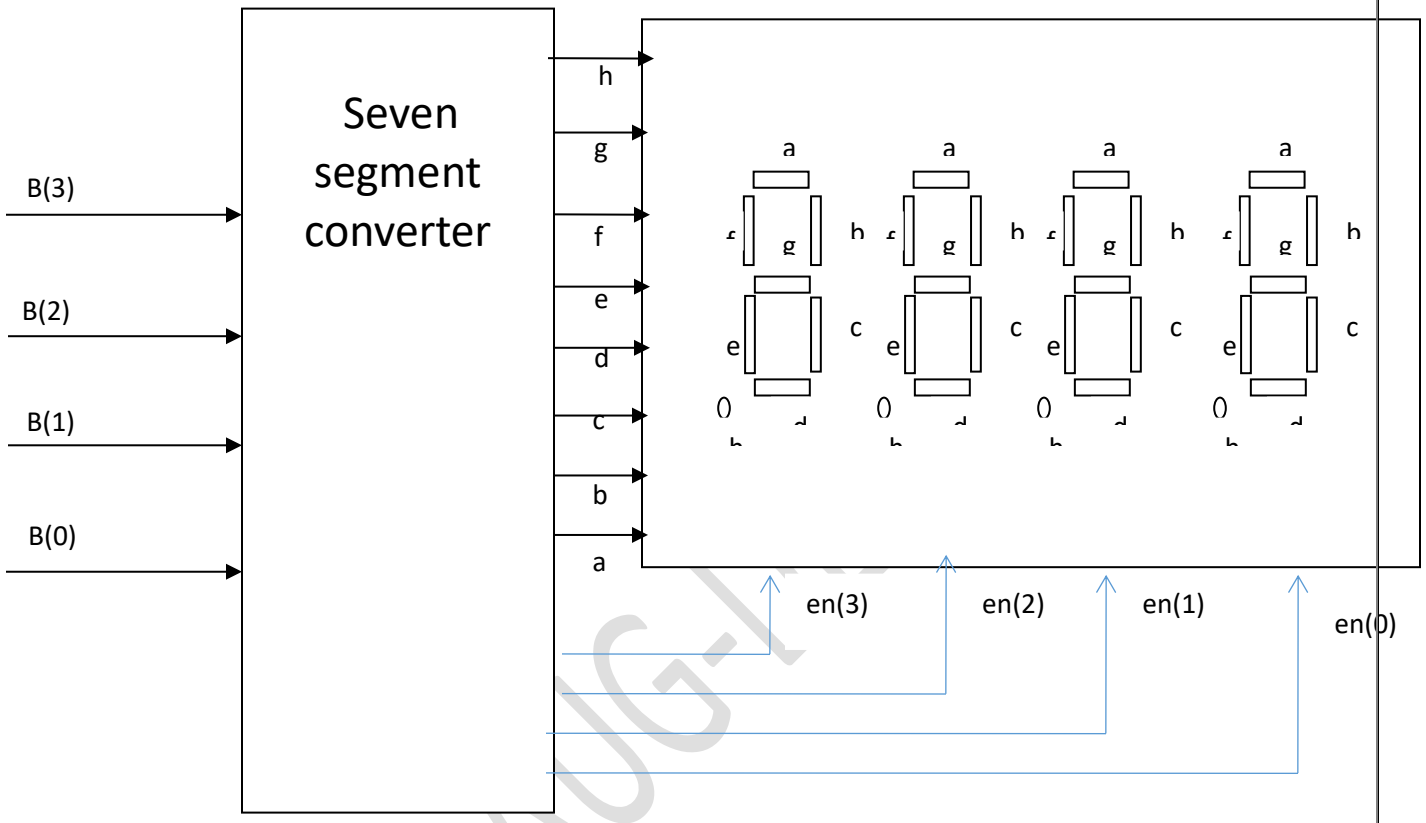


	Clk	T	Q	Qb
	↑	0	Q	Qb
	↑	1	Qb	Q

Verilog Program:
module tff(t, clk, q, qb);
input t, clk;
output q, qb;
reg q, qb;
always @(posedge clk)
begin
if (t == 1'B0)
begin
q = q; qb = qb;
end
else
begin
q = ~q; qb = ~qb;
end
end
endmodule
Result:

11. Write HDL code to display messages on the given seven segment display, LCD and accepting Hex key pad input data.

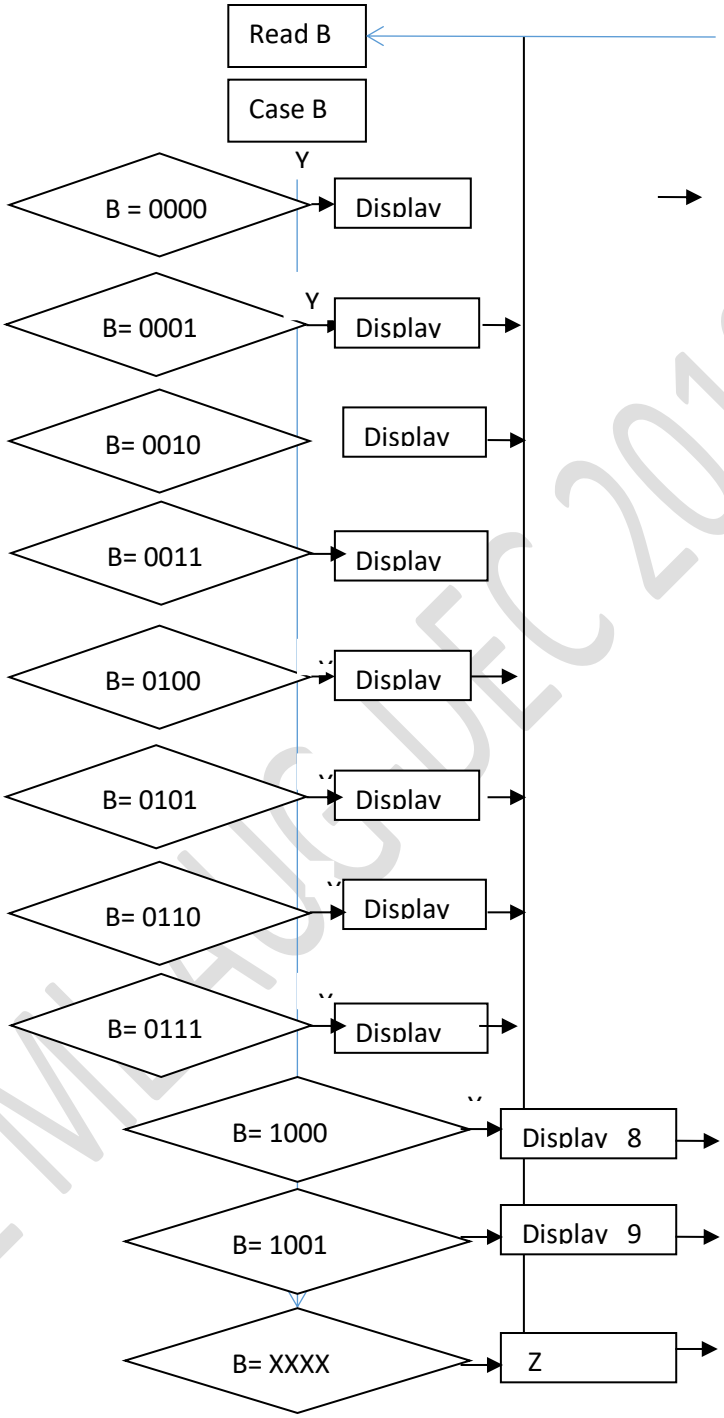
Block Diagram



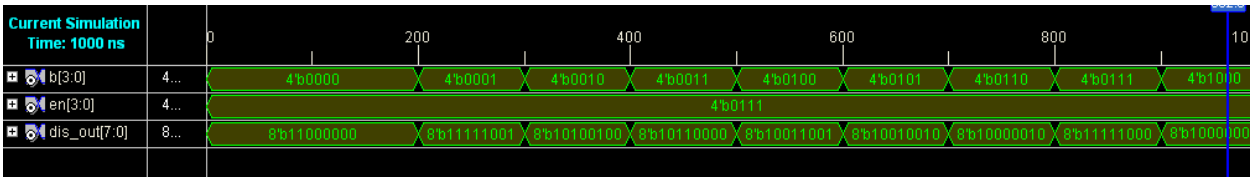
Truth table:

B	dis_out(hgfedcba)	
0000	11000000	0
0001	11111001	1
0010	10100100	2
0011	10110000	3
0100	10011001	4
0101	10010010	5
0110	10000010	6
0111	11111000	7
1000	10000000	8
1001	10010000	9
1010	10001000	A
1011	10000011	B
1100	11000110	C
1101	10100001	D
1110	10000110	E
1111	10001110	F

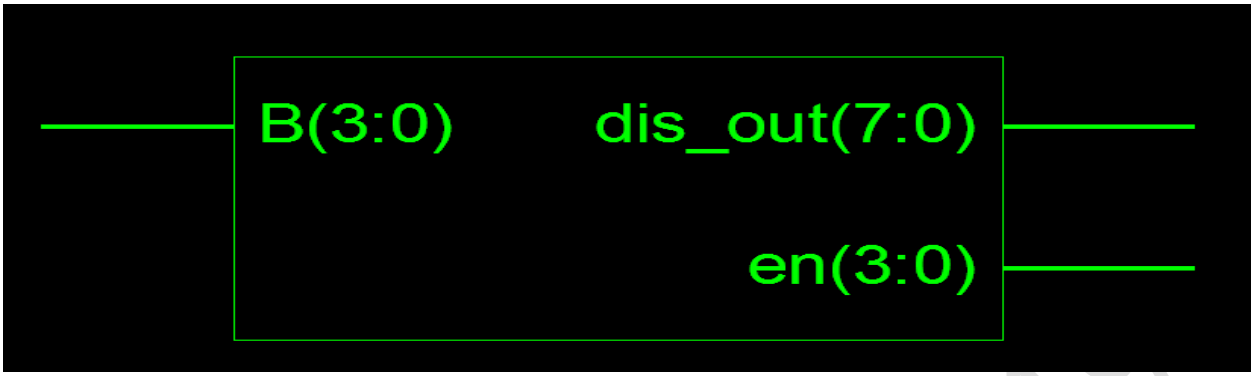
Flow Chart:



Simulation:



Synthesis

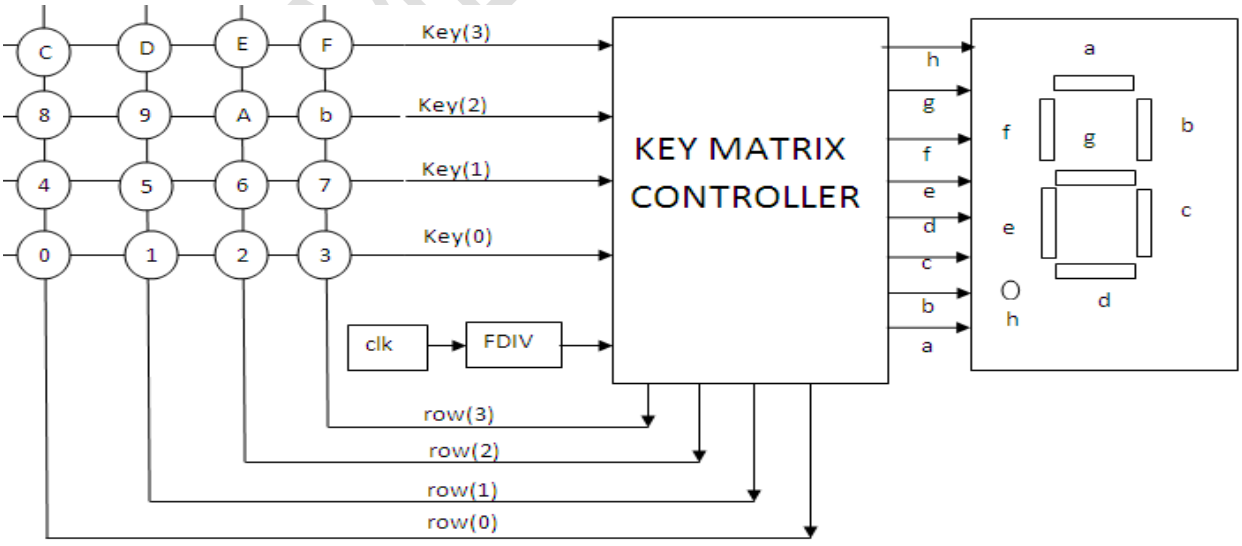


Implementation Details (UCF)

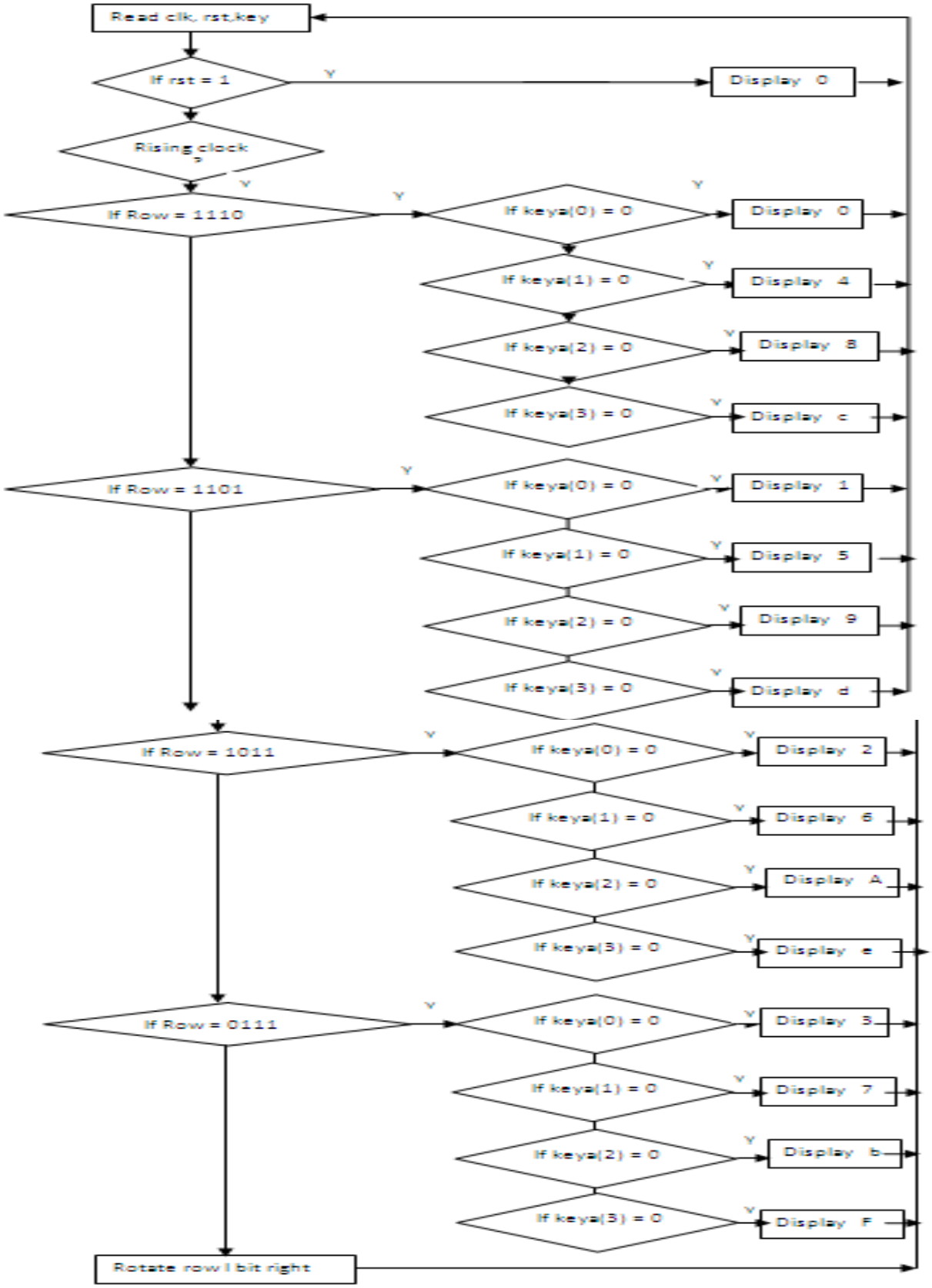
```
NET "dis_out<0>" LOC = "p10" ;
NET "dis_out<1>" LOC = "p11" ;
NET "dis_out<2>" LOC = "p12" ;
NET "dis_out<3>" LOC = "p13" ;
NET "dis_out<4>" LOC = "p15" ;
NET "dis_out<5>" LOC = "p16" ;
NET "dis_out<6>" LOC = "p18" ;
NET "dis_out<7>" LOC = "p19" ;
NET "en<0>" LOC = "p9" ;
NET "en<1>" LOC = "p7" ;
NET "en<2>" LOC = "p3" ;
NET "en<3>" LOC = "p2" ;
NET "B<0>" LOC = "p35" ;
NET "B<1>" LOC = "p29" ;
NET "B<2>" LOC = "p27" ;
NET "B<3>" LOC = "p21" ;
```

Key matrix

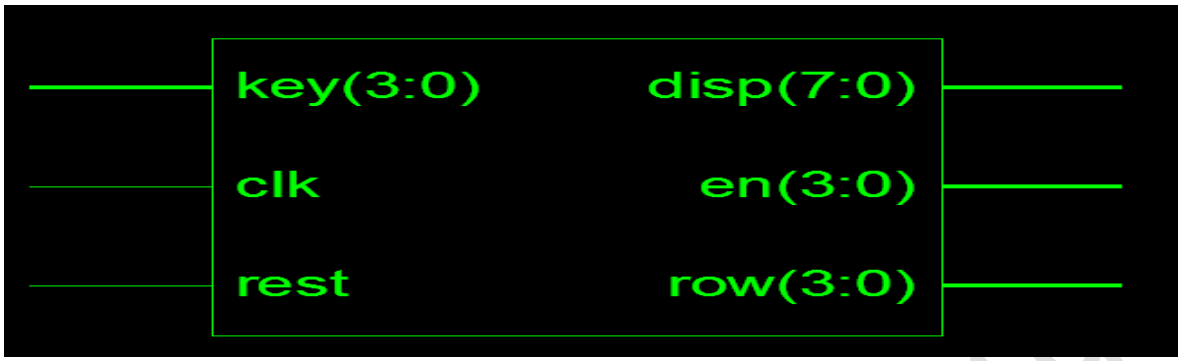
A key matrix is used when lots of keys are needed, or a keypad is being used. A key matrix has the following schematic.



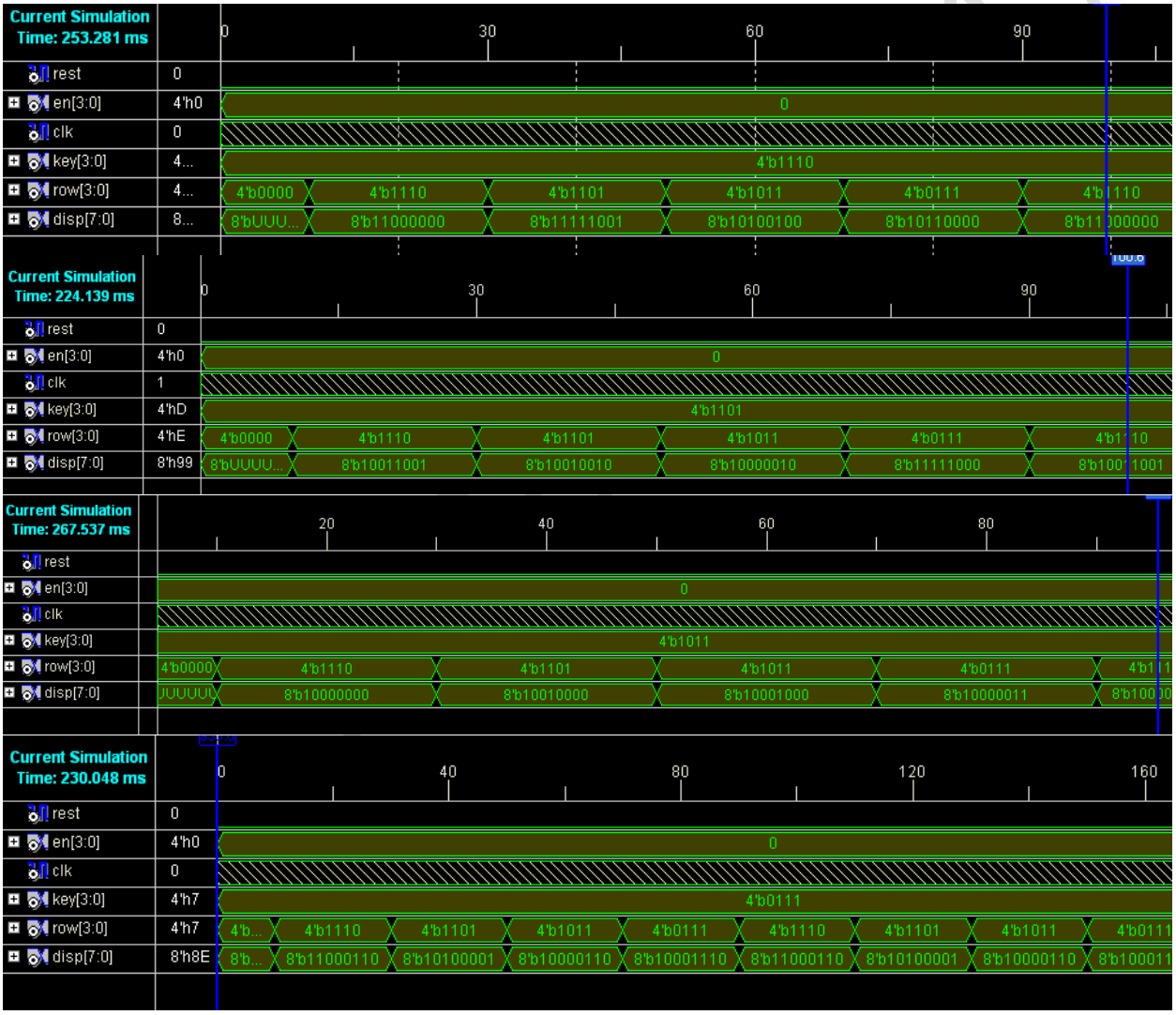
Flow chart


Read clk, rst,key
If rst = 1
Y
Display 0
Rising clock
If Row = 1110
If keya(0) = 0
Display 0
If keya(1) = 0
Display 4
If keya(2) = 0
Display 8
If keya(3) = 0
Display c
If Row = 1101
If keya(0) = 0
Display 1
If keya(1) = 0
Display 5
If keya(2) = 0
Display 9
If keya(3) = 0
Display d
If Row = 1011
If keya(0) = 0
Display 2
If keya(1) = 0
Display 6
If keya(2) = 0
Display A
If keya(3) = 0
Display e
If Row = 0111
If keya(0) = 0
Display 3
If keya(1) = 0
Display 7
If keya(2) = 0
Display b
If keya(3) = 0
Display F
Rotate row l bit right

Synthesis



Simulation



Summary of the simulation results

key	Row	Disp	
1110	1110	11000000	0
1110	1101	11111001	1
1110	1011	10100100	2
1110	0111	10110000	3

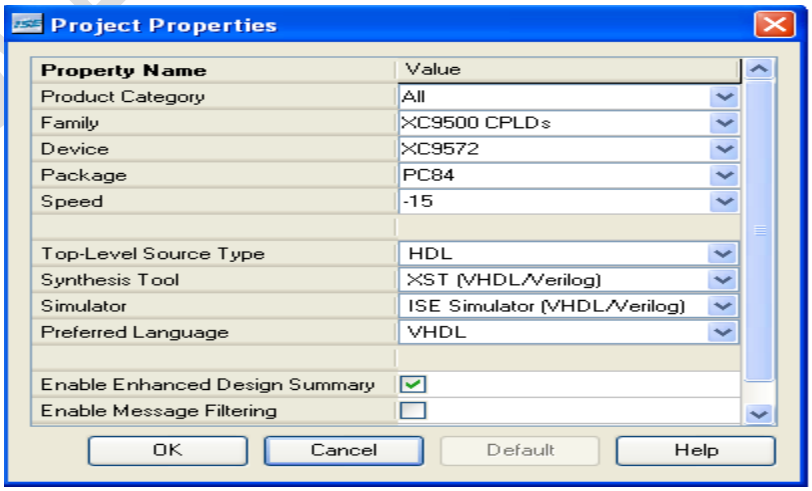
1101	1110	10011001	4
1101	1101	10010010	5
1101	1011	10000010	6
1101	0111	11111000	7
1011	1110	10000000	8
1011	1101	10010000	9
1011	1011	10001000	A
1011	0111	10000011	B
0111	1110	11000110	C
0111	1101	10100001	D
0111	1011	10000110	E
0111	0111	10001110	F

Implementation

----------UCF for FPGA----------

#PACE: Start of PACE I/O Pin Assignments
NET "clk" LOC = "p79" ;
NET "disp<0>" LOC = "p10" ;
NET "disp<1>" LOC = "p11" ;
NET "disp<2>" LOC = "p12" ;
NET "disp<3>" LOC = "p13" ;
NET "disp<4>" LOC = "p15" ;
NET "disp<5>" LOC = "p16" ;
NET "disp<6>" LOC = "p18" ;
NET "disp<7>" LOC = "p19" ;
NET "en<0>" LOC = "p2" ;
NET "en<1>" LOC = "p3" ;
NET "en<2>" LOC = "p7" ;
NET "en<3>" LOC = "p9" ;
NET "key<0>" LOC = "p147" ;
NET "key<1>" LOC = "p146" ;
NET "key<2>" LOC = "p144" ;
NET "key<3>" LOC = "p143" ;
NET "rest" LOC = "p21" ;
NET "row<0>" LOC = "p139" ;
NET "row<1>" LOC = "p140" ;
NET "row<2>" LOC = "p141" ;
NET "row<3>" LOC = "p138" ;
#PACE: Start of PACE Area Constraints

Using Cpld bord CPLD 006



```
NET "clk"  LOC = "p9"  ;
NET "disp<0>"  LOC = "p17"  ;
NET "disp<1>"  LOC = "p18"  ;
NET "disp<2>"  LOC = "p19"  ;
NET "disp<3>"  LOC = "p20"  ;
NET "disp<4>"  LOC = "p21"  ;
NET "disp<5>"  LOC = "p23"  ;
NET "disp<6>"  LOC = "p24"  ;
NET "disp<7>"  LOC = "p25"  ;
NET "en<0>"  LOC = "p12"  ;
NET "en<1>"  LOC = "p13"  ;
NET "en<2>"  LOC = "p14"  ;
NET "en<3>"  LOC = "p15"  ;
NET "key<0>"  LOC = "p67"  ;
NET "key<1>"  LOC = "p66"  ;
NET "key<2>"  LOC = "p65"  ;
NET "key<3>"  LOC = "p63"  ;
NET "rest"  LOC = "p31"  ;
NET "row<0>"  LOC = "p58"  ;
NET "row<1>"  LOC = "p61"  ;
NET "row<2>"  LOC = "p62"  ;
NET "row<3>"  LOC = "p57"  ;
#PACE: Start of PACE Area Constraints
```

12. Write HDL code to generate different waveforms (Sine, Square, Triangle, Ramp etc.,) using DAC (with change in frequency and amplitude)

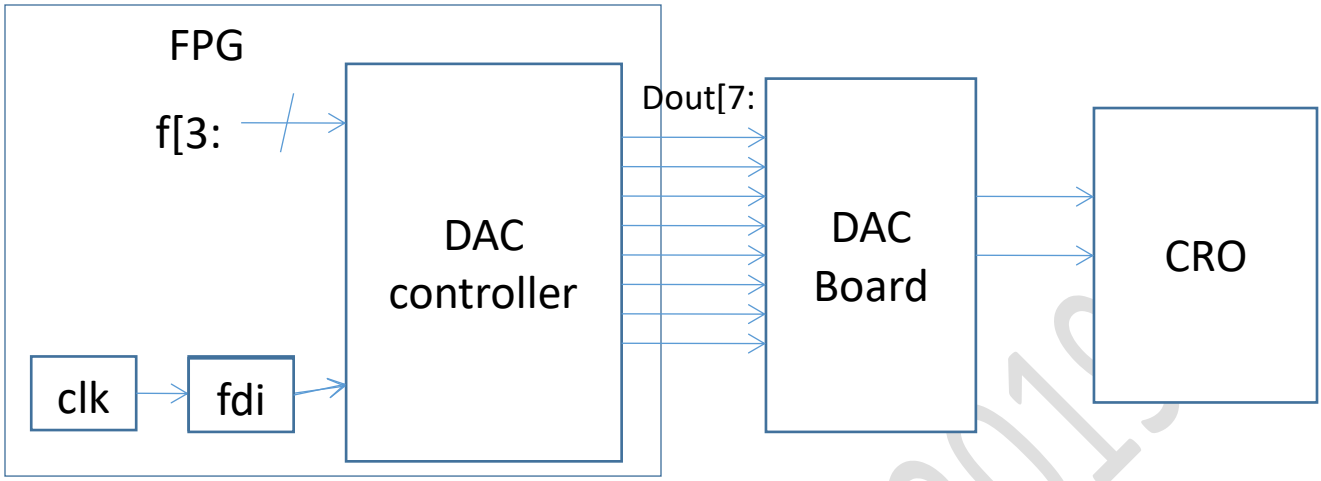
Theory:

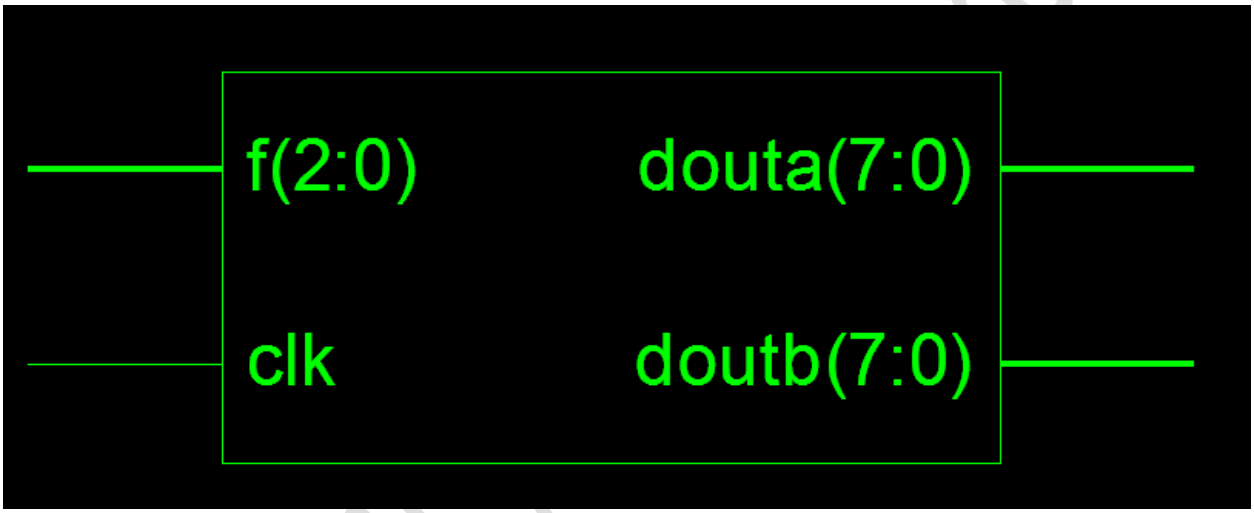
Digital-to-Analog Converter (DAC or D-to-A) is a device that converts a digital (usually binary) code to an analog signal (current, voltage, or electric charge). DAC is needed for the signal to be recognized by human senses or other non-digital systems. A common use of digital-to-analog converters is generation of audio signals from digital information in music players. Digital video signals are converted to analog in televisions and mobile phones to display colors and shades.

Digital-to-analog conversion can degrade a signal, so conversion details are normally chosen so that the errors are negligible. Due to cost and the need for matched components, DACs are almost exclusively manufactured on integrated circuits (ICs). There are many DAC architectures which have different advantages and disadvantages. The suitability of a particular DAC for an application is determined by a variety of measurements including speed and resolution.

A square wave is obtained by generating a high voltage for certain time then low voltage. These low and high voltages are obtained by sending 00 & FF to the input of the DAC. The on times and off times are generated by delay loop.

Block diagram:



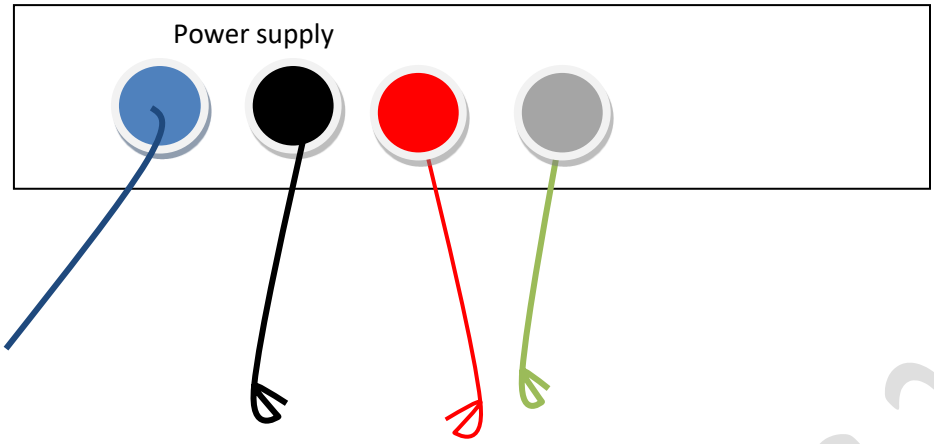


Implementation (UCF):

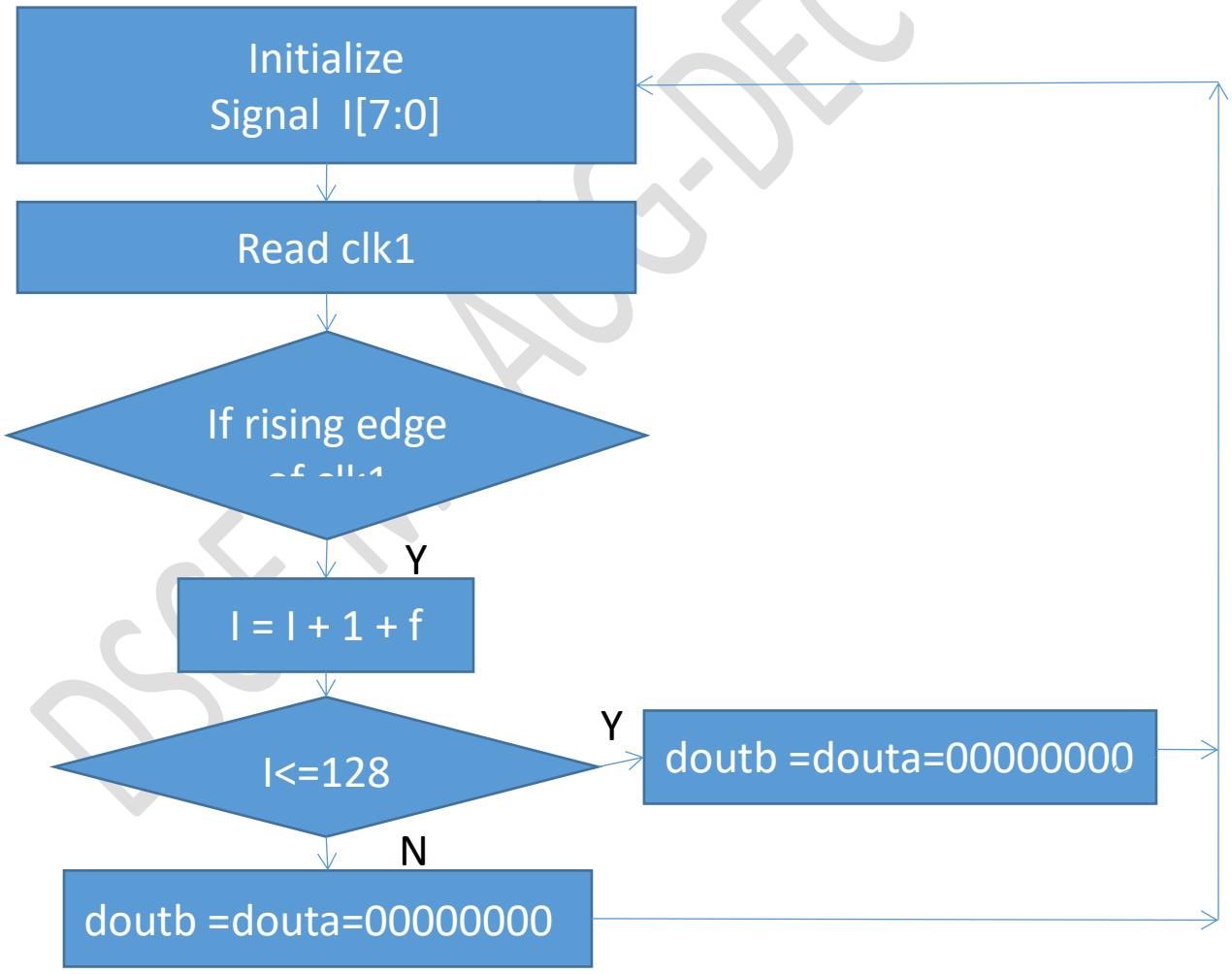
#PACE: Start of Constraints generated by PACE
#PACE: Start of PACE I/O Pin Assignments
NET "clk" LOC = "p79" ;
NET "douta<0>" LOC = "p196" ;
NET "douta<1>" LOC = "p197" ;
NET "douta<2>" LOC = "p191" ;
NET "douta<3>" LOC = "p194" ;
NET "douta<4>" LOC = "p189" ;
NET "douta<5>" LOC = "p190" ;
NET "douta<6>" LOC = "p185" ;
NET "douta<7>" LOC = "p187" ;
NET "doutb<0>" LOC = "p183" ;
NET "doutb<1>" LOC = "p184" ;
NET "doutb<2>" LOC = "p181" ;
NET "doutb<3>" LOC = "p182" ;
NET "doutb<4>" LOC = "p178" ;
NET "doutb<5>" LOC = "p180" ;
NET "doutb<6>" LOC = "p175" ;
NET "doutb<7>" LOC = "p176" ;
NET "f<0>" LOC = "p29" ;
NET "f<1>" LOC = "p27" ;
NET "f<2>" LOC = "p21" ;

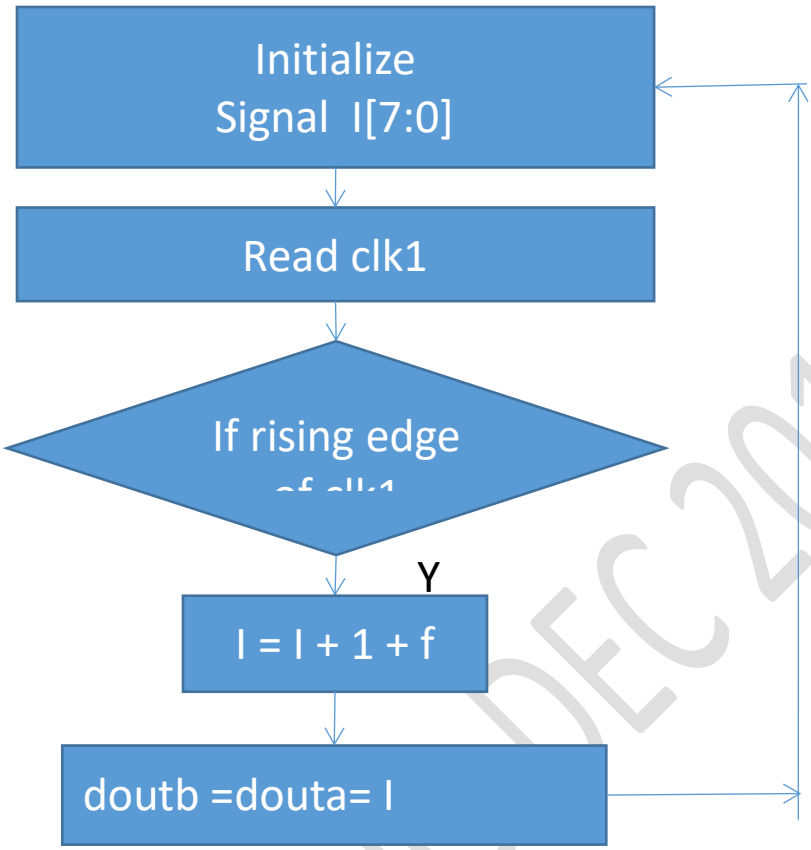
#PACE: Start of PACE Area Constraints

Connection Details:

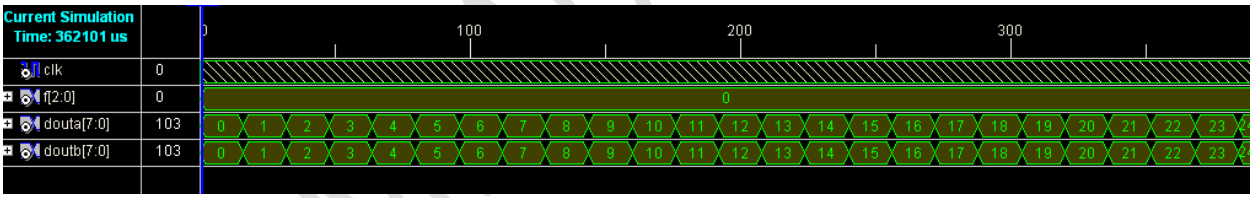


Flow chart: Square wave

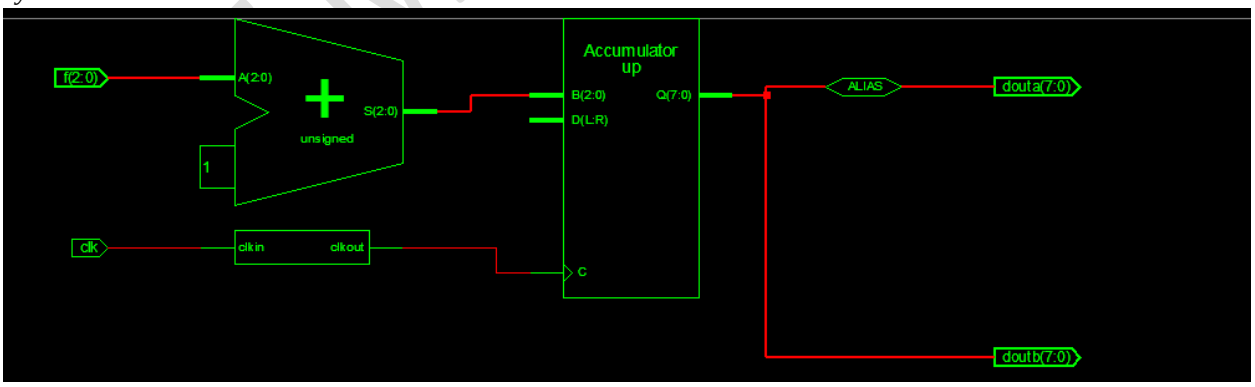




Simulation:



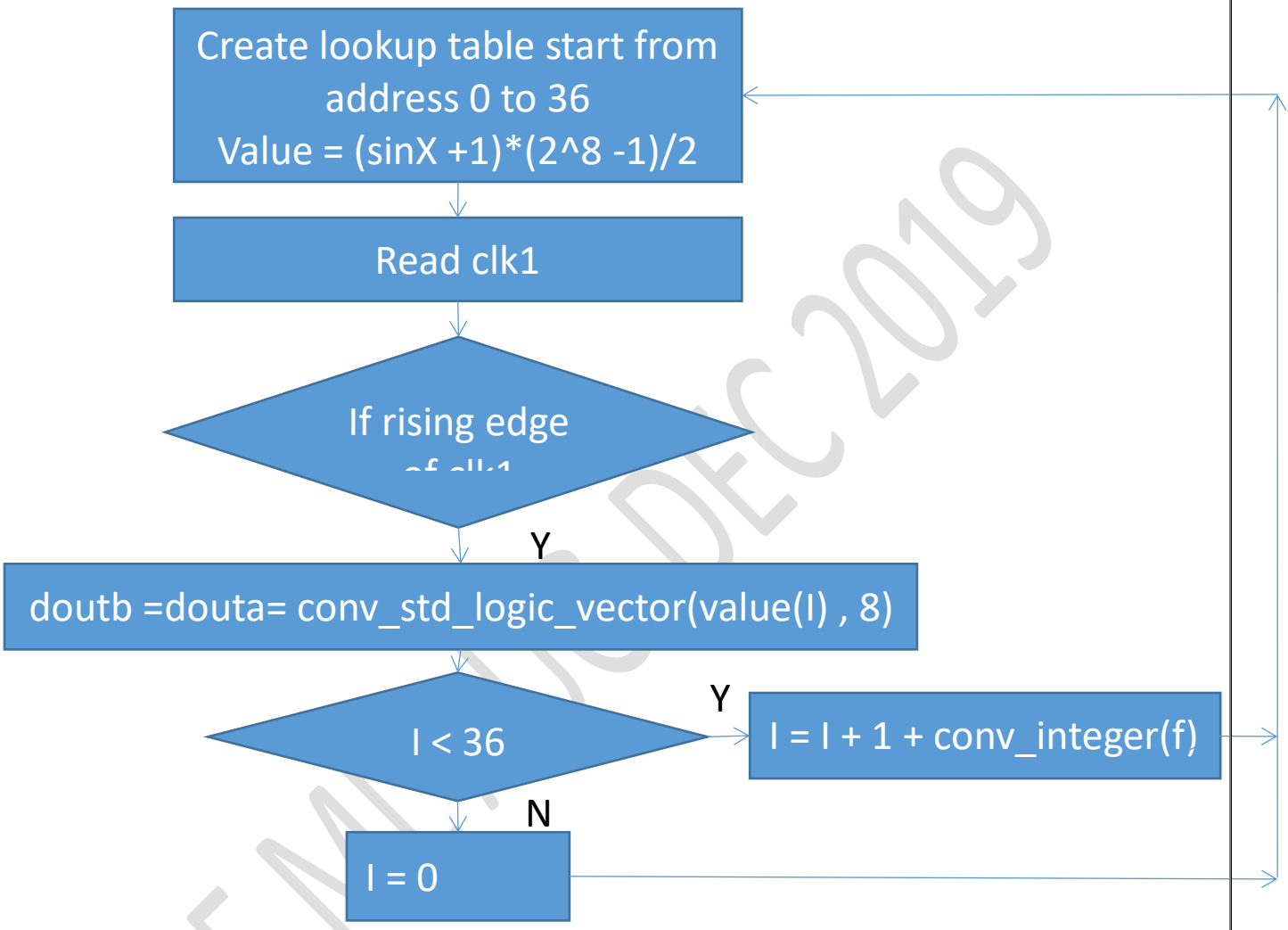
Synthesis:



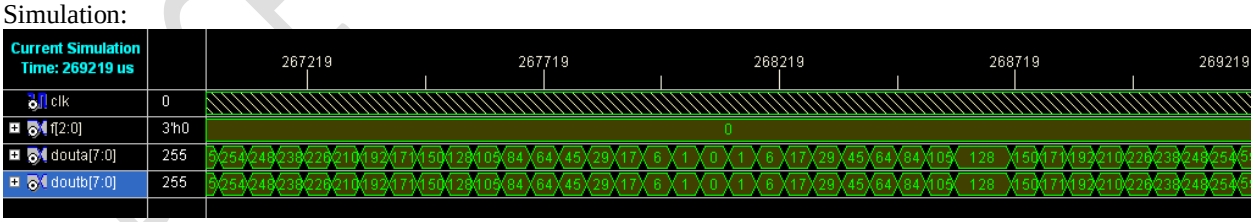
DAC-Sine wave

A sine wave is generated by using a look up table and using equation V = 2.5 + 2.5 sinθ

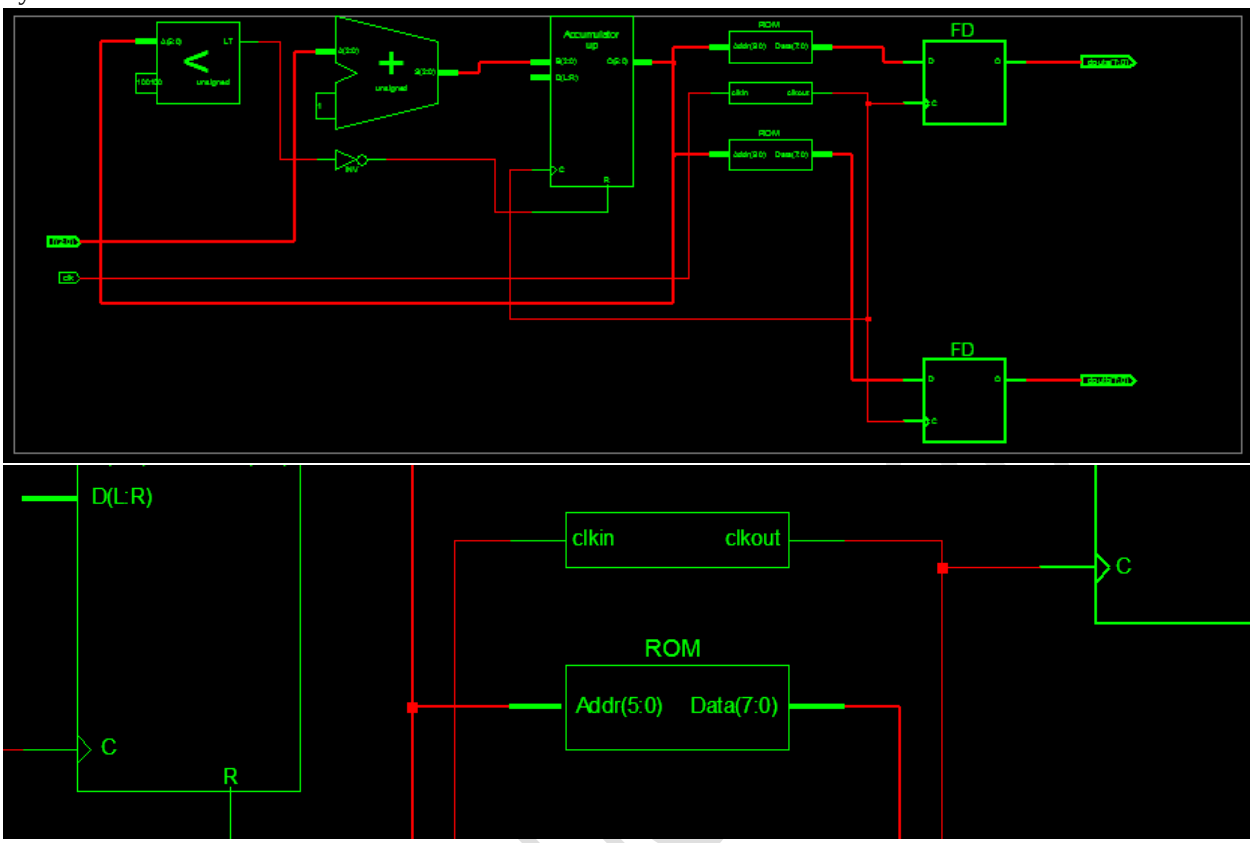
Sine Flow chart:



Simulation:



Synthesis:



PIN DETAILS SPARTAN-3 IC AND CPLD (XC 9572)

	Signal	XC3S50-SPARTAN3 FPGA	CPLD XC9572
	AN1	PIN 2	PIN 12
	AN2	PIN 3	PIN 13
	AN3	PIN 7	PIN 14
	AN4	PIN 9	PIN 15
	SEG1/RS	PIN 10	PIN 17
	SEG2/RW	PIN 11	PIN 18
	SEG3/EI	PIN 12	PIN 19
	SEG4/DT0	PIN 13	PIN 20
	SEG5/DT1	PIN 15	PIN 21
	SEG6/DT2	PIN 16	PIN 23
	SEG7/DT3	PIN 18	PIN 24
	SEG8	PIN 19	PIN 25
	TS1	PIN 21	PIN 31
	TS2	PIN 27	PIN 33
	TS3	PIN 29	PIN 35
	TS4	PIN 35	PIN 37
	TS5	PIN 37	PIN 40
	TS6	PIN 40	PIN 43
	TS7	PIN 43	PIN 45
	TS8	PIN 45	PIN 47
	TS9	PIN 48	PIN 50
	TS10	PIN 52	PIN 52
	TS11	PIN 58	PIN 54
	TS12	PIN 62	PIN 56
	TS13	PIN 64	
	TS14	PIN 67	
	TS15	PIN 71	
	TS16	PIN 74	
	TS17	PIN 80	
	TS18	PIN 162	
	TS19	PIN 86	
	TS20	PIN 90	
	TS21	PIN 94	
	TS22	PIN 100	
	TS23	PIN 102	
	TS24	PIN 107	
	TS25	PIN 113	
	TS26	PIN 115	
	TS27	PIN 117	
	TS28	PIN 120	
	TS29	PIN 123	
	TS30	PIN 125	

	TS31	PIN 131	
	TS32	PIN 133	
	OPLED1	PIN 20	PIN 26
	OPLED2	PIN 26	PIN 32
	OPLED3	PIN 28	PIN 34
	OPLED4	PIN 34	PIN 36
	OPLED5	PIN 36	PIN 39
	OPLED6	PIN 39	PIN 41
	OPLED7	PIN 42	PIN 44
	OPLED8	PIN 44	PIN 46
	OPLED9	PIN 46	PIN 48
	OPLED10	PIN 51	PIN 51
	OPLED11	PIN 57	PIN 53
	OPLED12	PIN 61	PIN 55
	OPLED13	PIN 63	
	OPLED14	PIN 65	
	OPLED15	PIN68	
	OPLED16	PIN 72	
	OPLED17	PIN 78	
	OPLED18	PIN81	
	OPLED19	PIN 85	
	OPLED20	PIN 87	
	OPLED21	PIN 93	
	OPLED22	PIN 95	
	OPLED23	PIN 101	
	OPLED24	PIN 106	
	OPLED25	PIN 111	
	OPLED26	PIN 114	
	OPLED27	PIN 116	
	OPLED28	PIN 119	
	OPLED29	PIN 122	
	OPLED30	PIN 124	
	OPLED31	PIN 130	
	OPLED32	PIN 132	
	10MHZCLK	PIN 79	
	CLK1	PIN 76	PIN 9
	CLK2	PIN 77	
	PB1	PIN 138	PIN 57
	PB2	PIN 139	PIN 58
	PB3	PIN 140	PIN 61
	PB4	PIN 141	PIN 62
	PB5	PIN 143	PIN 63
	PB6	PIN 144	PIN 65

	PB7	**PIN 146**	**PIN 66**
	PB8	**PIN 147**	**PIN 67**
	PSB1	**PIN 165**	
	R	**PIN 148**	
	G	**PIN 149**	
	B	**PIN 150**	
	HS	**PIN 152**	
	VS	**PIN 155**	
	TXD	**PIN 156**	
	RXD	**PIN 161**	
	PS2C	**PIN 166**	
	PS2D	**PIN 167**	
	IO1	**PIN 168**	**PIN 68**
	IO2	**PIN 169**	**PIN 69**
	IO3	**PIN 171**	**PIN 70**
	IO4	**PIN 172**	**PIN 71**
	IO5	**PIN 175**	**PIN 72**
	IO6	**PIN 176**	**PIN 74**
	IO7	**PIN 178**	**PIN 75**
	IO8	**PIN 180**	**PIN 76**
	IO9	**PIN 181**	**PIN 77**
	IO10	**PIN 182**	**PIN 79**
	IO11	**PIN 183**	**PIN 80**
	IO12	**PIN 184**	**PIN 81**
	IO13	**PIN 185**	**PIN 82**
	IO14	**PIN 187**	**PIN 83**
	IO15	**PIN 189**	**PIN 84**
	IO16	**PIN 190**	**PIN 1**
	IO17	**PIN 191**	**PIN 2**
	IO18	**PIN 194**	**PIN 3**
	IO19	**PIN 196**	**PIN 4**
	IO20	**PIN 197**	**PIN 5**
	IO21	**PIN 198**	**PIN 6**
	IO22	**PIN 199**	**PIN 7**
	IO23	**PIN 203**	**PIN 10**
	IO24	**PIN 204**	**PIN 11**

VIVA QUESTIONS:

1. *WHAT IS THE EXPANSION AND IMPORTANCE OF VHDL?*
2. *NAME THE DATA TYPES IN VHDL.*
3. *NAME THE DATA OBJECTS OF VHDL.*
4. *WHAT ARE LOCAL AND GLOBAL OBJECTS?*
5. *HOW TO DENOTE A VECTOR QUANTTITY IN VHDL?*

6. *WHAT IS AN ATTRIBUTE?*
7. *HOW DO YOU REPRESENT POSITIVE AND NEGATIVE CLOCK SIGNAL?*
8. *WHAT ISS AN IDENTIFIER?*
9. *WHAT IS 'SEL'?*
10. *WHAT IS CONCANTENATION OPERATOR?*
11. *WHAT ARE COMPILER DIRECTIVES?*
12. *WHAT IS CASE INEQUALITY?*
13. *WHAT IS REPLICATION OPERATOR?*
14. *HOW DO YOU SPECIFY NUMBERS IN VERILOG?*
15. *NAME THE STYLES OF MODELLING IN VHDL.*
16. *DISTINGUISH BETWEEN CONTINUOUS ASSSIGNMENT & PROCEDURAL ASSIGNMENT STATEMENTS?*
17. *WHAT ARE LOGICAL OPERATORS IN VERILOG?*
18. *NAME THE ASSIGNMENT OPERATORS IN HDL?*
19. *NAME THE OPERATOR '?'*
20. *DEFINE TASK, PROCEDURE AND FUNCTION.*
21. *WHAT ARE THE LOGIC LEVELS APPLICABLE IN HDL?*
22. *WHAT ARE THE CHARACTERS THAT AN IDENTIFIER ACCEPT IN HDL?*
23. *WHAT IS PORTMAP?*
24. *WHAT DOES A TEST BENCH DO?*
25. *WHERE CAN YOU LOCATE A PROCEDURE IN THE CODE?*
26. *WHICH IS THE LOWEST STYLE OF MODELLING? WHY ITS CALLED SO?*
27. *NAME THE PACKAGES UNDER IEEE LIBRARY?*
28. *NAME SOME OF THE SHIFT OPERATORS IN HDL.*
29. *EXPLAIN OPERATIONS OF MUX & DEMUX.*
30. *WHAT IS A GRAY CODE CONVERTER AND FLIP FLOP?*