

# **LAB MANUAL**

## **ECL203 LOGIC CIRCUIT DESIGN LAB**

**Department of Electronics and  
Communication Engineering  
CET**

## **INDEX**

### **PART A**

1. Realization of functions using basic and universal gates (SOP and POS forms)
2. Design and Realization of half /full adder and subtractor using basic gates and universal gates.
3. 4-bit adder/subtractor and BCD adder using 7483.
4. Study of Flip Flops: S-R, D, T, JK and Master Slave JK FF using NAND gates.
5. Asynchronous Counter:3 bit up/down counter and Mod-N counter
6. Synchronous Counter: Realization of 3-bit up/down counter and Mod-N counter.
7. Ring counter and Johnson Counter.
8. Multiplexers and De-multiplexers using gates and ICs. (74150, 74154)

### **PART B**

1. Realization of Logic Gates and Familiarization of FPGAs
2. Adders in Verilog
3. Mux and Demux in Verilog
4. Flipflops and counters
5. Flip-Flops and their Conversion in FPGA
6. BCD to Seven Segment Decoder in FPGA
7. Asynchronous and Synchronous Counters in FPGA

**EXPT NO: 1    REALIZATION OF FUNCTIONS USING BASIC AND UNIVERSAL GATES (SOP AND POS FORMS)**

**DATE:**

**AIM:** Realize the given Boolean function using logic gates in both SOP and POS forms.

Two input SOP -  $A.B + A'.B'$

Two input POS: -  $(A+B) (B+C) (A+C')$

**COMPONENTS AND EQUIPMENTS REQUIRED:**

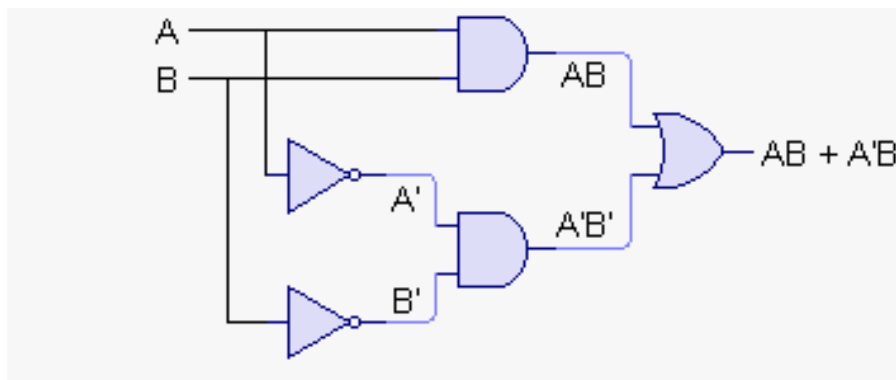
Sl No	Components and equipments	Specification	Quantity
1	IC	7400,7402,7404,7408,7411,7410,7427 7432,7486	Depending upon the function
2	IC trainer kit		1

**THEORY:**

SOP: - It is the Sum of product form in which the terms are taken as 1. It is denoted in the K-map expression by sigma ( $\Sigma$ )

$A.B. + A'B'$

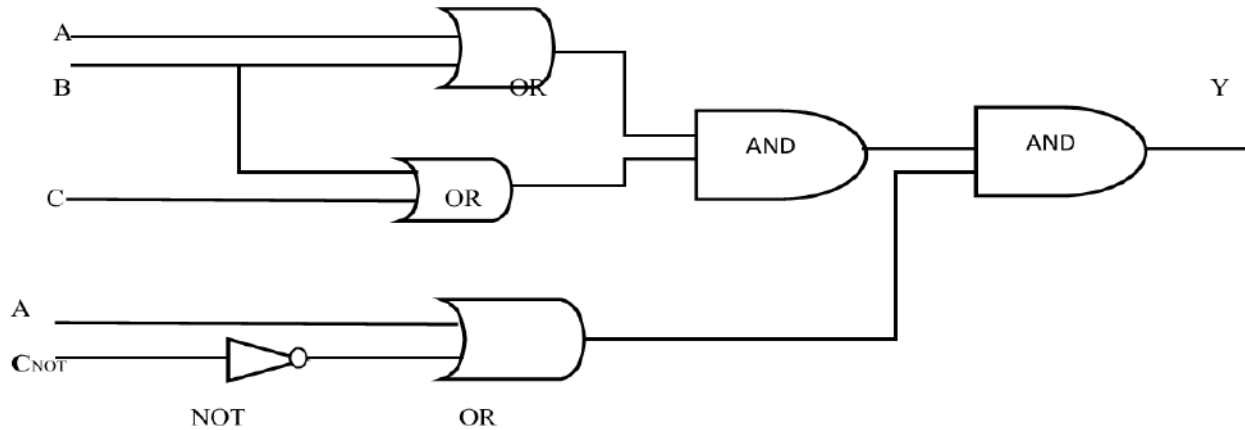
Logic Circuit Of this expression:-



POS: - It is the product of the sums form in which the terms are taken as 0. It is denoted in the K-Map expression by the Sign pie ( $\pi$ )

$$(A+B) (B+ C) (A + C')$$

Logic Circuit Of this expression:-



### PROCEDURE:

1. Realize the functions using variables A, B, and C.
2. Write the truth table of the corresponding functions.
3. Write the expression and solve it using K maps.

C \ AB	AB			
	00	01	11	10
0				
1				

4. Write expression of  $Y \rightarrow f(A,B,C)$ , apply De-Morgans law to simply the equation.
5. Realize using suitable gates.
6. Verify the output using different inputs.
7. Set up the circuits and observe the outputs. Enter the output states in truth table corresponding to the input combination.

**RESULT:** Realized the Boolean functions using logic gates and verified the outputs for different inputs.

**EXPT NO: 2   DESIGN AND REALIZATION OF HALF/FULL ADDER AND SUBTRACTOR USING BASIC GATES AND UNIVERSAL GATES.**

**DATE:**

**AIM:** To design and setup Half/Full Adder and Subtractor using basic gates and universal gates.

**COMPONENTS AND EQUIPMENTS REQUIRED:**

SL NO:	COMPONENTS AND EQUIPMENTS	SPECIFICATION	QUANTITY
1.	IC Trainer kit		1
2.	IC	7408	1
		7432	1
		7486	1
		7400	2
		7404	1

**THEORY:**

**Half Adder:** The simplest binary adder is called a half adder. Half adder has two input bits and two output bits. One output bit is the sum and the other is carry. They are represented by S and C in the logic symbol.

$$S = A \oplus B$$

$$C = AB$$

**Full Adder:** A half adder has no provision to add a carry from the lower order bits when binary numbers are added. When two input bits and a carry are to be added, the number of input bits becomes three and the input combinations increases to eight. For this, a full adder is used. Like

half adder, it also has a sum bit and a carry bit. The new carry generated is represented by  $C_n$

and carry generated from the previous addition is represented by  $C_{n-1}$ .

$$S = A \oplus B \oplus C_{n-1}$$

$$C_n = AB + C_{n-1} \quad (A \oplus B)$$

**Half Subtractor:** Subtracting a single-bit binary value B from another A (i.e. A - B ) produces a difference bit D and a borrow out bit B-out. This operation is called half subtraction and the circuit to realize it is called a half subtractor. The Boolean functions describing the half-Subtractor are:

$$D = A \oplus B$$

$$Br = \bar{A} B$$

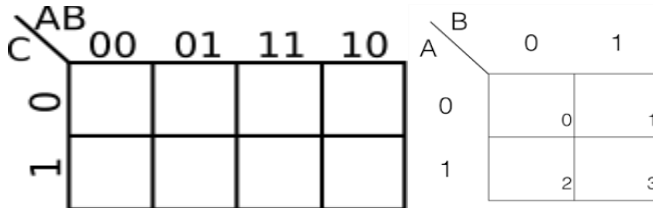
**Full Subtractor:** Subtracting two single-bit binary values, B, Cin from a single-bit value A produces a difference bit D and a borrow out Br bit. This is called full subtraction. The Boolean functions describing the full-subtractor are:

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

$$Br = \bar{A} B + B Cin + \bar{A} Cin$$

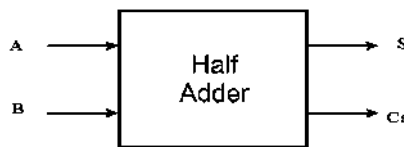
### PROCEDURE:

1. Realize the functions using input variables .
2. Write the truth table of the corresponding functions.
3. Write the expression and solve it using K maps.

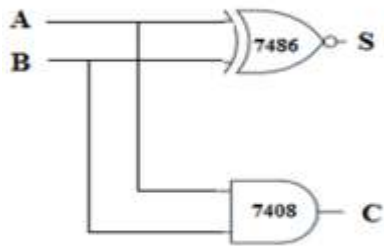


4. Write expression of  $Y \rightarrow f(A, B, C)$ , apply De-Morgans law to simplify the equation for implementing with universal gates..
5. Realize using suitable gates.
6. Verify the output using different inputs.
7. Set up the circuits and observe the outputs. Enter the output states in truth table corresponding to the input combination.

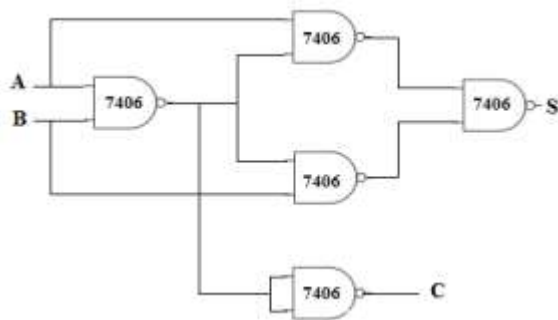
### HALF ADDER



a) Using basic gates



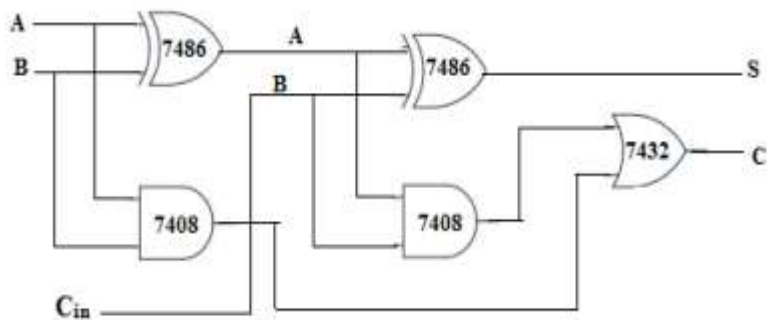
b) Using NAND gates



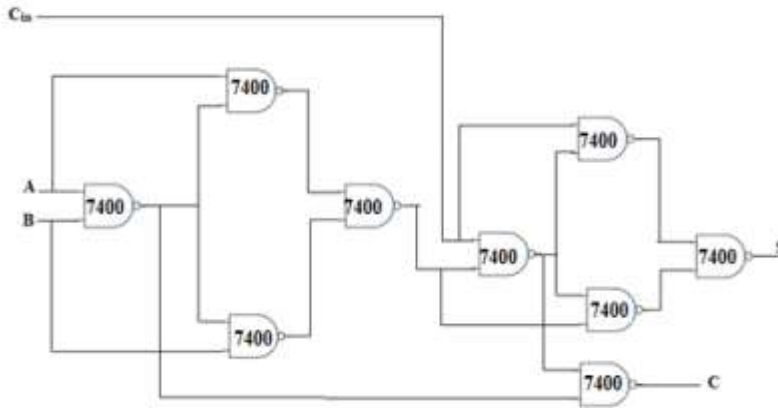
## FULL ADDER



a) Using basic gates

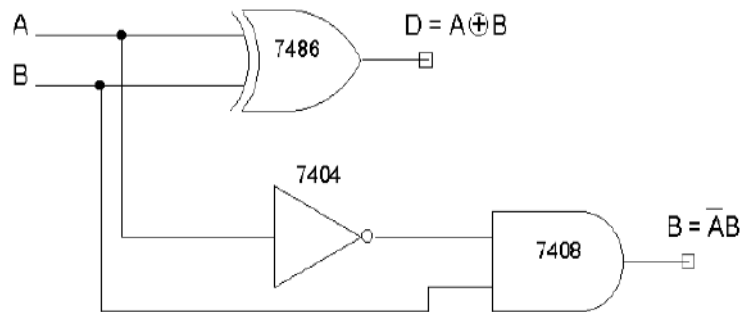


b) Using NAND gates

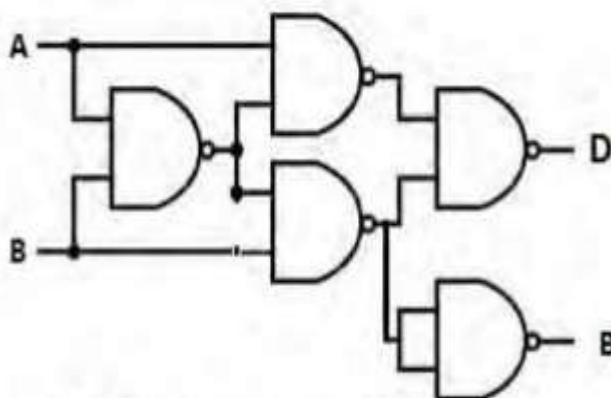


### HALF SUBTRACTOR

a) Using basic gates



b) Using NAND gates



### FULL SUBTRACTOR

a) Using basic gates





Realized half/full adder and subtractor using basic and universal gates and verified the outputs

**EXPT NO. 3    4 BIT ADDER/SUBTRACTOR AND BCD ADDER USING 7483**

**DATE :**

**AIM:**

To design and implement 4-bit adder / subtractor and BCD adder using IC 7483.

**COMPONENTS AND EQUIPMENTS REQUIRED:**

Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	IC	IC 7483	1
2.	EX-OR GATE	IC 7486	1
3.	NOT GATE	IC 7404	1
4.	IC TRAINER KIT	-	1

**THEORY:**

**4 BIT BINARY ADDER:**

A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain. The augend bits of 'A' and the addend bits of 'B' are designated by subscript numbers from right to left, with subscript 0 denoting the least significant bits. The carry's are connected in chains through the full adder. The input carry to the adder is  $C_0$  and it ripples through the full adder to the output carry  $C_4$ .

**4 BIT BINARY SUBTRACTOR:**

The circuit for subtracting A-B consists of an adder with inverters, placed between each data input 'B' and the corresponding input of full adder. The input carry  $C_0$  must be equal to 1 when performing subtraction.

**4 BIT BINARY ADDER/SUBTRACTOR:**

The addition and subtraction operation can be combined into one circuit with one common binary adder. The mode input M controls the operation. When  $M=0$ , the circuit is an adder circuit. When  $M=1$ , it becomes a subtractor.

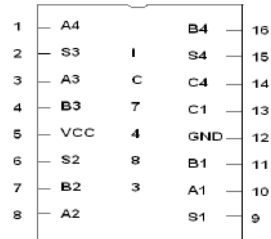
**4 BIT BCD ADDER:**

Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be

greater than 19, the 1 in the sum being an input carry. The output of two decimal digits must be represented in BCD and should appear in the form listed in the columns.

ABCD adder that adds 2 BCD digits and produces a sum digit in BCD. The 2 decimal digits, together with the input carry, are first added in the top 4-bit adder to produce the binary sum.

#### PIN DIAGRAM FOR IC 7483:

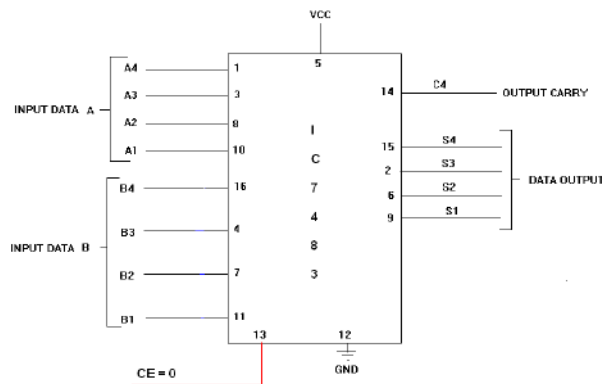


#### PROCEDURE

1. Realize adder/ subtractor using suitable gates and IC 7483.
2. Realize the BCD adder using IC7483,and K map for converting binary to BCD.
3. Verify the output using different input combinations.
4. Set up the circuits and observe the outputs.

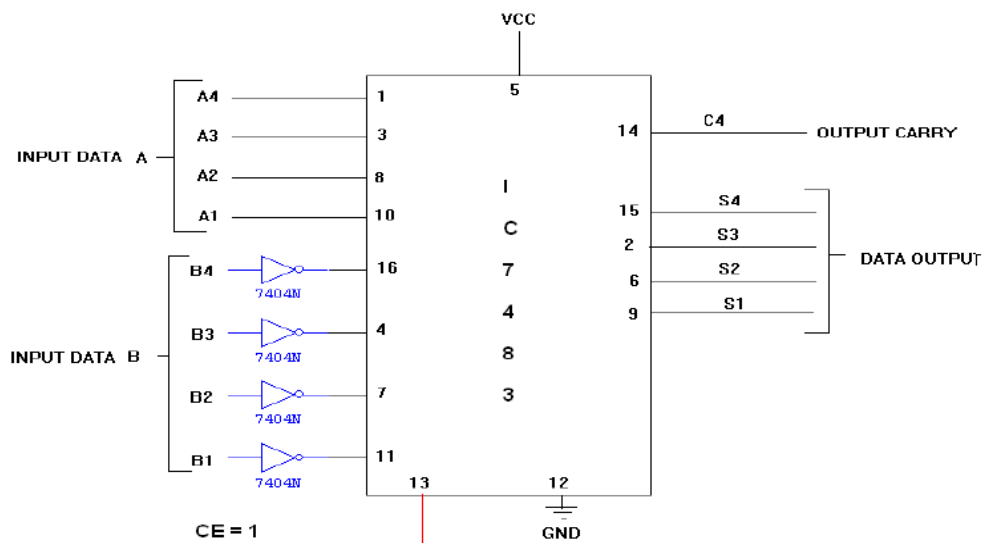
#### LOGIC DIAGRAM:

##### 4-BIT BINARY ADDER



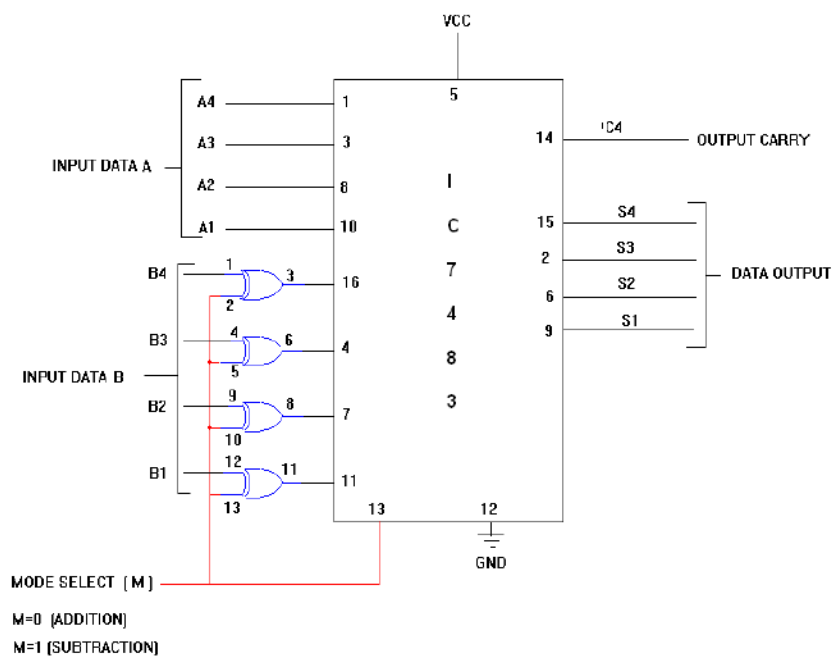
#### LOGIC DIAGRAM:

##### 4-BIT BINARY SUBTRACTOR



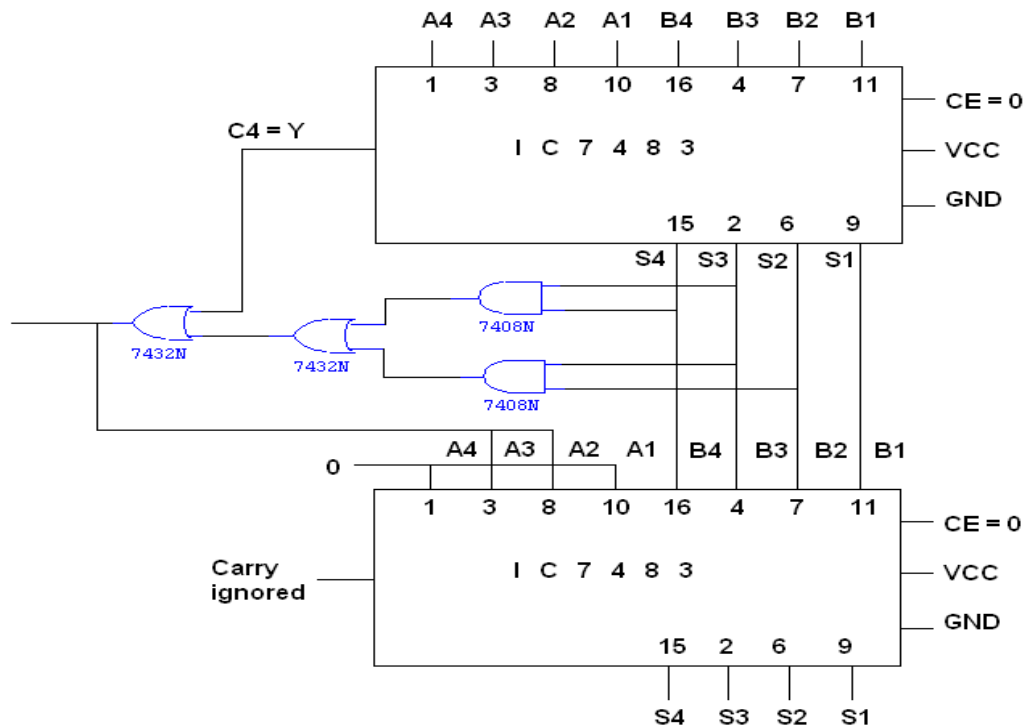
**LOGIC DIAGRAM:**

**4-BIT BINARY ADDER/SUBTRACTOR**



**LOGIC DIAGRAM:**

**BCD ADDER**



**RESULT:** Realized the circuit for 4-bit adder/subtractor and BCD adder using IC 7483 and observed the outputs for different inputs.

**EXPT NO:4     STUDY OF FLIP FLOPS: S-R, D, T, JK AND MASTER SLAVE JK FF**  
**USING NAND GATES**

**DATE :**

**AIM:** To implement the following flip flops using logic gates.

1. SR flip flop
2. JK flip flop
3. D flip flop
4. T flip flop
5. JK Master slave flip flop

**COMPONENTS AND EQUIPMENTS REQUIRED:**

Sl No	Components and equipments	Specification	Quantity
1	IC	7400,7404,7410	1,1,2
2	IC trainer kit		1

## THEORY

**SR FLIP-FLOP:** There are two inputs to the flip-flop defined as R and S. When I/Ps  $R=0$  and  $S=0$  then O/P remains unchanged. When I/Ps  $R=0$  and  $S=1$  the flip-flop switches to the stable state where O/P is 1 i.e. SET. The I/P condition is  $R=1$  and  $S=0$  the flip-flop is switched to the stable state where O/P is 0 i.e. RESET. The I/P condition is  $R=1$  and  $S=1$  the flip-flop is switched to the stable state where O/P is forbidden.

**JK FLIP-FLOP:** For the purpose of counting, the J K flip-flop is the ideal element to use. The variables J and K are called control I/Ps because they determine what the flip-flop does when a positive edge arrives. When J and K are both 0s, both AND gates are disabled and Q retains its last value.

**D FLIP FLOP:** This kind of flip flop prevents the value of D from reaching the Q output until clock pulses occur. When the clock is low, both AND gates are disabled D can change value without affecting the value of Q. On the other hand, when the clock is high, both AND gates are enabled. In this case, Q is forced to equal the value of D. When the clock again goes low, Q retains or stores the last value of D. D flip flop is a bistable circuit whose D input is transferred to the output after a clock pulse is received.

**T FLIP-FLOP:** The T or "toggle" flip-flop changes its output on each clock edge, giving an output which is half the frequency of the signal to the T input. It is useful for constructing binary counters, frequency dividers, and general binary addition devices. It can be made from a J-K flip-flop by tying both of its inputs high.

**MASTER-SLAVE JK FLIP-FLOP:** The Master-Slave Flip-Flop is basically two gated SR flip-flops connected together in a series configuration with the slave having an inverted clock pulse. The outputs from Q and  $\bar{Q}$  from the "Slave" flip-flop are fed back to the inputs of the "Master" with the outputs of the "Master" flip flop being connected to the two inputs of the "Slave" flip flop. This feedback configuration from the slave's output to the master's input gives the characteristic toggle of the JK flip flop.

The input signals J and K are connected to the gated "master" SR flip flop which "locks" the input condition while the clock (Clk) input is "HIGH" at logic level "1". As the clock input of the "slave" flip flop is the inverse (complement) of the "master" clock input, the "slave" SR flip flop does not toggle. The outputs from the "master" flip flop are only "seen" by the gated "slave" flip flop when the clock input goes "LOW" to logic level "0".

When the clock is “LOW”, the outputs from the “master” flip flop are latched and any additional changes to its inputs are ignored. The gated “slave” flip flop now responds to the state of its inputs passed over by the “master” section.

Then on the “Low-to-High” transition of the clock pulse the inputs of the “master” flip flop are fed through to the gated inputs of the “slave” flip flop and on the “High-to-Low” transition the same inputs are reflected on the output of the “slave” making this type of flip flop edge or pulse-triggered. Then, the circuit accepts input data when the clock signal is “HIGH”, and passes the data to the output on the falling-edge of the clock signal. In other words, the Master-Slave JK Flip flop is a “Synchronous” device as it only passes data with the timing of the clock signal.

### **PROCEDURE:**

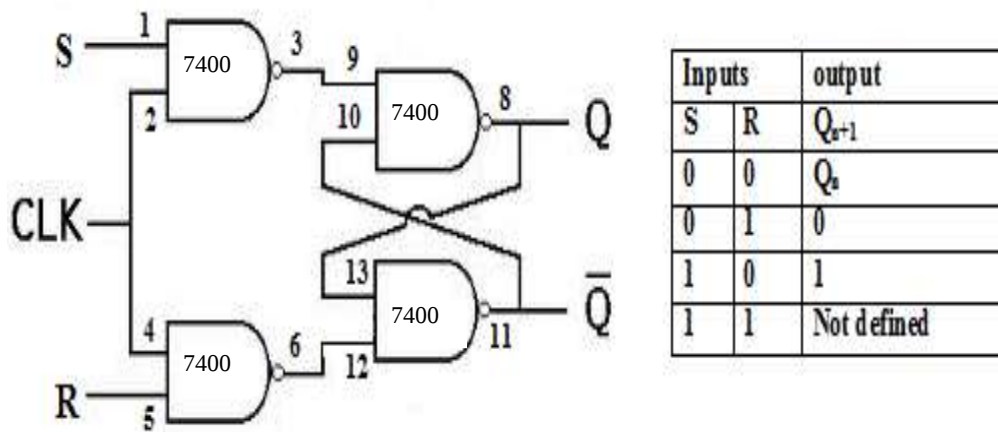
1. Realize the functions of flipflop using suitable variables..
2. Write the truth table of the corresponding functions.
3. Write the expressions and solve it using K maps.

		B	
		0	1
A	0	0	1
	1	2	3

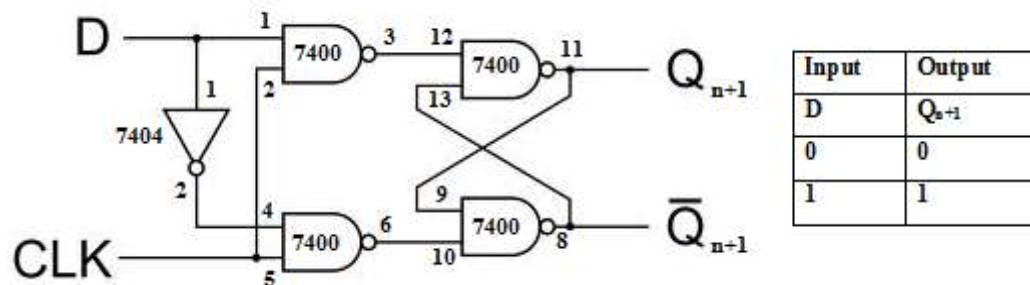
4. Write an expression of  $Q_{n+1}$ .
5. Realize using suitable gates and ICs.
6. Verify the output using different inputs.
7. Set up the circuits and observe the outputs. Enter the output states in truth table corresponding to the input combination.

### **SR FLIP FLOP**

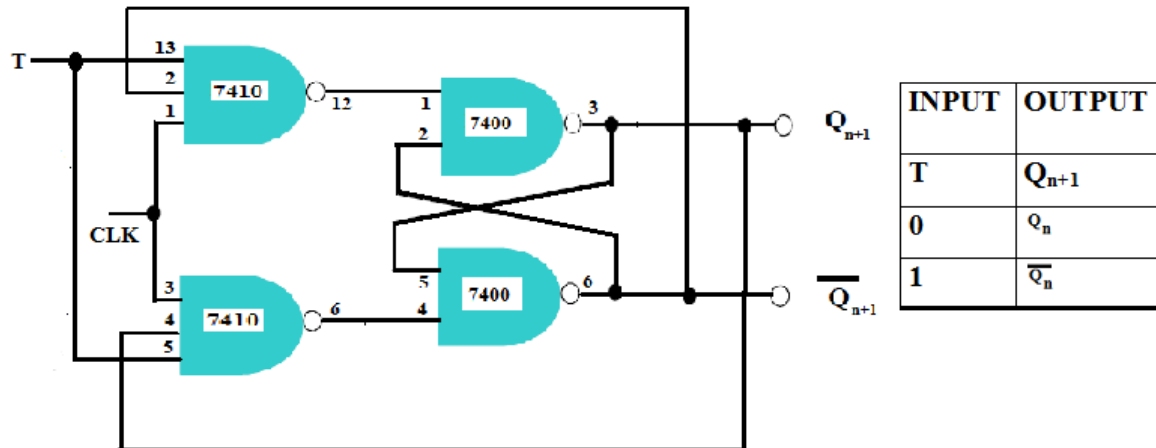




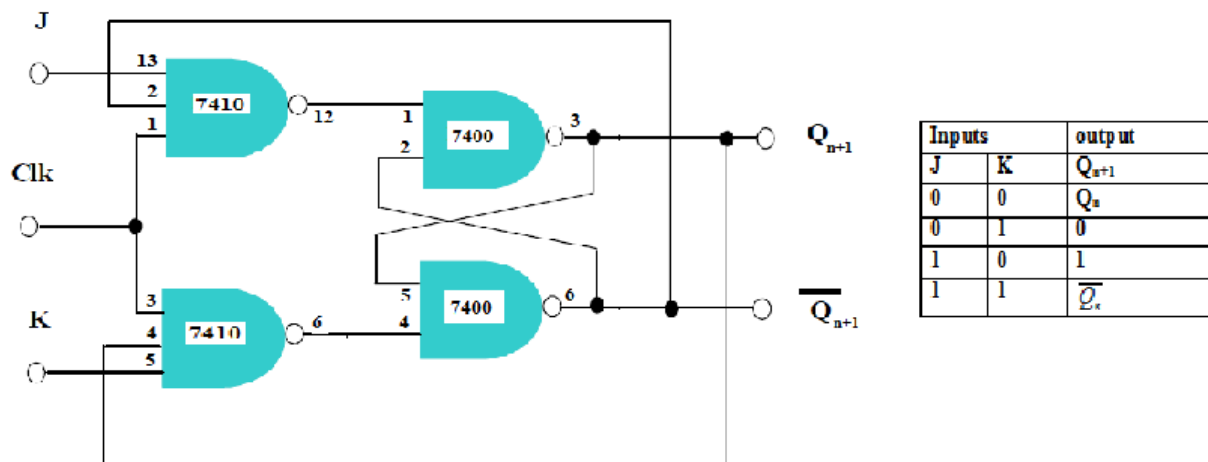
### D FLIP FLOP



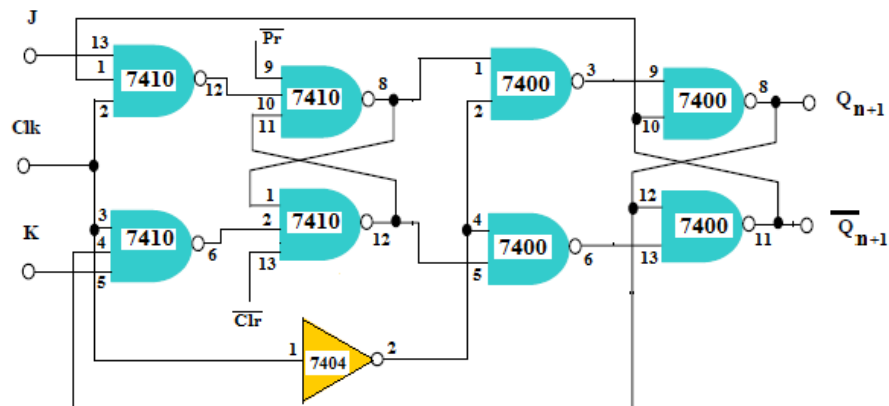
### T FLIP FLOP



### JK FLIP FLOP



### MASTER SLAVE JK FLIP FLOP



INPUT		OUTPUT
J	K	$Q_{n+1}$
0	0	$Q_n$
0	1	0
1	0	1
1	1	$\overline{Q_n}$

**RESULT:** Realized different flipflops using logic gates and verified its outputs for different inputs.

**DATE:**

**AIM:** To realize an asynchronous 3 bit up/down counter and decade counter.

**COMPONENTS AND EQUIPMENTS REQUIRED:**

Sl No	Components and equipments	Specification	Quantity
1	IC	7476,7400	2,2 each
2	IC trainer kit		1

**THEORY:**

**Asynchronous Decade Counter**

This type of asynchronous counter counts upwards on each trailing edge of the input clock signal starting from 0000 until it reaches an output 1001 (decimal 9). Both outputs Q0 and Q3 are now equal to logic “1”. On the application of the next clock pulse, the output from the 74LS10 NAND gate changes state from logic “1” to a logic “0” level. As the output of the NAND gate is connected to the CLEAR (CLR) inputs of all the 74LS73 J-K Flip-flops, this signal causes all of the Q outputs to be reset back to binary 0000 on the count of 10. As outputs QA and QD are now both equal to logic “0” as the flip-flop’s have just been reset, the output of the NAND gate returns back to a logic level “1” and the counter restarts again from 0000. We now have a decade or Modulo-10 up-counter.

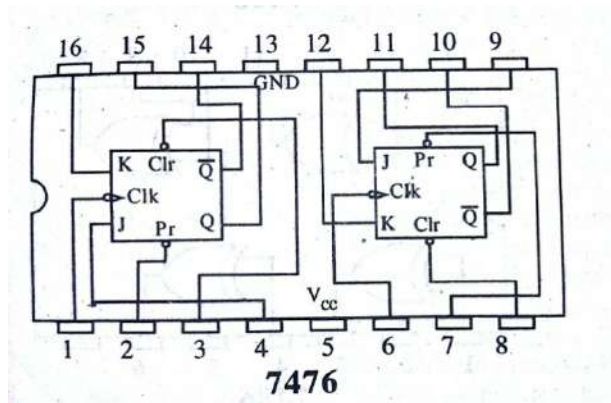
**3- bit Up-Down Counter**

As we know that in the up-counter each flip-flop is triggered by the normal output of the preceding flip-flop (from output Q of first flip-flop to clock of next flip-flop); whereas in a down-counter, each flip-flop is triggered by the complement output of the preceding flip-flop (from output  $Q^{\wedge}$  of first flip-flop to clock of next flip-flop). The operation of such a counter is controlled by the up-down control input. i.e, in the circuit diagram mode input(M) is the up-down control input. When M=0, the output of XOR gate is Q output and it will act as a 3-bit up counter. When M=1, the output of the XOR gate is a complement of Q output and it will act as a 3-bit down counter.

**PROCEDURE:**

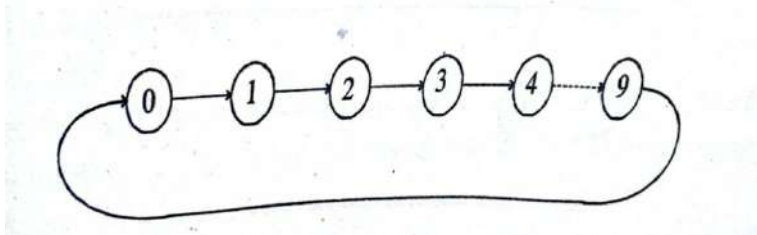
1. Realize the functions using suitable variables.
2. Write the truth table of the corresponding functions.
3. Setup the decade counter circuit; reset the outputs using clear input. Apply clock pulses and observe the counting from 0000 to 1001.
4. Setup 3 bit up/down counter. Clear all FF. Apply logic 0 to the mode control pin and observe the up counting.
5. Preset all FF and apply logic 1 to mode control and observe down counting.
6. Verify the output using different inputs.
7. Set up the circuits and observe the outputs. Enter the output states in the Truth table corresponding to the input combination.

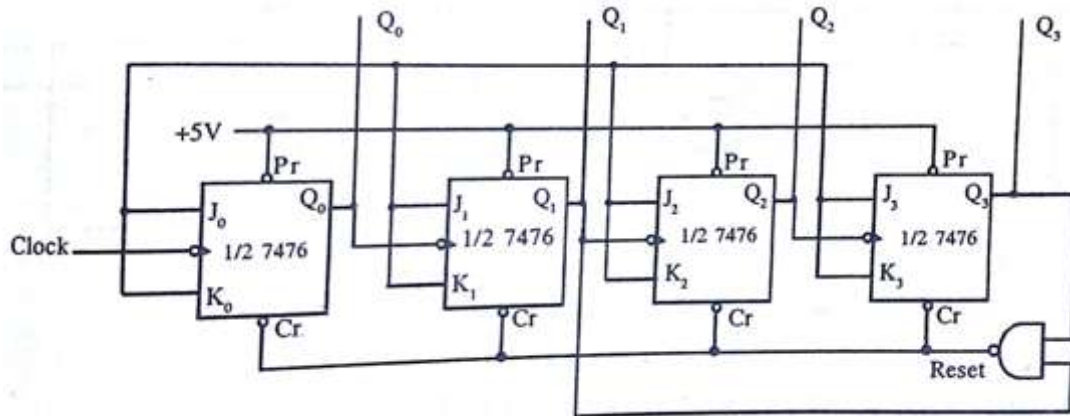
**Pin diagram of 7476**



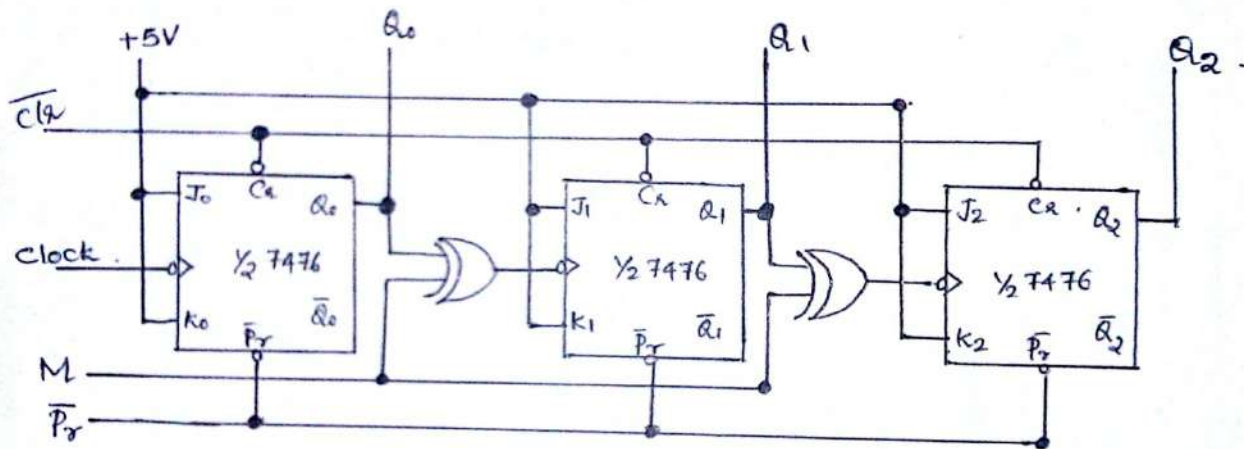
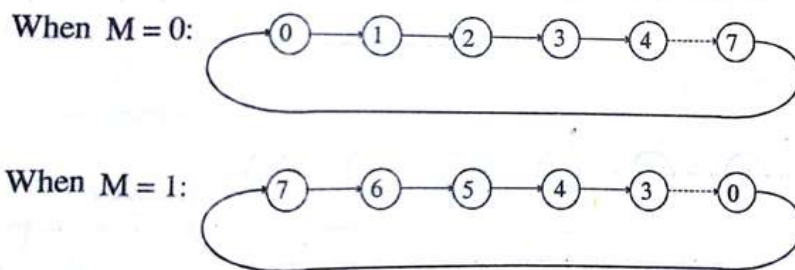
**CIRCUIT DIAGRAMS**

**DECADE COUNTER**





### 3 BIT UP/DOWN COUNTER



**RESULT:** Realized asynchronous 3 bit up/down counter and decade counter and verified the output.

**EXPT NO: 6****SYNCHRONOUS COUNTERS****DATE:**

**AIM:** To design and setup synchronous 3 bit up/down counter and decade counter using JK flipflops.

**COMPONENTS AND EQUIPMENTS REQUIRED:**

Sl No	Components and equipments	Specification	Quantity
1	IC	7476,7400,7408,7410	2,2,1,1 each
2	IC trainer kit		1

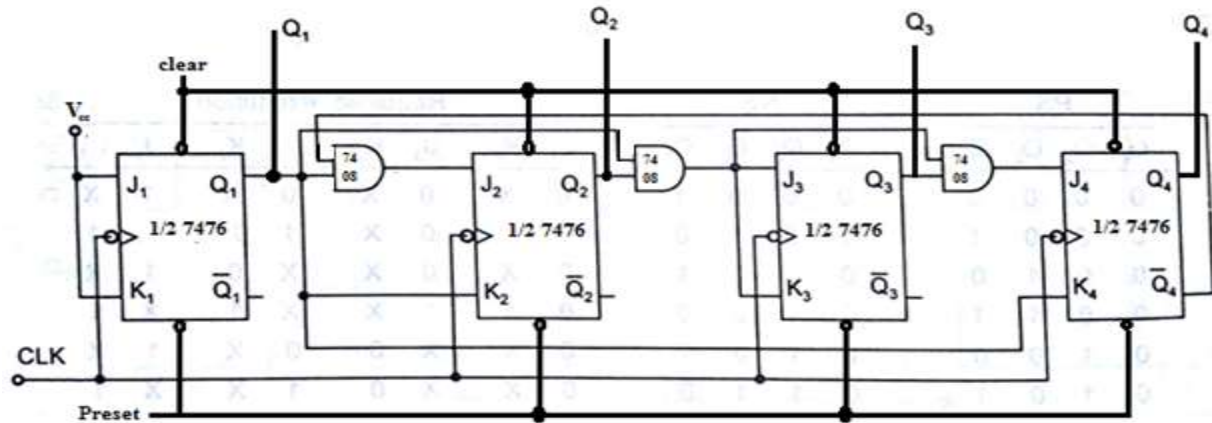
**THEORY:**

In Asynchronous binary counters the output of one counter stage is connected directly to the clock input of the next counter stage and so on along the chain. The result of this is that the Asynchronous counter suffers from what is known as “Propagation Delay” in which the timing signal is delayed a fraction through each flip-flop. However, with the Synchronous Counter, the external clock signal is connected to the clock input of EVERY individual flip-flop within the counter so that all of the flip-flops are clocked together simultaneously (in parallel) at the same time giving a fixed time relationship. In other words, changes in the output occur in “synchronization” with the clock signal. The result of this synchronization is that all the individual output bits change state at exactly the same time in response to the common clock signal with no ripple effect and therefore, no propagation delay.

**PROCEDURE:**

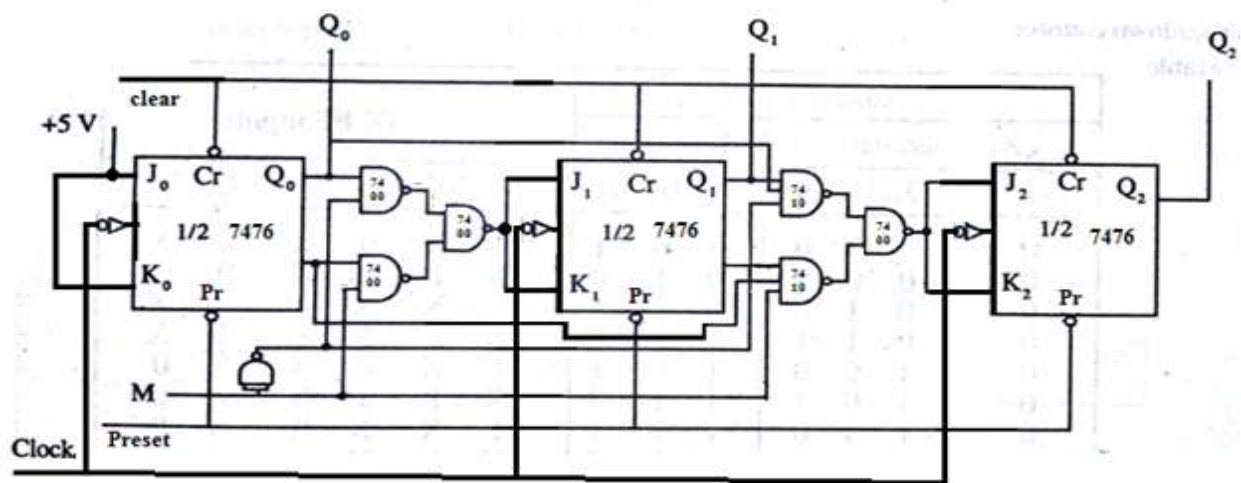
1. Decide the number and type of FF.
2. Realize the functions using suitable variables.
3. Write the excitation table of flipflop.
4. Decide the Mode to control the input
5. Draw the state transition diagram and circuit excitation table.
6. Set up the circuit and observe the outputs.

**CIRCUIT DIAGRAM****DECADE COUNTER**



### 3 BIT UP/DOWN COUNTER

#### Circuit Diagram



**RESULT :** Realized 3 bit up/down counter and decade counter using JK flipflops and verified the outputs for different inputs.



**EXPT NO: 7      RING COUNTER AND JOHNSON COUNTER****DATE:****AIM:** To implement ring and Johnson counter using flip flop integrated circuits.**COMPONENTS AND EQUIPMENTS REQUIRED:**

Sl No	Components and equipments	Specification	Quantity
1	IC	7474	2
2	IC trainer kit		1

**THEORY:****4-bit Ring Counter**

The synchronous **Ring Counter**, is preset so that exactly one data bit in the register is set to logic “1” with all the other bits reset to “0”. To achieve this, a “CLEAR” signal is firstly applied to all the flip-flops together in order to “RESET” their outputs to a logic “0” level and then a “PRESET” pulse is applied to the input of the first flip-flop before the clock pulses are applied. This then places a single logic “1” value into the circuit of the ring counter.

So on each successive clock pulse, the counter circulates the same data bit between the four flip-flops over and over again around the “ring” every fourth clock cycle. But in order to cycle the data correctly around the counter we must first “load” the counter with a suitable data pattern as all logic “0’s” or all logic “1’s” outputted at each clock cycle would make the ring counter invalid.

**Johnson Counter**

The **Johnson Ring Counter** or “Twisted Ring Counters”, is another shift register with feedback exactly the same as the standard *Ring Counter* , except that this time the inverted output Q of the last flip-flop is now connected back to the input D of the first flip-flop as shown in circuit diagram.

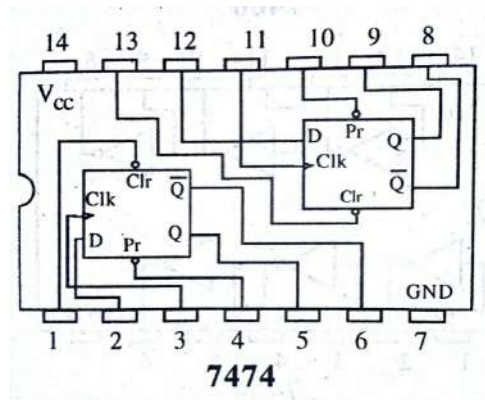
The main advantage of this type of ring counter is that it only needs half the number of flip-flops compared to the standard ring counter then its modulo number is halved. So a “n-stage” Johnson

counter will circulate a single data bit giving sequence of  $2^n$  different states and can therefore be considered as a “mod- $2^n$  counter”.

**PROCEDURE:**

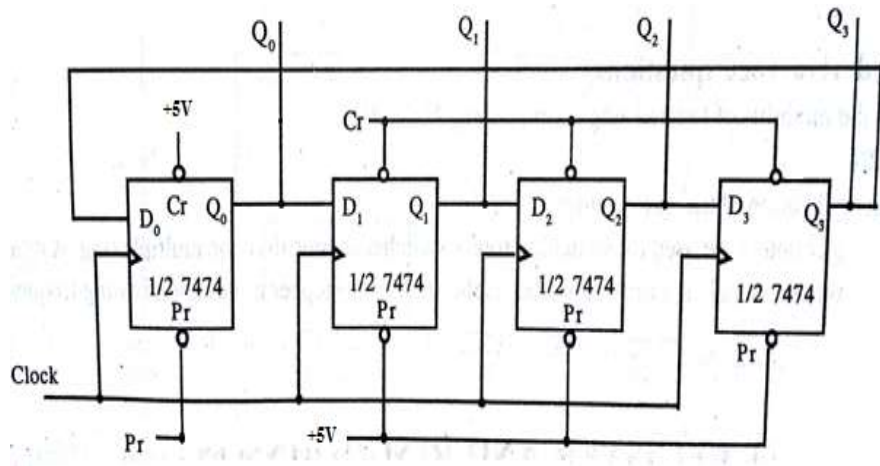
1. Decide the type of FF.
2. Test all the components and IC packages using a digital IC tester.
3. Set up the circuits to verify the counter states.

**Pin Diagram of 7474**



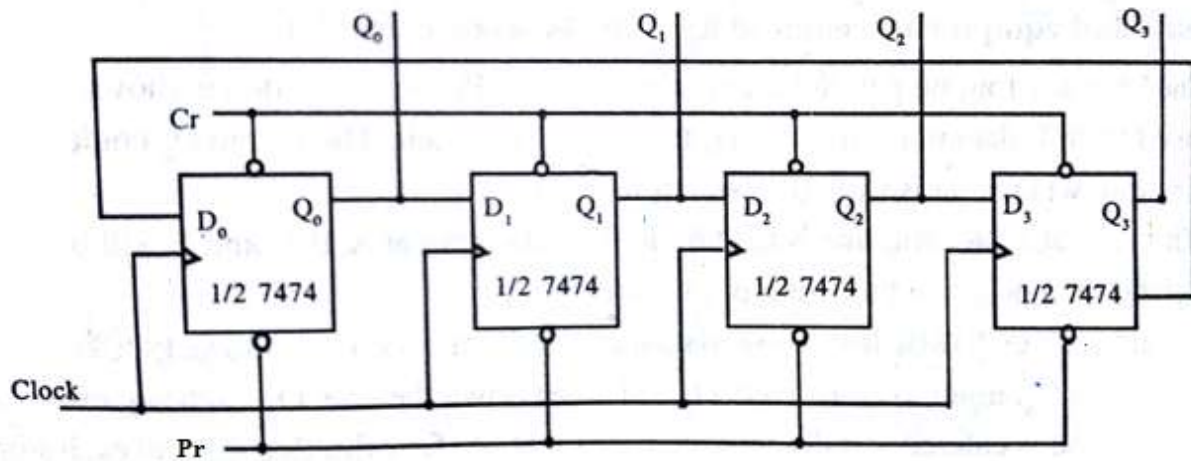
Ring Counter

**Circuit Diagram**



**Johnson Counter**

**Circuit Diagram**



**RESULT:** Realized Ring and Johnson counter using flip flop and observed its outputs for different inputs.

**EXPT NO: 8      MULTIPLEXERS AND DE-MULTIPLEXERS USING GATES**  
**AND ICs. (74150, 74154)**

**DATE :**

**AIM:** To design and implement multiplexer and demultiplexer using logic gates and study of IC 74150 and IC 74154.

**COMPONENTS AND EQUIPMENTS REQUIRED:**

Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	3 I/P AND GATE	IC 7411	2
2.	OR GATE	IC 7432	1
3.	NOT GATE	IC 7404	1
4.	IC TRAINER KIT	-	1

**THEORY:**

**MULTIPLEXER:**

Multiplexer means transmitting a large number of information units over a smaller number of channels or lines. A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally there are  $2^n$  input lines and  $n$  selection lines whose bit combination determines which input is selected.

**DEMULTIPLEXER:**

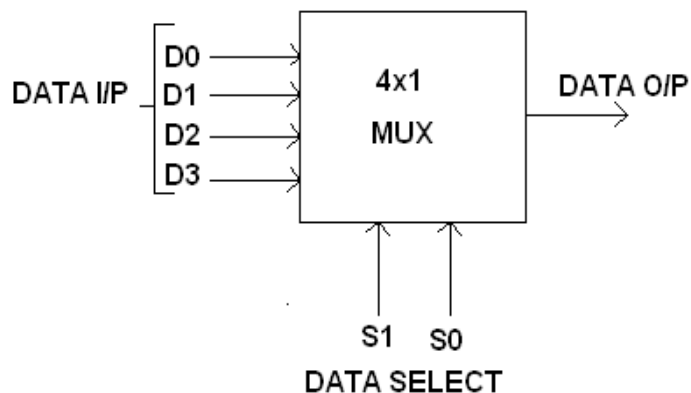
The function of the Demultiplexer is in contrast to the multiplexer function. It takes information from one line and distributes it to a given number of output lines. For this reason, the demultiplexer is also known as a data distributor. Decoder can also be used as a demultiplexer. In the 1: 4 demultiplexer circuit, the data input line goes to all of the AND gates. The data select lines enable only one gate at a time and the data on the data input line will pass through the selected gate to the associated data output line.

**PROCEDURE**

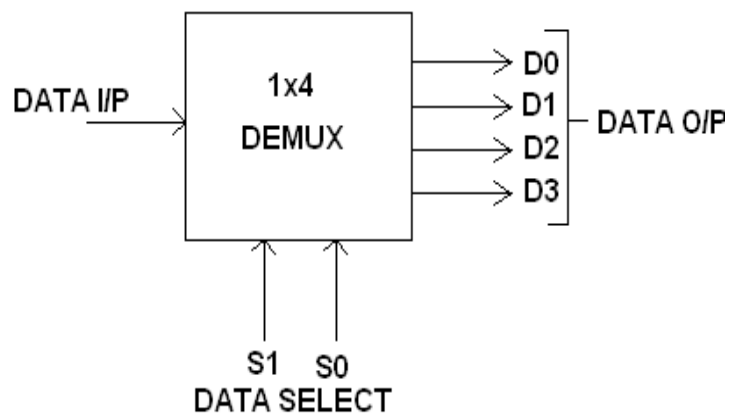
1. Select the type of Mux [  $2^{n-1}$  to 1 ].
2. Select (n-1) as select line and other input connects as input for MUX.
3. Select the type of Demux, one input,  $n$  select lines and  $2^n$  outputs.

4. Apply suitable select lines, to get the corresponding outputs.
5. Write the truth table and verify the outputs for different inputs.

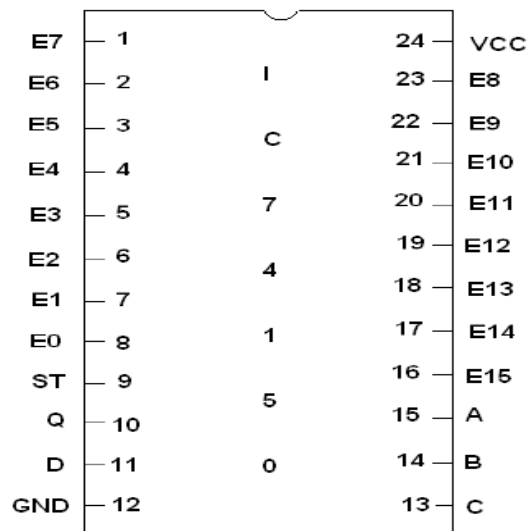
**BLOCK DIAGRAM FOR 4:1 MULTIPLEXER:**



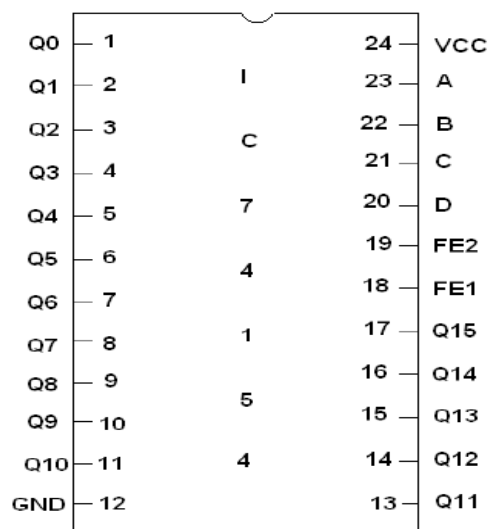
**BLOCK DIAGRAM FOR 1:4 DEMULTIPLEXER:**



**PIN DIAGRAM FOR IC 74150:**



**PIN DIAGRAM FOR IC 74154:**



**RESULT:** The circuit for multiplexer and demultiplexer using logic gates were set up and the output for various combinations of input are observed.

# PART B

## Basys 3™ FPGA Board- Overview

The Basys 3 board is a complete, ready-to-use digital circuit development platform based on the latest Artix®-7 Field Programmable Gate Array (FPGA) from Xilinx®. With its high-capacity FPGA (Xilinx part number XC7A35T- 1CPG236C), low overall cost, and collection of USB, VGA, and other ports, the Basys 3 can host designs ranging from introductory combinational circuits to complex sequential circuits like embedded processors and controllers. It includes enough switches, LEDs, and other I/O devices to allow a large number of designs to be completed without the need for any additional hardware, and enough uncommitted FPGA I/O pins to allow designs to be expanded using Digilent Pmods or other custom boards and circuits.

The Artix-7 FPGA is optimized for high performance logic, and offers more capacity, higher performance, and more resources than earlier designs. Artix-7 35T features include:

Figure 1. Basys 3 FPGA board with callouts.

Figure 2. Basys 3 FPGA board with callouts.

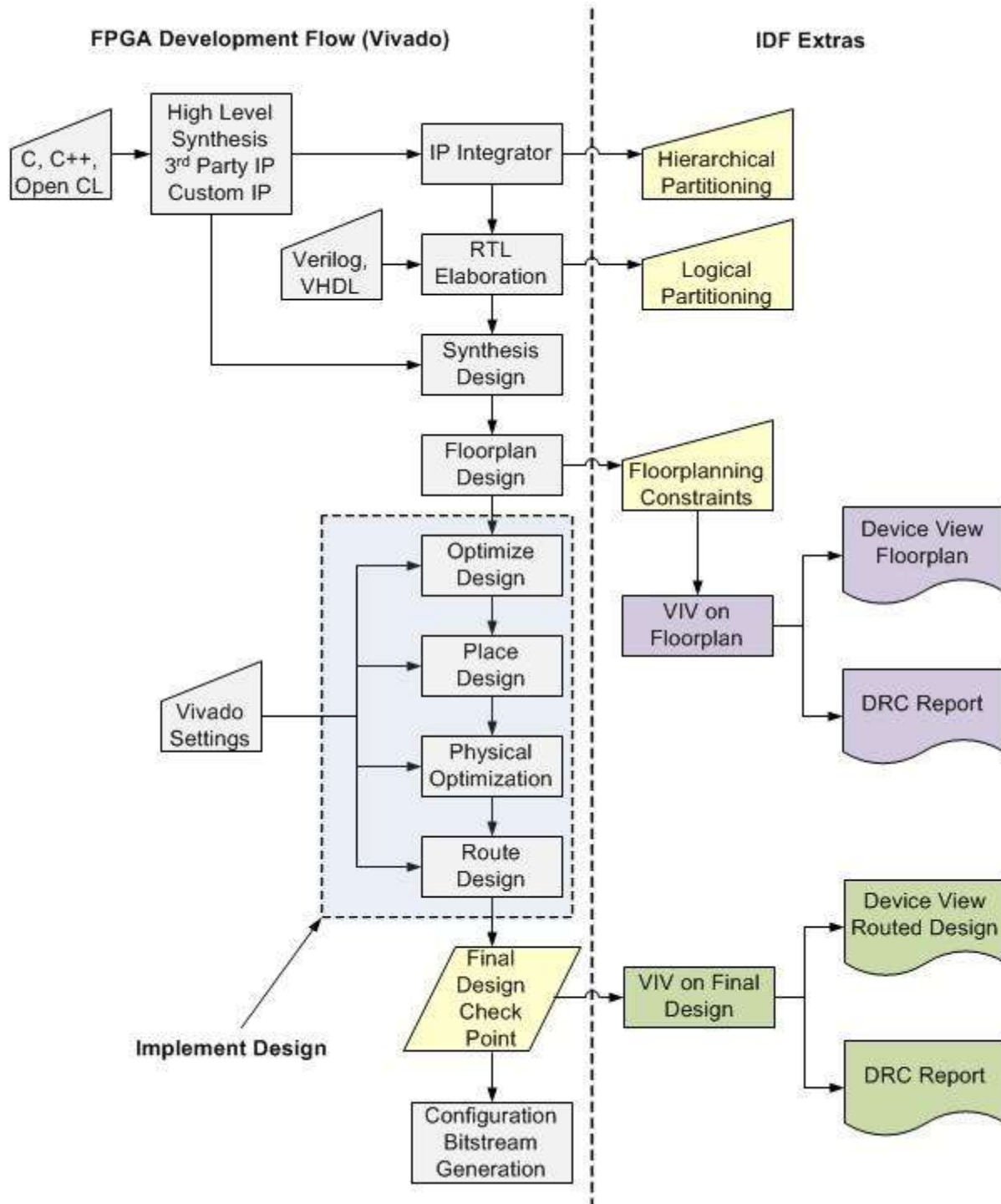
33,280 logic cells in 5200 slices (each slice contains four 6-input LUTs and 8 flip-flops)  
 1,800 Kbits of fast block RAM.  
 Five clock management tiles, each with a phase-locked loop (PLL).  
 90 DSP slices.  
 Internal clock speeds exceeding 450MHz .

Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod port(s)	10	Programming mode jumper
3	Analog signal Pmod port (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

Table 1. Basys 3 Callouts and component descriptions.



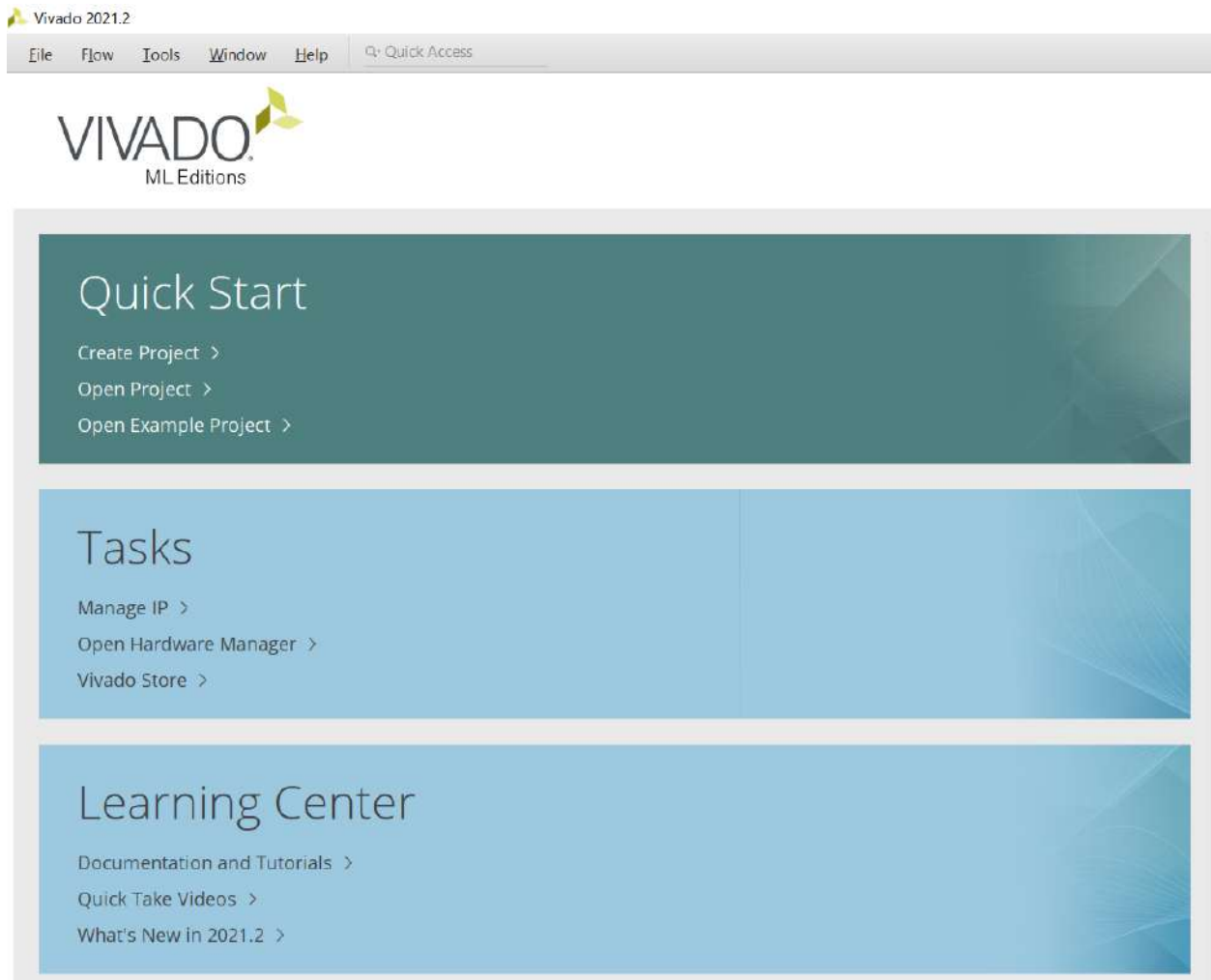
## **Xilinx Vivado Design Flow**



## Introduction to Xilinx's FPGA Vivado MLSoftware

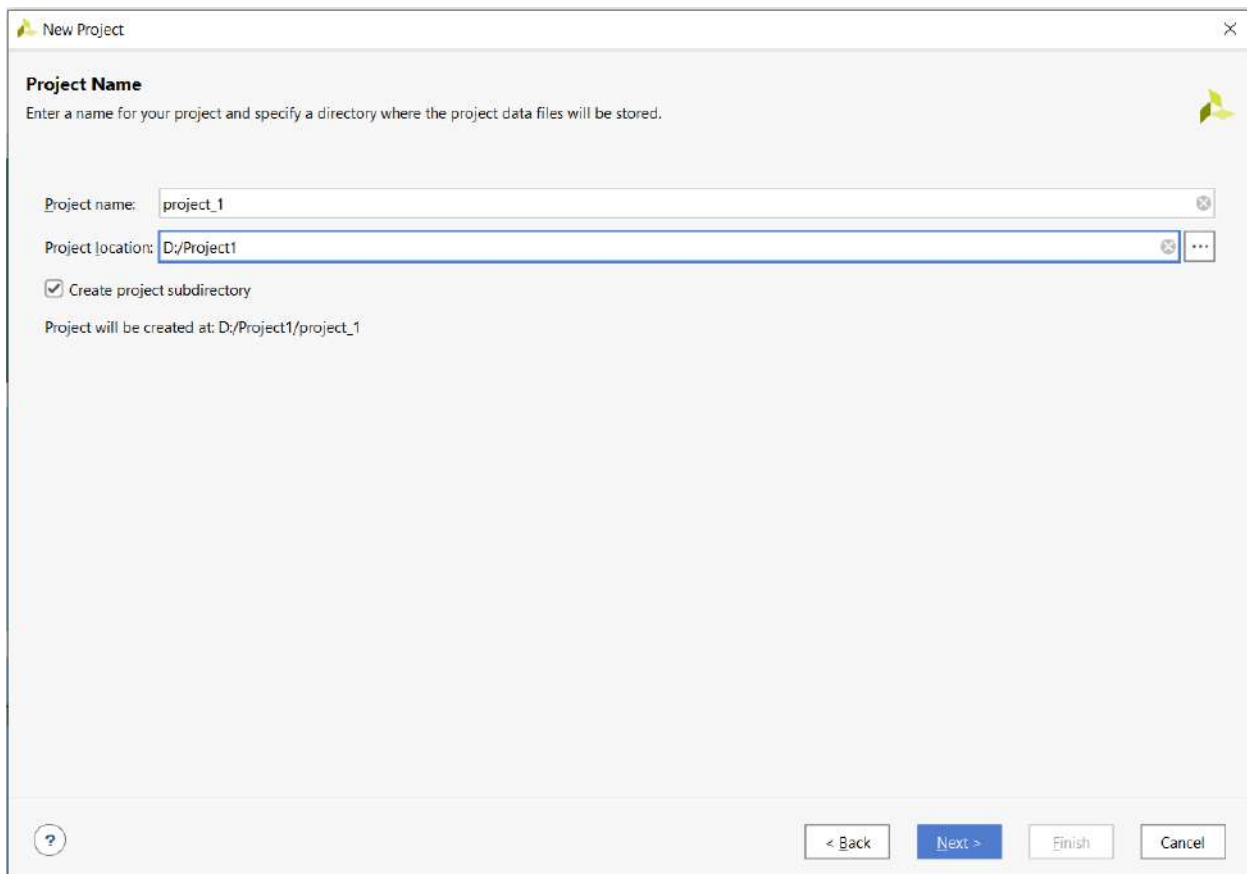
In this part, you will use Xilinx's Vivado to design, simulate and implement a simple 2 input AND gate digital circuit. Once completed, this 2 input AND gate implementation will be downloaded to the BASYS board and then tested using the on board LED's and switches.

**STEP 1:** Double click on the Vivado icon on your desktop to open up the welcome window of the development tool (as shown below). Three main sections can be observed in this window: “Quick Start”, “Tasks”, and “Learning Center”.



STEP 2: Now, click on “Create Project” to create a new project. You have to be careful about where to save your project file in the computer lab. The computers in the lab run a hard disk protection program that could interfere with Xilinx. So, if you save your project in a preserved folder, Xilinx might have problem with running the simulation. You have two choices: (1) either save the project directly on your USB flash disk. This option is good since your USB disk have normal read/write access and Xilinx runs correctly. However, this option can be slow for USB flash disks. The option (2) is to save the project in a folder that's in the desktop. Then, compress the folder into a ZIP file and email it to yourself. Start by creating a folder on the desktop called 'Lab\_1'. Create this folder in Windows, not from Xilinx. Then, in Xilinx, create a new project inside “**Project1**”. Name your project, '**Project\_1**' and it will be in the folder

“**D:/Project1**”. When you finish your lab, you can copy your project on your flash disk.



New Project

**Project Name**  
Enter a name for your project and specify a directory where the project data files will be stored.

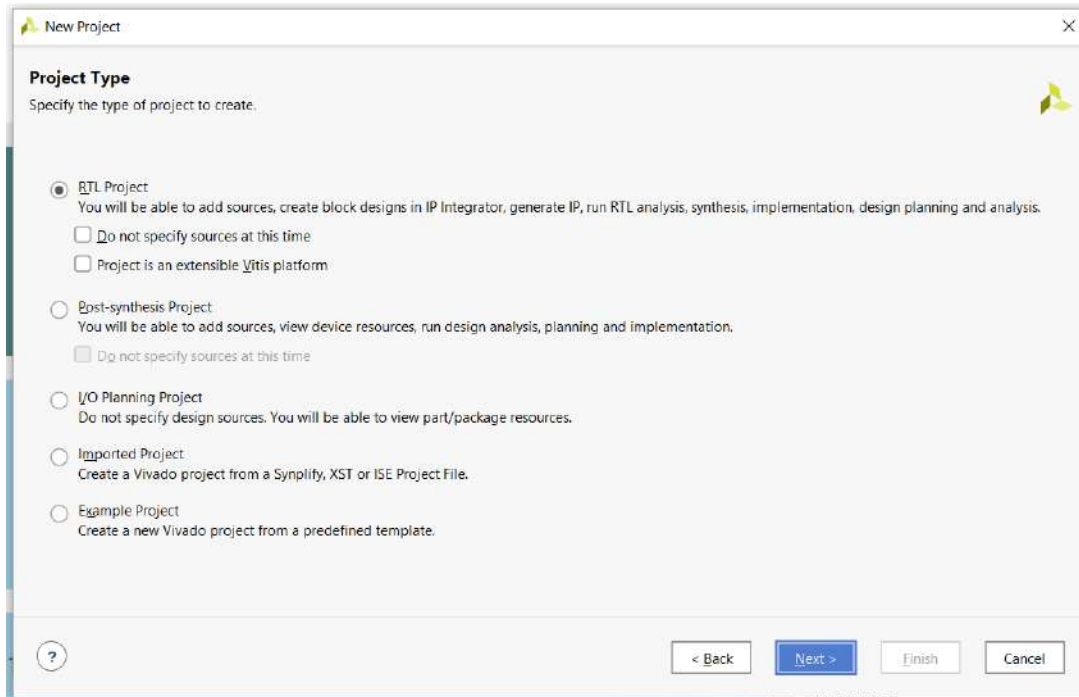
Project name:

Project location:

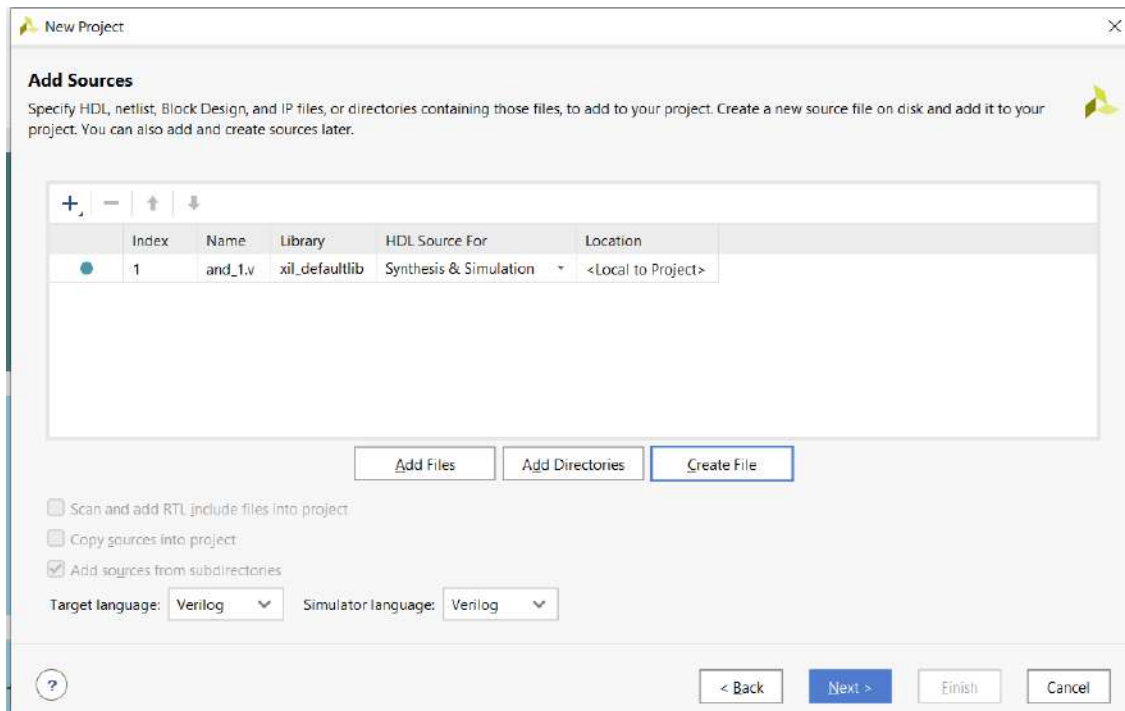
☒ Create project subdirectory

Project will be created at: D:/Project1/project\_1

**STEP 3:** In the next window, choose “RTL Project” as the project type. You can the description of this type in the window.



**STEP 4:** In the opened window, you can create source file (Verilog/Verilog Header/SystemVerilog/VHDL/Memory File) for your new project or add sources from the existing projects. Click on “Create File”, and in the opened window choose “Verilog” for the “File type”, write a name for your file (“and\_1”), and click on “Ok”. Continue clicking on Next until reaching the “Default Part” window



**New Project**

**Add Sources**

Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

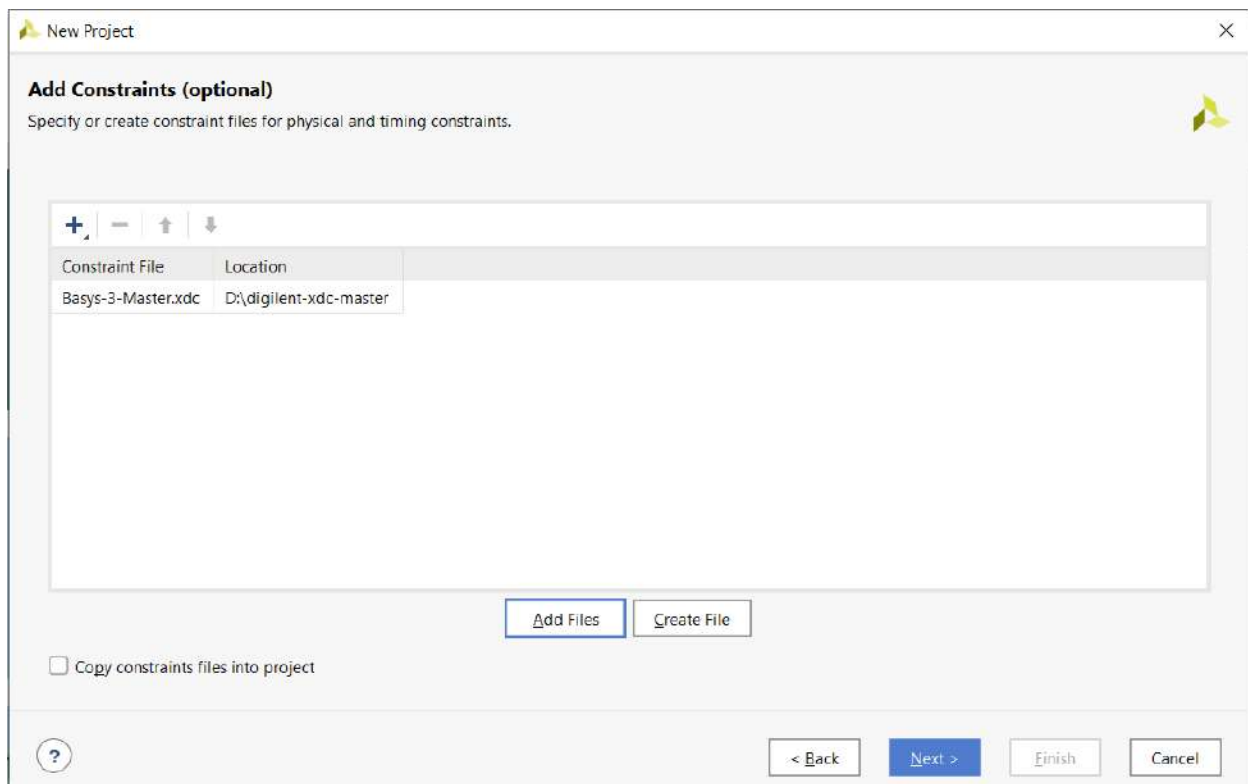
	Index	Name	Library	HDL Source For	Location
	1	and_1.v	xil_defaultlib	Synthesis & Simulation	<Local to Project>

☐ Scan and add RTL include files into project

☐ Copy sources into project

☒ Add sources from subdirectories

Target language: Verilog Simulator language: Verilog

**STEP 5: Add constraints file for Basys3 FPGA board**

**New Project**

**Add Constraints (optional)**

Specify or create constraint files for physical and timing constraints.

Constraint File	Location
Basys-3-Master.xdc	D:\digilent-xdc-master

☐ Copy constraints files into project

**STEP 6:** In this window, choose “Artix-7” for the “Family”, “-1” for “Speed grade”, and “cpg236” for “Package”. In the shown parts, select “xc7a35tcpg236-1”. Take a look at the configuration of this part for your own familiarity. If the board files are installed then the board files are installed then ou can select Basys3 directly from boards

**New Project**

**Default Part**  
Choose a default Xilinx part or board for your project. This can be changed later.

Select: ☒ Parts ☐ Boards

**Filter**

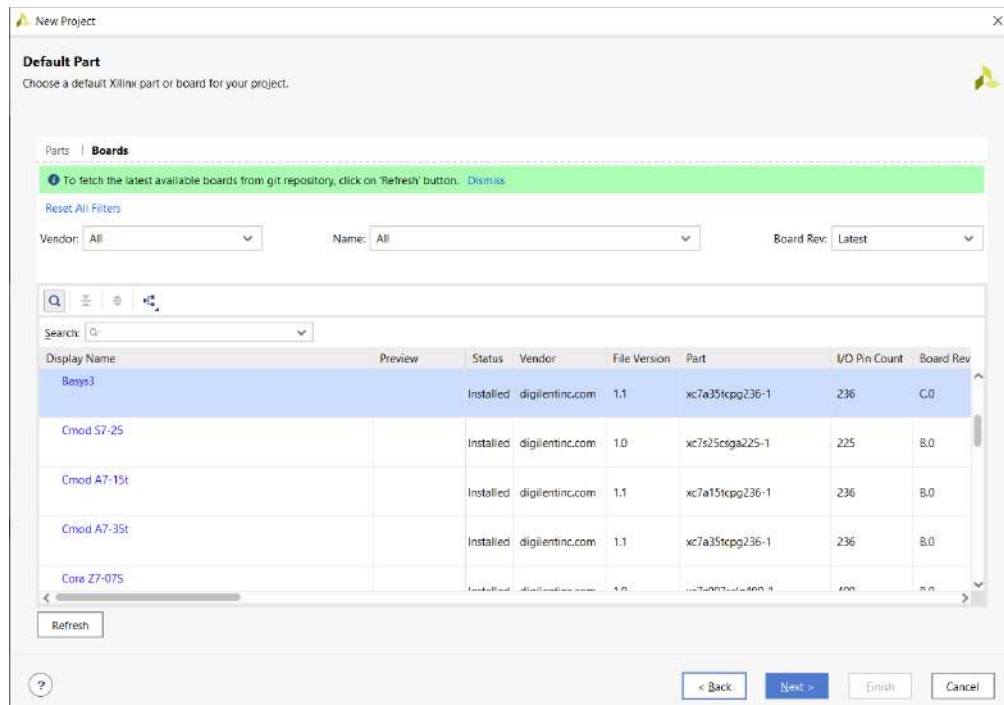
Product category: All Speed grade: -1  
Family: Artix-7 Temp grade: All Remaining  
Package: cpg236

Reset All Filters

Search:

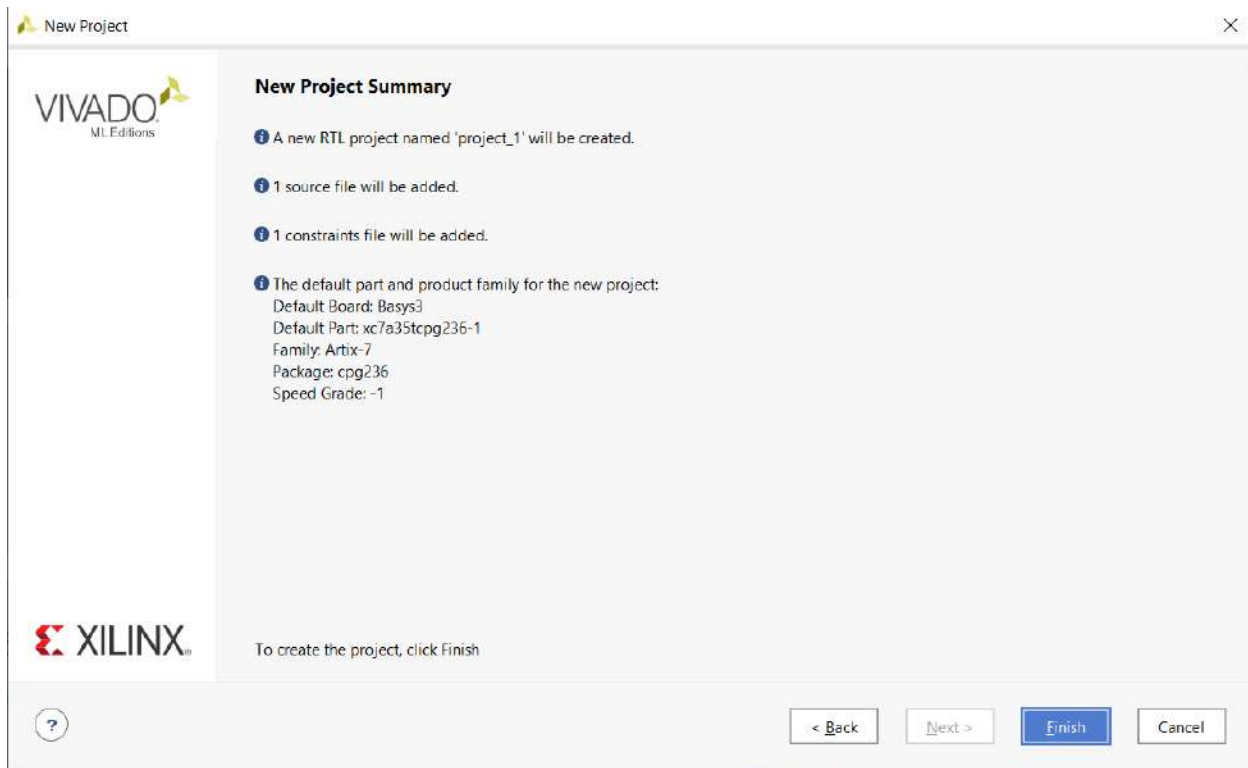
Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transceivers	GTPE2 Transceiv
xc7a15tcpg236-1	236	106	10400	20800	25	0	45	2	2
xc7a35tcpg236-1	236	106	20800	41600	50	0	90	2	2
xc7a50tcpg236-1	236	106	32000	65200	75	0	120	2	2

< Back Next > Finish Cancel

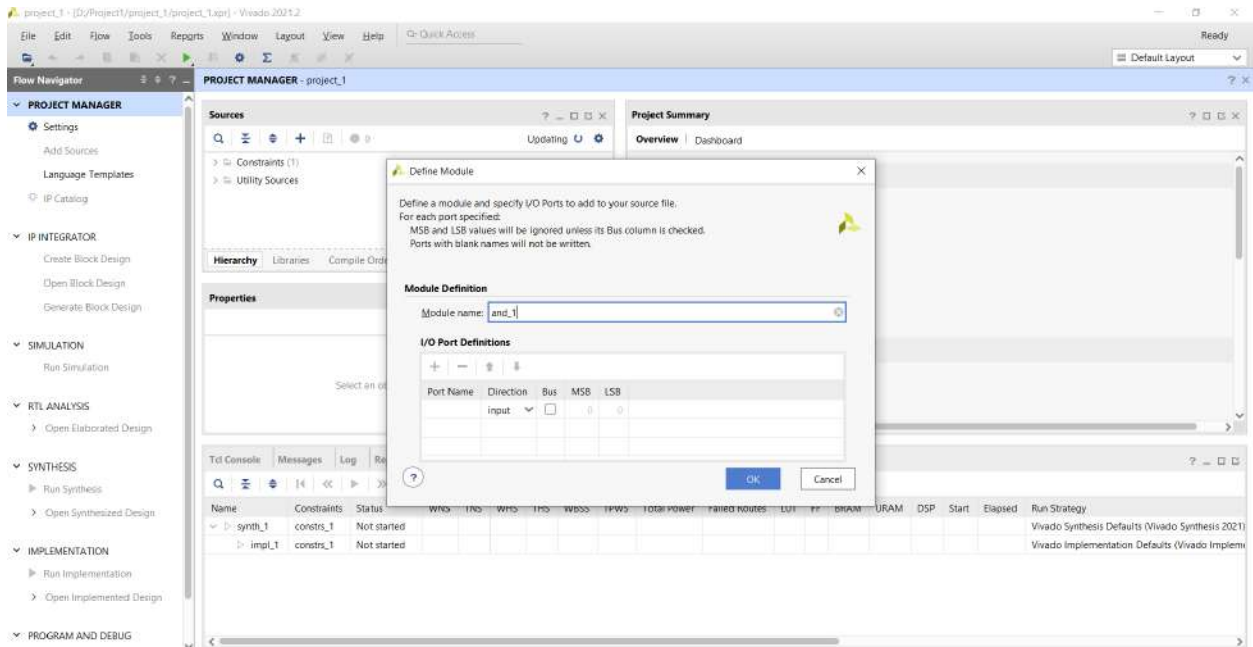


STEP 7: Look at your new project summary.



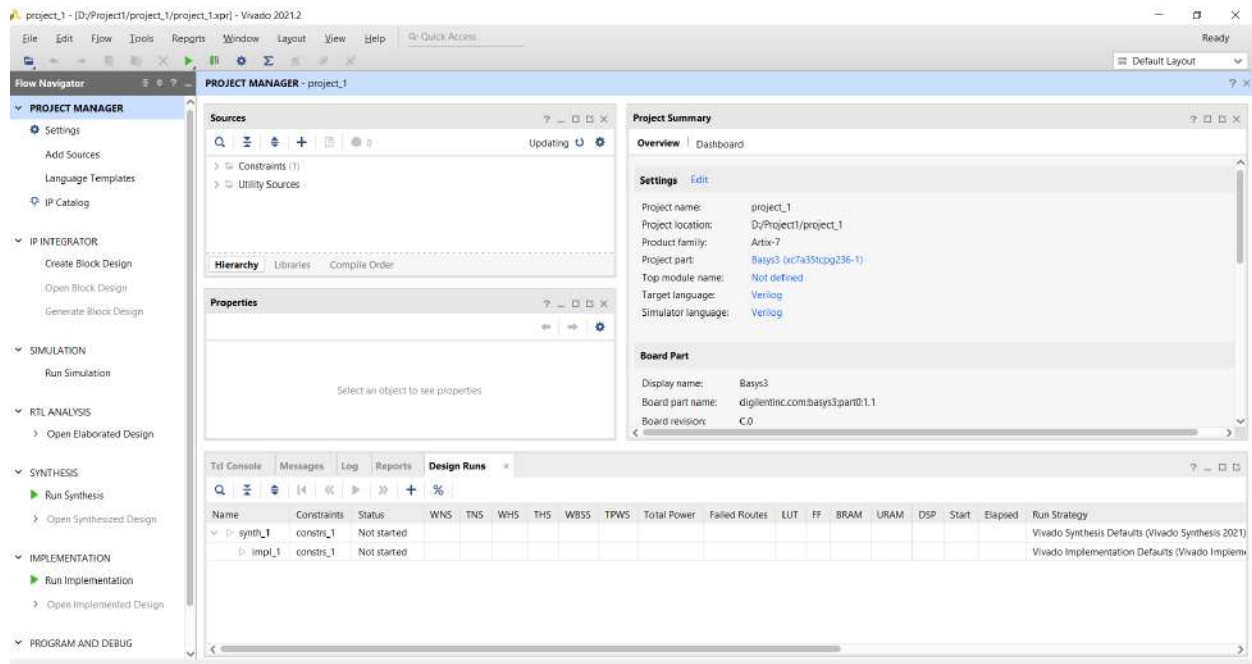


STEP 8: Define the input and the output ports of your module according to the shown window.

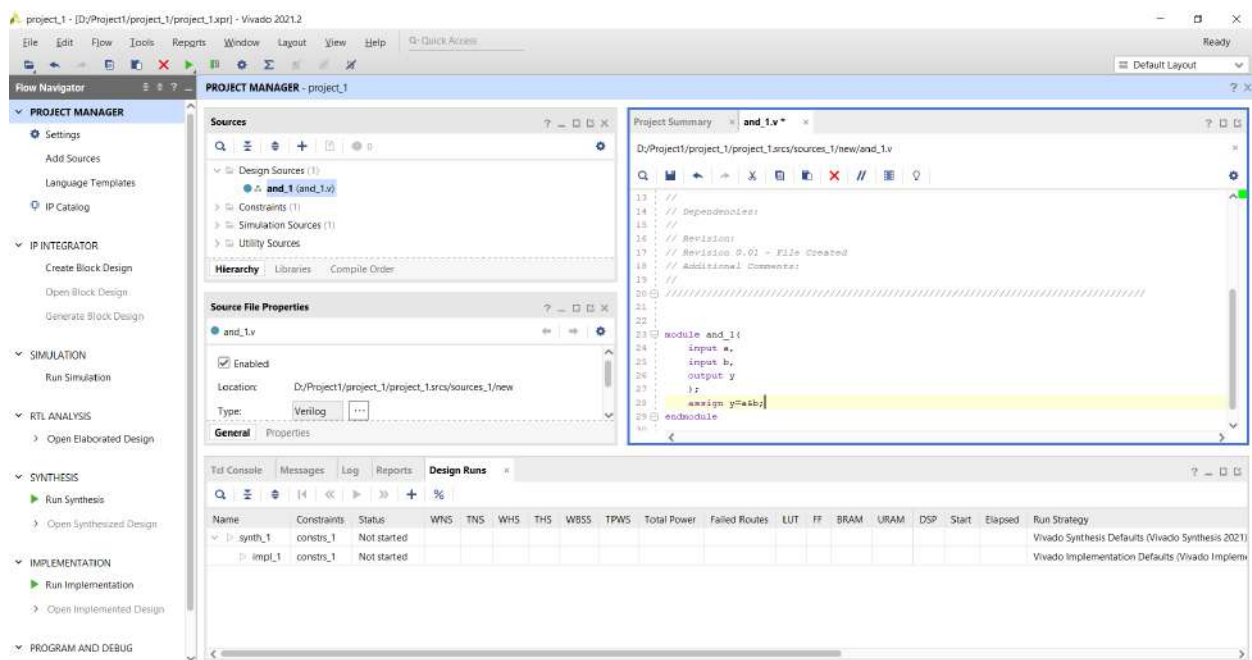


STEP 9: The opened window is the main environment for your project that is called “Project Manager”. You can explore it by seeing the options of each category in the toolbar on top of the window. In the left side, you can see the “Settings”, “Add Sources”, “Language Template”, “IP Catalog”, “IP Integrator”, “Simulation”, “RTL Analysis”, “Synthesis”, “Implementation”, and “Program and Debug”. Each of these serves a part of the digital design flow. In the middle, you can see the windows for “Sources”, “Properties”, “Project Summary”, and the reports and summaries for the execution of the project files.

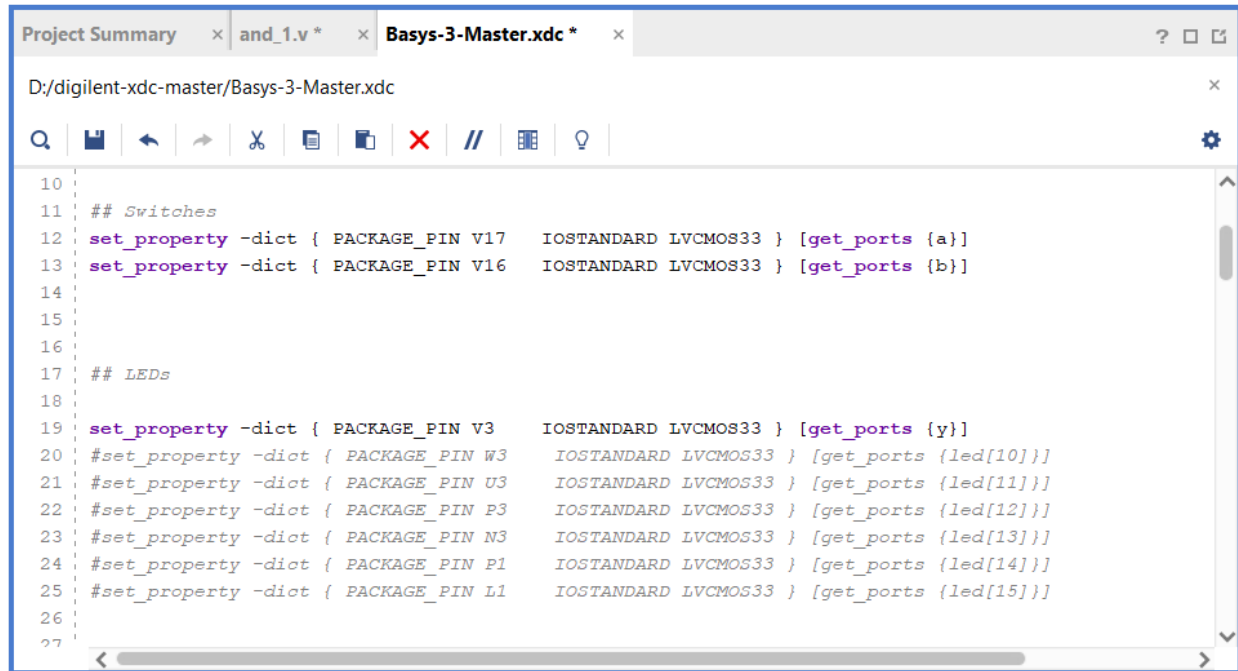
## ECL203 LOGIC CIRCUIT DESIGN LAB



STEP 10: Double click on the “and\_1.v” file (\*.v) in the “Sources” window. The VERILOG source file appears where the window is located in right side. Note that the module shows the defined inputs and outputs that were selected previously. Add you program to the file and then save.



STEP 11: Under the tab 'Constraints' select Basys-3-Master.xdc and edit the file for switches and led.



```
10
11 ## Switches
12 set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports {a}]
13 set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports {b}]
14
15
16
17 ## LEDs
18
19 set_property -dict { PACKAGE_PIN V3 IOSTANDARD LVCMOS33 } [get_ports {y}]
20 #set_property -dict { PACKAGE_PIN W3 IOSTANDARD LVCMOS33 } [get_ports {led[10]}]
21 #set_property -dict { PACKAGE_PIN U3 IOSTANDARD LVCMOS33 } [get_ports {led[11]}]
22 #set_property -dict { PACKAGE_PIN P3 IOSTANDARD LVCMOS33 } [get_ports {led[12]}]
23 #set_property -dict { PACKAGE_PIN N3 IOSTANDARD LVCMOS33 } [get_ports {led[13]}]
24 #set_property -dict { PACKAGE_PIN P1 IOSTANDARD LVCMOS33 } [get_ports {led[14]}]
25 #set_property -dict { PACKAGE_PIN L1 IOSTANDARD LVCMOS33 } [get_ports {led[15]}]
26
27
```

STEP 13: Now that the design is finished, you must build the project. Click **Run Synthesis** on the left-hand menu towards the bottom, on successful completion click on **Run Implementation**. On successful completion of implementation click on **Generate Bitstream** in the dialog box.

project\_1 - [D:/Project1/project\_1/project\_1.xpr] - Vivado 2021.2

File Edit Flow Tools Reports Window Layout View Help Q Quick Access

**Flow Navigator**

- Add Sources
- Language Templates
- IP Catalog
- IP INTEGRATOR
  - Create Block Design
  - Open Block Design
  - Generate Block Design
- SIMULATION
  - Run Simulation
- RTL ANALYSIS
  - Open Elaborated Design
- SYNTHESIS
  - Run Synthesis
  - Open Synthesized Design
- IMPLEMENTATION
  - Run Implementation
  - Open Implemented Design
- PROGRAM AND DEBUG
  - Generate Bitstream
  - Open Hardware Manager

**PROJECT MANAGER - project\_1**

**Sources**

Design Sources (1)

- and\_1 (and\_1.v)

Constraints (1)

- constrs\_1 (1)
  - Basys-3-Master.xdc

**Hierarchy** Libraries Compile Order

**Source File Properties**

Basys-3-Master.xdc

☒ Enabled

Location: D:/digilent-xdc-master

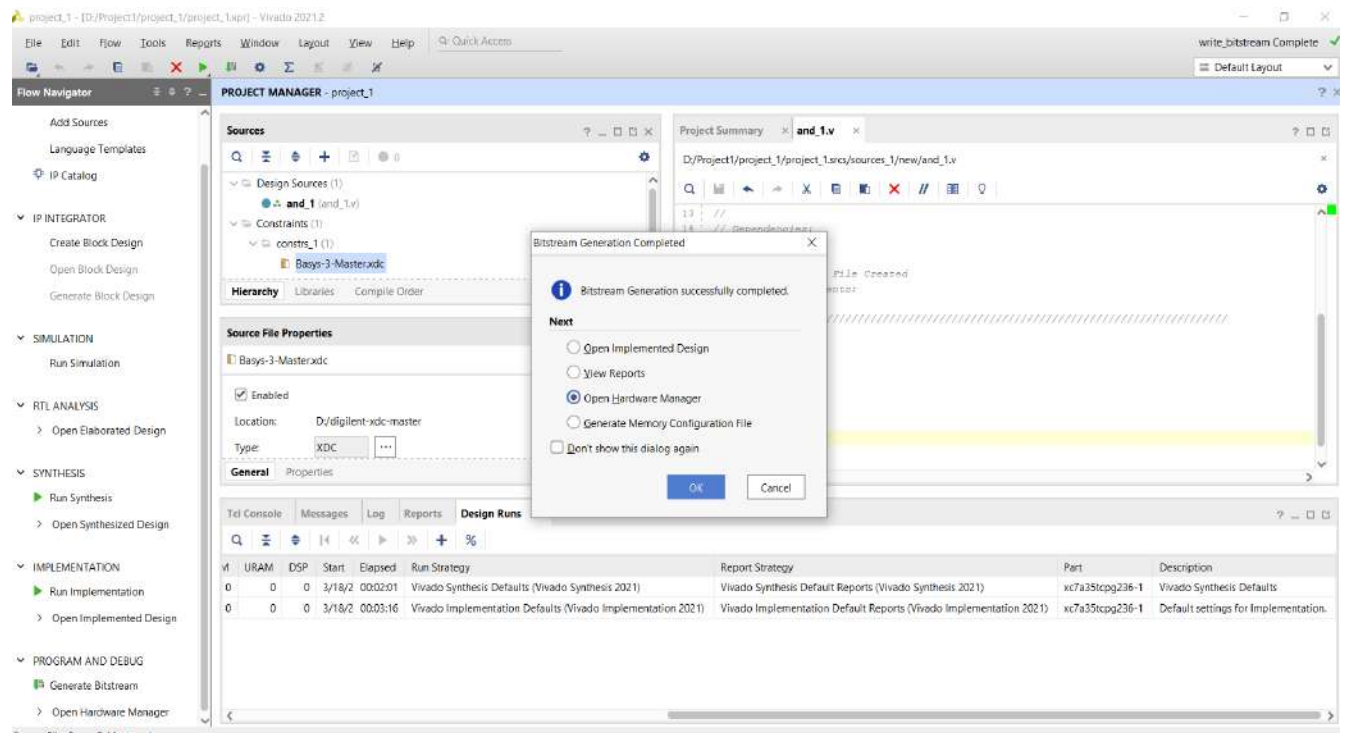
Type: XDC

**General** Properties

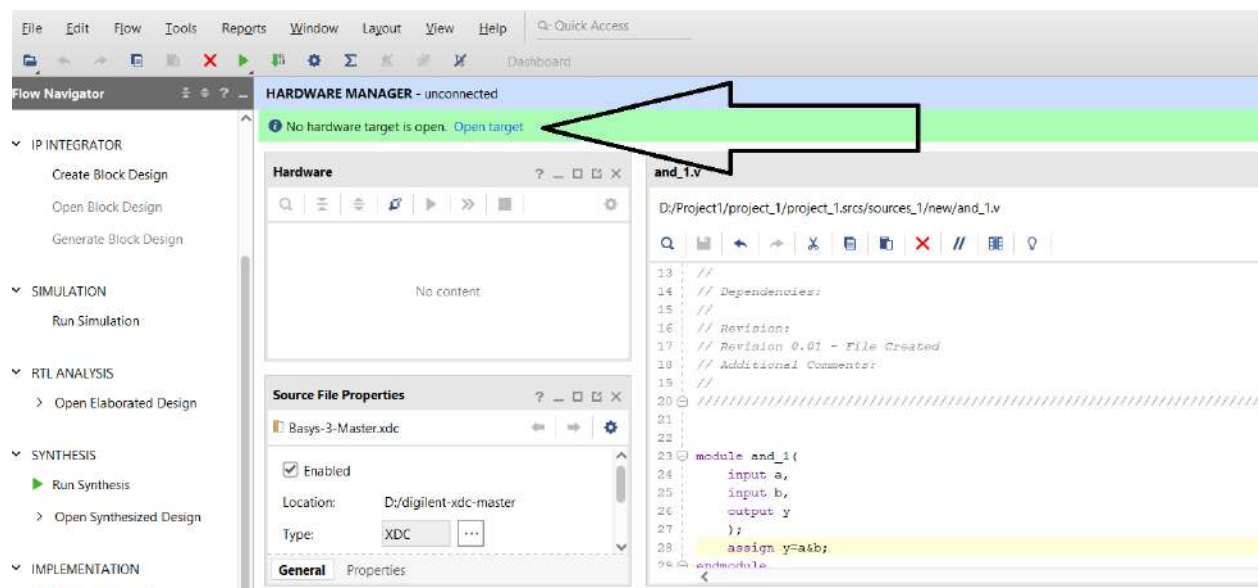
**Tcl Console** Messages Log Reports **Design Runs**

Name	Constraints	Status	WNS	TNS	WHS
synth_1	constrs_1	Not started			
impl_1	constrs_1	Not started			

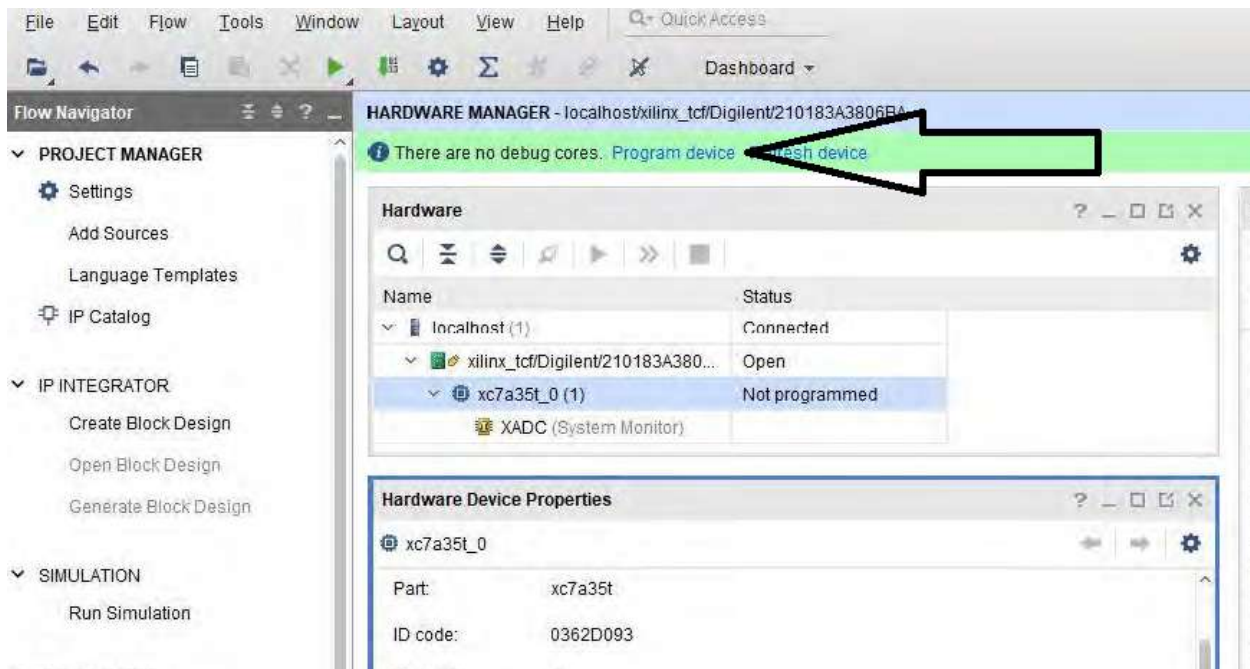
STEP 14: After completion of Bitstream file open **hardware manager** for downloading the program to FPGA.



Click on Open target



Click on program device



After successful completion of the above steps, you will be able to verify the functionality of AND gate by checking all possible combination on the switches.

**Expt No: 1    REALIZATION OF LOGIC GATES AND FAMILIARIZATION OF FPGAs****Date:**

- Aim:** (a) Familiarization of a basys 3 FPGA board and its ports and interface.  
(b) Create the bitstream files for your FPGA board.  
(c) Familiarization of the basic syntax of verilog  
(d) Development of verilog modules for basic gates, synthesis and implementation in the above FPGA to verify the truth tables.  
(e) Verify the universality and non-associativity of NAND and NOR gates by uploading the corresponding verilog files to the FPGA boards.  
(f) Verify the functionality of each gates by assigning the inputs to switches in FPGA and outputs to LEDs.

**Components and Equipments Required:**

Sl No	Components and equipments	Specification	Software	Quantity
1	FPGA	Basys 3	Xilinx Vivado	1

**Procedure:**

1. Write a Verilog module to realize a basic gate.
2. Write a test bench to verify the design.
3. Simulate the test bench and verify the design.
4. Synthesize the design, implement in FPGA and verify the outputs.

**RESULT:** Verilog code for basic gates were familiarized and the following codes were synthesized and bitstream file was generated and dumped to FPGA and output was verified.



**Expt No: 2****ADDERS IN VERILOG****Date:****Aim:**

- (a) Development of Verilog modules for half adder in 3 modeling styles (dataflow/structural/behavioral).
- (b) Development of Verilog modules for full adder in structural modeling using half adder.

**Components and Equipments Required:**

Sl No	Components and equipments	Specification	Software	Quantity
1	FPGA	Basys 3	Xilinx Vivado	1

**Procedure:**

1. Write the Verilog module to realize half adder in three different styles.
2. Write a test bench to verify the design.
3. Simulate the test bench and verify the design.
4. Synthesize the design, implement in FPGA and verify the outputs and timing of output.
5. Realize Full Adder using half adder and perform above steps.

**RESULT:** Verilog modules for half adder and full adder in 3 modeling styles were familiarized.

**Expt No: 3****MUX AND DEMUX IN VERILOG**

**Date:**

**Aim:**

(a) Development of verilog modules for a 4x1 and 8x1 MUX.

(b) Development of verilog modules for a 1x4 DEMUX.

**Components and Equipments Required:**

Sl No	Components and equipments	Specification	Software	Quantity
1	FPGA	Basys 3	Xilinx Vivado	1

**Procedure:**

1. Write the Verilog module to realize 4x1 MUX and 1x4 DEMUX.
2. Write a test bench to verify the design.
3. Simulate the test bench and verify the design.
4. Synthesize the design, implement in FPGA and verify the outputs.

**RESULT:** Developed verilog modules for MUX and DEMUX and observed its outputs.

**Expt No: 4**

**FLIP FLOPS AND COUNTERS**

**Date:**

**Aim:**

- (a) Development of verilog modules for SR, JK, T and D flipflops.  
(b) Development of verilog modules for a binary decade/Johnson/Ring counters.

**Components and Equipments Required:**

Sl N o	Components and equipments	Specification	Software	Quantity
1	FPGA	Basys 3	Xilinx Vivado	1

**Procedure:**

1. Write the Verilog module to realize the counters
2. Write a test bench to verify the design.
3. Simulate the test bench and verify the design.

**RESULT:** Developed verilog modules for SR, D, JK, D, T flipflops and binary decade/Johnson/Ring counters and observed the outputs.

**Expt No: 5**

**Flip-Flops and their Conversion in FPGA**

**Date:**

**Aim:**

- (a) Make gate level designs of J-K, J-K master-slave, T and D flip-flops, implement and test them on the FPGA board.
- (b) Implement and test the conversions such as T to D, D to T, J-K to T and J-K to D

**Components and Equipments Required:**

Sl No	Components and equipments	Specification	Software	Quantity
1	FPGA	Basys 3	Xilinx Vivado	1

**Procedure:**

1. Write the Verilog module to realize the flip flops.
2. Write a test bench to verify the design.
3. Simulate the test bench and verify the design.
4. Synthesize the design, implement in FPGA and verify the outputs and timing of output..

**RESULT:** Developed verilog modules for flip flops and their conversions and observed its outputs. Output for different combinations of input are verified using behavioral simulation and also generated bit stream then is dumped to FPGA and outputs verified

**Expt No: 6**

**BCD to Seven Segment Decoder in FPGA**

**Date:**

**Aim:**

- (a) Make a gate level design of a seven segment decoder, write to FPGA and test its functionality.
- (b) Test it with switches and seven segment display. Use output ports for connection to the display

**Components and Equipments Required:**

Sl No	Components and equipments	Specification	Software	Quantity
1	FPGA	Basys 3	Xilinx Vivado	1

**Procedure:**

1. Write the Verilog module to realize the seven segment decoder
2. Write a test bench to verify the design.
3. Simulate the test bench and verify the design.
4. Synthesize the design, implement in FPGA and verify the outputs and timing of output..

**RESULT:** Developed verilog modules for seven segment decoder and observed its outputs. Implemented in FPGA and tested its functionality.

**Expt No: 7                      Asynchronous and Synchronous Counters in FPGA**

**Date:**

**Aim:**

- (a) Make a design of a 4-bit up down ripple counter using T-flip-flops in the previous experiment, implement and test them on the FPGA board.

(b) Make a design of a 4-bit up down synchronous counter using T-flip-flops in the previous experiment, implement and test them on the FPGABoard.

**Components and Equipments Required:**

Sl No	Components and equipments	Specification	Software	Quantity
1	FPGA	Basys 3	Xilinx Vivado	1

**Procedure:**

5. Write the Verilog module to realize the 4-bit up down ripple counter and 4-bit up down synchronous counter.
6. Write a test bench to verify the design.
7. Simulate the test bench and verify the design.
8. Synthesize the design, implement in FPGA and verify the outputs and timing of output..

**RESULT:** Developed verilog modules for 4-bit up down ripple counter and 4-bit up down synchronous counter and observed its outputs. Implemented in FPGA and tested its functionality.